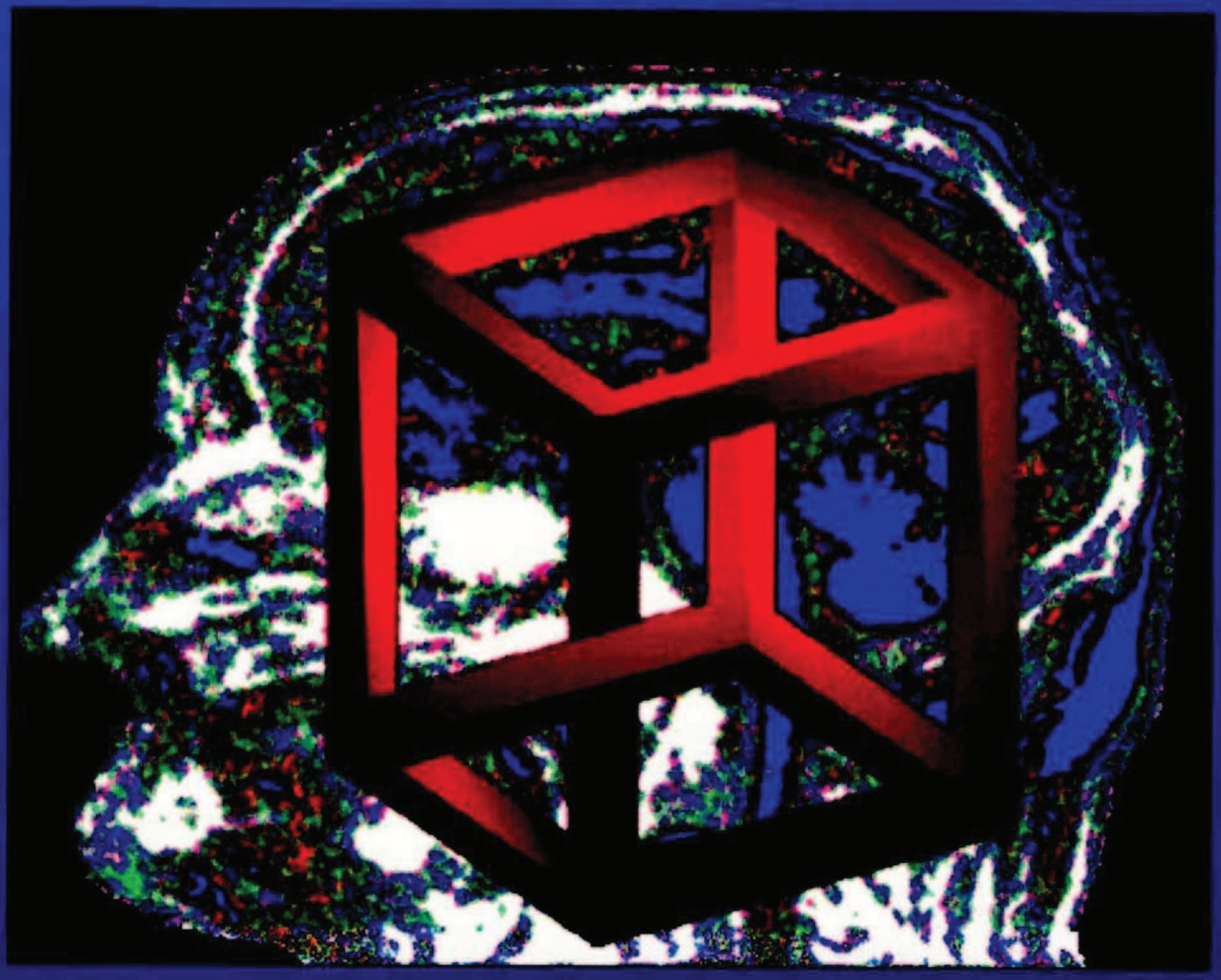
VYDAVATEĽSTVO SAMOSATO 2012

POČÍTAČOVÁ GRAFIKA A SPRACOVANIE OBRAZU

EUGEN RUŽICKÝ
ANDREJ FERKO



Vydavateľstvo SAMOSATO 2012

POČÍTAČOVÁ GRAFIKA

A

SPRACOVANIE OBRAZU

Eugen Ružický, Andrej Ferko

Prvá vysokoškolská učebnica počítačovej grafiky a spracovania obrazu

Záujem o štúdium počítačovej grafiky a spracovania obrazu neustále vzrastá. Autori podávajú výklad viacerých oblastí, ktoré sú buď klasické alebo dôležité perspektívne, s dôrazom na algoritmické riešenia. Táto kniha je určená širokému kruhu čitateľov, študentom, pedagógom a programátorom. Základné vzdelanie počítačovej grafike patrí k samozrejmej výbave každého absolventa počítačovo orientovaného smeru univerzitného štúdia.

Recenzenti: Doc. RNDr, Ľudovít Niepel, CSc., Ing. Martin Šperka, CSc.

Za odbornú a jazykovú stránku tejto vysokoškolskej učebnice zodpovedajú autori.

Odborný redaktor: Mgr. Pavol Eliáš

Eugen Ružický, Andrej Ferko

Počítačová grafika a spracovanie obrazu

Vydal SAMOSATO, s.r.o., Bratislava, v roku 2012.

Tretie, doplnené vydanie na DVD Virtuálny svet 2012.

(C) Eugen Ružický, Andrej Ferko 1995, 2012

ISBN 978 – 80 – 89464 – 13 – 5

EAN 9788089464135

Ľubke, Janke a Eugenovi

E. R.

Eve, Jurajovi a Michalovi

A. F.

Doplnenie vydania na DVD Virtuálny svet 2012

Vďaka vedeniu Avion Shopping Park v Bratislave vychádza Promo DVD vedeckej výstavy Virtuálny svet 2012. Cieľovou skupinou sú hostia výstavy, najmä najšikovnejší žiaci a študenti, ktorí získajú toto DVD za svoje riešenia priamo na tvorivých dielňach, ako aj ich učitelia. Rozhodli sme sa, že ako bonus venujeme pre toto DVD svoju učebnicu Počítačová grafika a spracovanie obrazu, symbolicky stredoškolskej mládeži, zadarmo. Našli sme pre tento krok porozumenie u pána vydavateľa, prof. Mirona Šramku, DrSc. i u našich zamestnávateľov, FMFI UK a Paneurópska univerzita v Bratislave. Digitálnu verziu starostlivo pripravila Mirka Valíková. Študijné programy počítačovej grafiky a geometrie v období od prvého vydania v roku 1995 rozvíjali najmä páni garanti doc. Valent Zaťko, doc. Miloš Božek, prof. Július Korbaš a doc. Roman Ďurikovič so svojimi splupracovníkmi. Všetkým, ktorí vychovali stovky úspešných absolventov, patrí naša úprimná vďaka.

Prirodzene, vznikla otázka, ako učebnicu z roku 1995 doplniť. Text ostal podľa našej mienky aktuálny vďaka dôrazu na algoritmické riešenia a klasické myšlienky medzinárodnej normotvorby a neustále sa podľa neho vyučujú viaceré predmety na FMFI UK. Zmenil sa však kontext. Zmenil sa o internet.

Oproti roku 1995 môžeme a musíme doplniť práve tento sieťový kontext. Napr. výklad jazyka na virtuálnu realitu VRML v češtine stačí ponúknuť cez odkaz na stiahnutie si dôkladnej učebnice prof. Jiřího Žáru http://www.cgg.cvut.cz/LaskavyPruvodce/ a v angličtine originálny tutoriál svetovej konferencie SIGGRAPH http://www.sdsc.edu/~nadeau/Courses/Siggraph98vrml/. Naša učebnica nezahŕňa teóriu texturovania ani počítačovú animáciu, no presný matematický model oboch ponúka po anglicky písaná učebnica nášho kolegu z Budapešti – prof. Lázsló Szirmay-Kalos, http://www.iit.bme.hu/~szirmay/book.html. Rozšírenú realitu vyučujeme čiastočne podľa knihy Spatial Augmented Reality od dvojice kolegov, prof. Oliver Bimber z Linza a prof. Ramesh Raskar z MIT ju tiež po dohode so šľachetným vydavateľom poskytli na stiahnutie: http://complexrhetoric.blogspot.com/2008/09/free-book-download-bimber-and-raskar.html. Najnovšie v češtine pribudla na sieti učebnica kolegov z Ostravy a Plzne, Eduard Sojka, Martin Němec a Tomáš Fabián: http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/zaklady_pocitacove_grafiky.pdf. Počítačová grafika sa vyučuje v mnohých predmetoch v Slovenskej republike i v Česku. Situáciu na Slovensku zmapovala vo svojej dizertácii Počítačová grafika pre učiteľov informatiky v roku 2005 pani doktorka Júlia Tomanová z Nitry. Pre mnohé oblasti existujú predmetové stránky, Moodle, e-learningové zdroje, na FMFI UK pg.netgraphics.sk, www.cagd.sk, dip.sccg.sk, e-matik... S úplne novou modalitou prišiel na sieť indický projekt videoprednášok, ktoré hodnotíme ako alternatívny výklad základov počítačovej grafiky – http://www.learnerstv.com/Free-Computers-Video-lectures-ltv046-Page1.htm – desiatky nafilmovaných prednášok v angličtine. Na konci knihy Real-time Rendering, ktorej komunita autorov počítačových hier žije okolo portálu www.realtimerendering.com, sa vtipne poznamenáva, že ťažko predpovedať, najmä budúcnosť. Pre počítačových grafikov blízku budúcnosť predpovedá Ke-Sen Huang z Taiwanu, http://kesen.realtimerendering.com/, tam možno nájsť vedecké články, ktoré výjdu v budúcnosti... Visual computing, spracovanie vizuálnej informácie, sa úspešne a explozívne rozvíja...

Ako teda ošetriť aj prípad, že v budúcnosti výjde výborné knižné alebo interaktívne dielo v blízkom či svetovom jazyku? Zriaďujeme preto stránku pgaso.sccg.sk, na ktorej budeme udržiavať aktuálne odkazy, errata a viaceré rozšírenia nášho výkladu. Inak povedané, perspektívne premeníme prítomný text na hypertextový živý portál.

Záujemcom o virtuálnu realitu a jej vedecké základy, predovšetkým počítačovú grafiku, geometrické modelovanie a počítačové videnie, želáme mnoho úspechov.

doc. RNDr. Eugen Ružický, CSc.
FI PEVS Bratislava

doc. RNDr. Andrej Ferko, PhD.
FMFI UK Bratislava

Predslov

Táto učebnica sa nazýva presne tak ako odborná komisia Medzinárodnej organizácie pre normalizáciu i ako novoustanovená katedra na MFF UK. Nie je to náhoda. Postupy i výsledky počítačovej grafiky a spracovania obrazu sa stávajú pre užívateľov miliónov osobných počítačov samozrejmosťou. Takýto stav však neprišiel bez enormného sústredenia výskumnej i výrobnej sféry v posledných desaťročiach. Len na najväčšej každoročnej konferencii SIGGRAPH sa v r. 1994 očakávalo do 35 tisíc účastníkov. Ročne vychádzajú desiatky tisíc strán o nových výsledkoch. Na trh sústavne vstupujú ďalšie hardverové i softverové novinky. Pokúsili sme sa popísať **základné pojmy a metódy počítačovej grafiky a spracovania obrazu**. U čitateľa sa predpokladajú základné matematické a informatické poznatky. Pri voľbe záberu i výkladu sme prihliadali ku všetkým nám dostupným prameňom, no i tak si nerobíme nárok na úplnosť a budeme vďační za každú pripomienku. Kniha sa člení na 20 kapitol, ktoré možno orientačne zhrnúť do štyroch celkov: Základy, Spracovanie obrazu, 3D grafika, Užívateľské rozhranie a grafické systémy.

A napokon nám pripadá milá povinnosť vyjadriť vďaku všetkým, ktorí tomuto projektu pomohli, často i viacerými spôsobmi. Kniha by nevyšla bez šľachetnosti a príspevku **Štátneho fondu kultúry PRO SLOVAKIA.** Časť obrázkov precízne nakreslila **Elena Šikudová**. Obdivuhodnú trpezlivosť s mnohými verziami textu prejavil pán Mgr. **Roman Kotrec**. Mnohými vylepšeniami prispel odborný redaktor Mgr. **Pavol Eliáš**, páni oponenti doc. **Ľudovít Niepel** a Ing. **Martin Šperka**, ako aj páni kolegovia RNDr. **Július Ertl**, doc. **Valent Zaťko** a doc. **Miloš Božek**. Tým všetkým, ale predovšetkým našim rodinám, patrí naša vďaka.

Bratislava, január 1995 Autori

Obsah

Úvod **1**
1.1. Ľudské oko a videnie 1
1.2. Základné pojmy 3
1.3. Štruktúra interaktívnej grafickej aplikácie 7
1.4. Vývoj počítačovej grafiky a spracovania obrazu 8
Transformácie v rovine a priestore **11**
2.1. Úvod 11
2.2. Matice 11
2.3. Otočenie a zmena mierky 12
2.4. Posunutie objektu a sústavy súradníc 14
2.5. Kompozícia 2-rozmerných transformácií 15
2.6. Iné transformácie 15
2.7. Zobrazenie okna na zobrazovacie pole 17
2.8. Pravotočivá sústava súradníc v priestore 18
2.9. Zmena mierky a posunutie 19
2.10. Otáčanie okolo súradnicových osí 20
2.11. Otočenie okolo ľubovoľnej osi 21
2.12. Iné transformácie v priestore 23
Orezávanie a prienik **25**
3.1. Úvod 25
3.2. Body v okne 25
3.3. Orezávanie úsečky 25
3.4. Prienik polroviny s mnohouholníkom 32
3.5. Prienik dvoch mnohouholníkov 35
3.6. Prístupy k orezávaniu textu 36
3.7. Záver 37
Krivky a plochy **39**
4.1. Vyjadrenie kriviek a plôch 39
4.2. Geometrické modelovanie kriviek 39
4.3. Geometrické modelovanie plôch 47
Rasterizácia kriviek a oblastí **55**
5.1. Úvod 55

5.2. Rastrový rozklad úsečky 55
5.3. Rastrový rozklad kružnice 60
5.4. Vyhladzovanie (antialiasing) a hrúbka čiary 62
5.5. Vypĺňanie oblasti 64
5.6. Rastrový rozklad mnohouholníka 70
Matematické základy spracovania obrazu 75
6.1. Úvod do Fourierovej transformácie 75
6.2. Diskrétna Fourierova transformácia 78
6.3. Niektoré vlastnosti Fourierovej transformácie 82
Jasová korekcia a filtrácia obrazu 87
7.1. Úvod 87
7.2. Jasová korekcia 87
7.3. Zlepšenie obrazu pomocou histogramu 89
7.4. Filtrácia 95
7.5. Vyhladenie obrazu 98
7.6. Ostrenie obrazu 102
Segmentácia a hranica obrazu 107
8.1. Úvod 107
8.2. Segmentácia prahovaním 107
8.3. Hranica a obrys binárneho obrazu 112
8.4. Segmentácia obrysom 120
8.5. Segmentácia narastaním oblasti 122
Morfologické transformácie a skelet 125
9.1. Úvod do morfológie 125
9.2. Konštrukcia elementárnych transformácií 126
9.3. Kostra (skelet) množiny 131
9.4. Základné pojmy 131
9.5. Algoritmy skeletovania 135
Metódy modelovania a zobrazovania 3D objektov 141
10.1. Úvod 141
10.2. Premietanie 141
10.3. Modelovanie 3D objektov 149
10.4. Kódovanie mnohostenov 154
10.5. Metódy zobrazovania 158
Určenie viditeľného povrchu 161
11.1. Úvod 161
11.2. Zjednodušenie stredového premietania a obálky 161
11.3. Algoritmus využívajúci z-bufer 163

11.4. Riadkovo skanovací algoritmus 164
11.5. Algoritmus hĺbkového triedenia 166
11.6. Warnockov algoritmus delenia okna 169
11.7. Algoritmus octree 172
11.8. Appelov algoritmus 174
11.9. Zobrazenie grafu funkcie dvoch premenných 176
Achromatické a farebné svetlo 179
12.1. Úvod 179
12.2. Základné fyzikálne vlastnosti svetla 179
12.3. Fyziologické vlastnosti svetla 181
12.4. Farebné modely 183
Osvetľovacie modely a tieňovanie 193
13.1. Úvod 193
13.2. Empirické osvetľovacie modely 194
13.3. Model svetla 197
13.4. Osvetlenie povrchu 199
13.5. Lokálne fyzikálne modely 201
13.6. Tieňovanie 203
Fotorealizmus 207
14.1. Úvod 207
14.2. Základný rekurzívny algoritmus sledovania lúča 208
14.3. Praktická realizácia algoritmu sledovania lúča 211
14.4. Ohraničujúce telesá 214
14.5. Priestorová koherencia 216
14.6. Radiačná metóda 219
14.7. Výpočet form faktorov 221
14.8. Implementácia intenzity vyžarovania 225
Textúry a fraktály 229
15.1. Mapovanie textúry 229
15.2. Fraktálne generovanie 234
Interakcia a oknové systémy 241
16.1. Grafické užívateľské rozhranie 241
16.2. Oknový systém 245
16.3. Spracovanie výstupu v oknovom systéme 247
16.4. Spracovanie vstupu v oknovom systéme 249
16.5. Príklady oknových systémov 250
16.6. Perspektívy 252
Normalizované grafické systémy 253

17.1. Normalizácia grafických systémov 253
17.2. Grafický výstup podľa normy GKS 255
17.3. Súradnicové systémy v GKS 261
17.4. Segmenty a ich atribúty 262
17.5. Logické vstupné zariadenia a režimy 263
17.6. Pracovné stanice 266
17.7. Ďalšia funkčnosť GKS 269
17.8. GKS-94 272
Kódovanie grafickej informácie 273
18.1. Kódovanie informácie 273
18.2. Prehľad normy CGM 274
18.3. Funkčná špecifikácia CGM 275
18.4. Kódovanie CGM znakmi a binárne 276
18.5. Niektoré ďalšie kódovania 276
PHIGS 279
19.1. Súvis PHIGS a GKS 279
19.2. Súradnicové systémy a transformácie 282
19.3. Grafický výstup 290
19.4. Centralizovaná pamäť štruktúr 295
19.5. Grafický vstup 298
19.6. Pracovné stanice 298
19.7. PHIGS PLUS 300
19.8. Syntéza PHIGS + X = PEX 301
Multimédiá 303
20.1. Architektúra multimediálneho systému 303
20.2. Spracovanie zvuku 305
20.3. PREMO 307
20.4. Dve multimediálne aplikácie 308
20.5. Najnovšie trendy 309
Literatúra 311

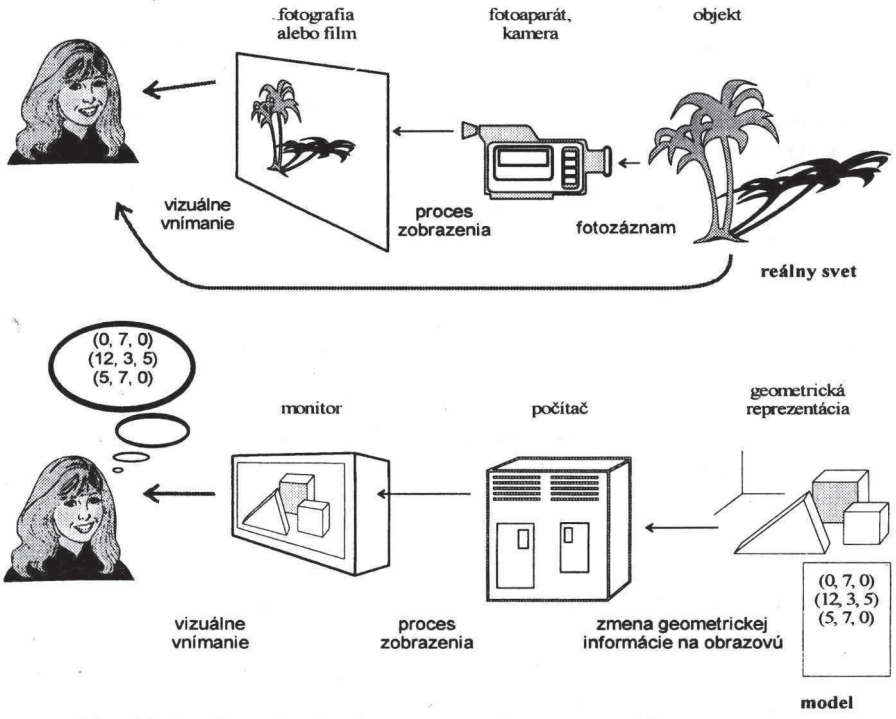
1

Úvod

Radšej raz vidieť ako tisíckrát počuť *(čínske príslovie)*

1.1 Ľudské oko a videnie

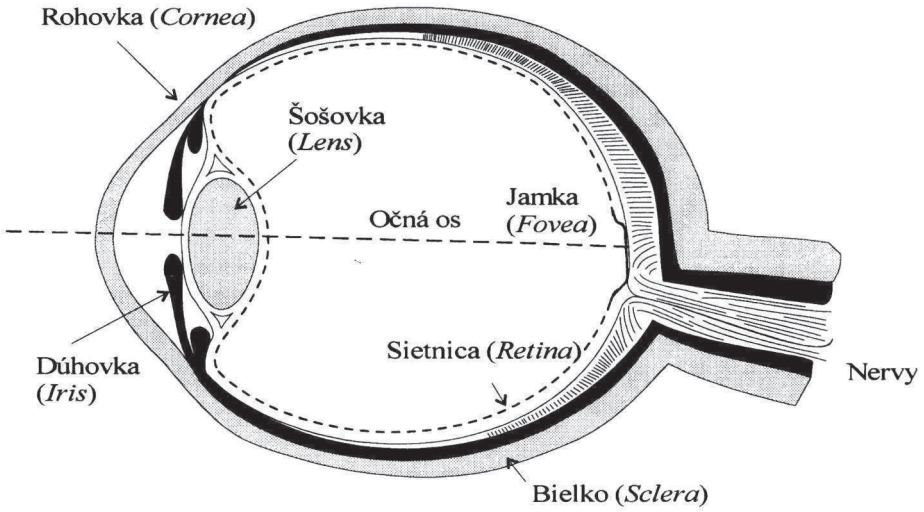
V tejto knihe sa pokúšame sprístupniť v slovenčine základy prepotrebnej oblasti - počítačovej grafiky a spracovania obrazu. Intuitívne je vytváranie obrázkov metódami počítačovej grafiky analógiou fotografovania. Postupy spracovania obrazu pripomínajú úpravy hotovej fotografie, napr. zvýraznenie kontrastov. Presné definície uvedieme neskôr. Najprv však treba uviesť niekoľko pojmov z oblasti ľudského videnia.



Obr. 1.1 Analógia fotografovania a počítačovej grafiky

Vizuálne vnímanie nám dáva dôležitú orientáciu v okolitom svete. Nadobudnuté informácie sa ukladajú v senzorickej pamäti na zlomky sekúnd v podobe akoby fotografie. V krátkodobej pamäti sa uchováva na niekoľko sekúnd zmysel naposledy spracovávanej informácie. Oba druhy informácie - **obrazová a obsahová** - môžu prejsť do dlhodobej pamäti, kde sa môžu zapamätať dlhodobo. Optické informácie sa vnímajú rýchlejšie a udržia sa v pamäti pevnejšie, lebo na rozdiel od abstraktných informácií môžu byť kódované dvojako - opticky i sémanticky. Vstup informácie sa začína v ľudskom oku.

Oko je párový zrakový orgán. Skladá sa z očnej gule a prídavných zrakových orgánov (viečok, mihalníc, spojiviek, slzného ústrojenstva a okohybných svalov). Očná guľa má priemer asi 20 mm. Vonkajšia časť oka sa skladá z dvoch membrán: rohovky (*cornea*) a bielka (*sclera*). Vnútorná časť mebrány je sietnica (*retina*). Sietnica má na optickej osi malú časť, ktorá sa líši od zvyšku, a volá sa jamka (*fovea*).

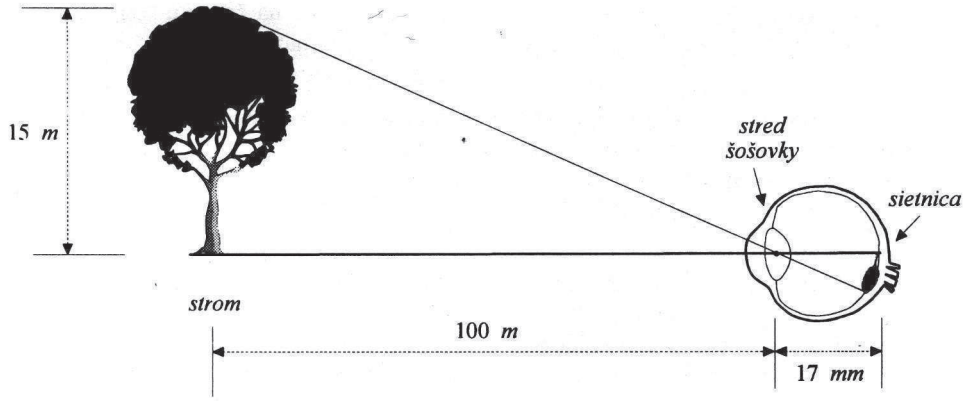


Obr. 1.2 Horizontálny rez ľudským okom

Intenzitu vstupujúceho svetla kontroluje oko sťahovaním strednej časti dúhovky (pupily). Receptory zrakového vnemu sú čapíkové a tyčinkové. Čapíkové bunky zabezpečujú **farebné (fotopické) zrakové vnímanie** a je ich približne 6 až 7 miliónov, rozmiestnených v oblasti fovea. **Vnímanie skotopické** sa uplatní zas pri slabom osvetlení, napr. za mesačnej noci a zabezpečujú ho tyčinkové zrakové bunky, rozptýlené viac po časti retiny. Týchto je podstatne viac - od 75 do 150 miliónov. Vnímanie v šere nazývame **mezopické.**

Pôsobenie svetla na tyčinky a čapíky vyvolá elektrický vzruch, ktorý sa šíri zrakovou dráhou do zrakového ústredia v centrálnej nervovej sústave, kde sa ďalej spracuje - ako sme už spomenuli, informácia obrazová i sémantická sa ukladá do pamäti.

Šošovka je v prednej strednej časti oka a vďaka jej pružnosti sa môže meniť **ohnisková vzdialenosť** jej zaostrenia. Rozdiel medzi očnou a sklenou šošovkou je v tom, že očná šošovka môže meniť svoju vonkajšiu krivosť, a tým sa mení ostrosť obrazu.



Obr. 1.3 Orientácia a veľkosť obrazu na sietnici

Vzdialenosť stredu šošovky od sietnice sa pohybuje približne od 14 mm do 17 mm. Obrázok 1.3 znázorňuje, ako oko sníma 15 m vysoký strom, vzdialený 100 m. Nech x je veľkosť obrazu na sietnici v millimetroch, potom veľkosť obrazu je:

$15/100 = x/17$, a preto $x = 2{,}55$ mm.

1.2 Základné pojmy

Opisne by sa dala počítačová grafika definovať takto. Na nakreslenie každého a akéhokoľvek obrázku na danej rastrovej obrazovke stačí systematicky rozsvecovať a zhasínať všetky obrazové body na obrazovke, kým konfigurácia obrazových bodov neznázorní požadovaný obrázok. Takto možno s určitou presnosťou získať ľubovoľný obrázok či text - vrátane štátnych tajomstiev, popisu lieku proti rakovine či tvárí všetkých ľudí, akí kedy žili na zemi. Pri niektorých z týchto obrázkov samozrejme nevieme rozhodnúť, či predstavujú práve to, čo požadujeme. Nevýhodou takéhoto postupu - systematického generovania všetkých obrazoviek - je, že čas získania požadovaného obrázku môže presiahnuť trvanie nášho života. Počítačová grafika je súhrn postupov, vrátane interaktívnych, ktoré skracujú čas získania požadovaného obrázku.

Nasleduje presná definícia Medzinárodnej organizácie pre normalizáciu (ISO): **Počítačová grafika sú metódy a techniky konštrukcie, manipulácie, ukladania a zobrazenia obrazov pomocou počítača.** (Obrazy generované počítačom môžu byť 2D alebo 3D.)

Interaktívna počítačová grafika je počítačová grafika, v ktorej užívateľ dynamicky riadi alebo mení obsah, formát, veľkosť alebo farby obrazu na obrazovke (*display surface*).

Interaktívna počítačová grafika kontrastuje s **pasívnou (zobrazovacou) grafikou** (*image synthesis*), ktorá z obrazových dát vytvára obraz, a kde užívateľ nemôže ani riadiť ani meniť prvky zobrazeného obrazu. Pojem grafika sa často používa užšie (na pasívnu grafiku) i širšie (interaktívnu počítačovú grafiku, zahŕňajúcu aj jej pasívnu časť).

Grafika sa zvykne podľa technológie vytvárania obrazu deliť na čiarovú resp. súradnicovú grafiku a rastrovú grafiku. Čiarová grafika sa niekedy nazýva vektorová. Do počítačovej grafiky sa neraz zahŕňa aj **spracovanie obrazu** (*image processing*).

Spracovanie obrazu je proces aplikovania akejkoľvek operácie na obraz alebo obrazové dáta pre daný účel. Napr. analýza scény, kompresia obrazu, konštrukcia 2D alebo 3D modelov objektov, ai.

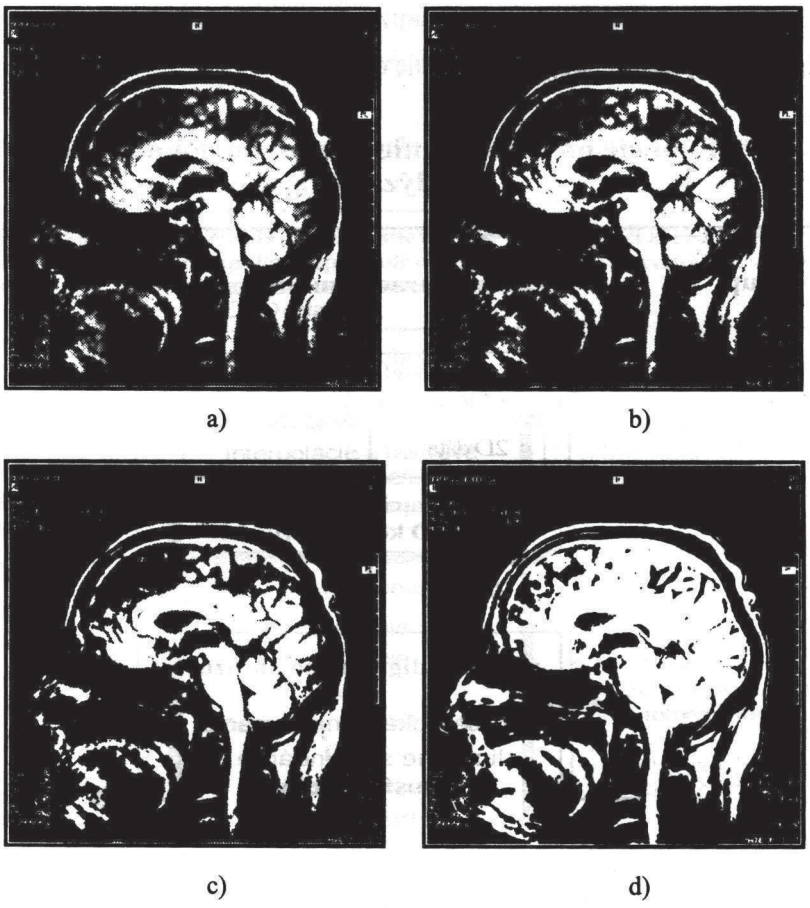
vstup je daný ako	**výstup je daný ako**		
	popis	**obrázok**	**zvuk**
popis	symbolická manipulácia	počítačová grafika	hlasový výstup
obrázok	rozpoznávanie obrazov	spracovanie obrazu	
zvuk	rozpoznávanie zvuku		spracovanie zvuku

Obr. 1.4 Oblasti spracovania informácie

Počítačovej grafike príbuzné úlohy rieši aj **rozpoznávanie obrazcov** (*pattern recognition*). Vzťahy týchto a ďalších oblastí spracovania informácie objasňuje obr. 1.4, [SKAL93]. Prázdne políčka označujú zatiaľ nepomenované oblasti.

Metódami počítačovej grafiky vytvorené grafické zobrazenie sa nazýva **obrázok** (*picture*). Definuje sa ako priestorovo štrukturovaná postupnosť grafických výstupných prvkov (napr. čiary, texty) určená na uloženie alebo zobrazenie. Niekedy budeme rozlišovať štrukturovaný obrázok od neštrukturovaného **obrazu** (*image*), pozostávajúceho iba z obrazových bodov. Pojem obraz používame aj pre **spojitý obraz,** kde ním rozumieme dvojrozmernú funkciu intenzity $f(x, y)$, kde x a y sú konečné priestorové súradnice bodu v rovine a funkčná hodnota vyjadruje **jas**. **Digitálny obraz** získame diskretizáciou priestorových i jasových hodnôt a môžeme si ho predstaviť ako maticu M, pre ktorú riadok i a stĺpec j určuje bod obrazu a hodnota prvku matice $M(i, j)$ jasovú uroveň. Bod digitálneho obrazu nazývame **pixel** (*picture element*).

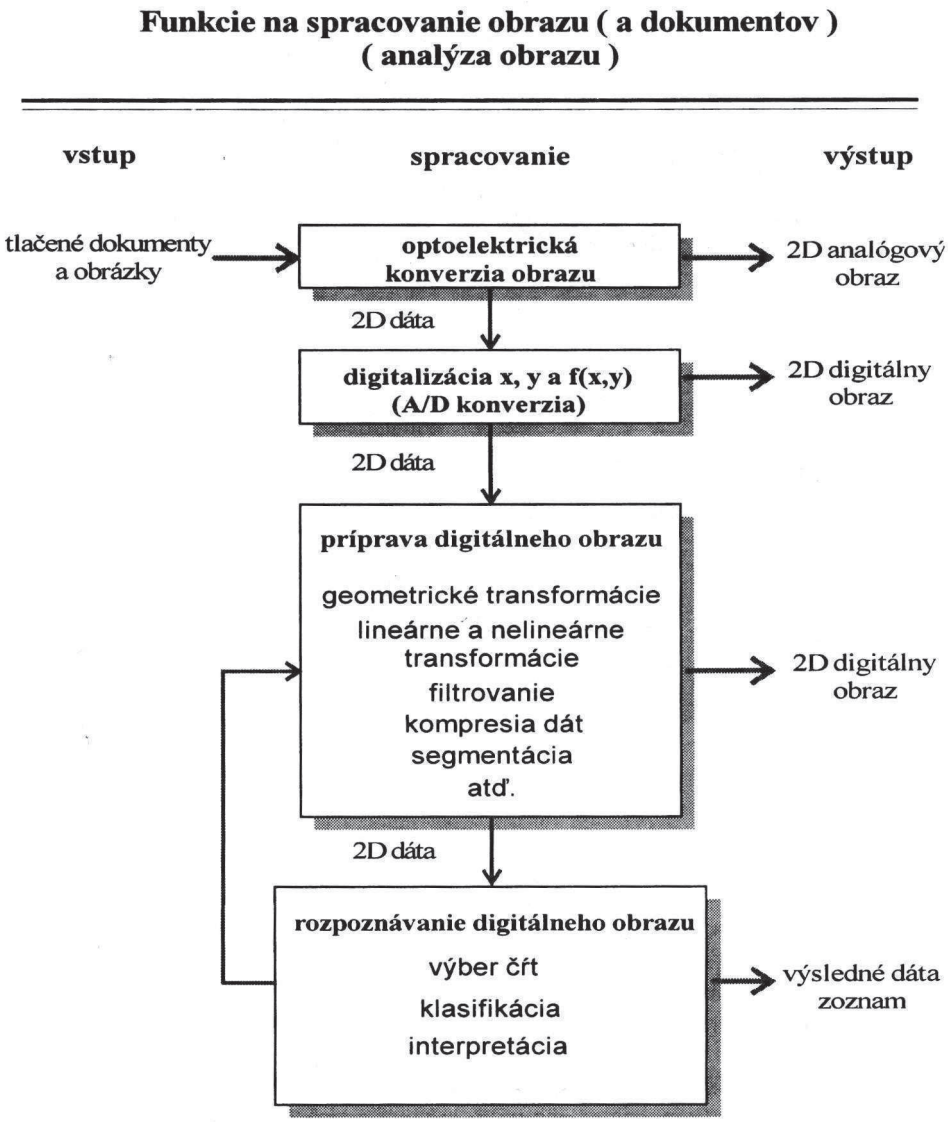
Ako vidno, spracovanie obrazu má na vstupe obraz a na výstupe tiež obraz (*image*). Vstupný obraz môže byť **prirodzený** (získaný kamerou, skenerom, apod.) alebo **syntetický** (získaný metódami počítačovej grafiky), kým výstupný obraz môže byť zobrazený na hociktorom výstupnom zariadení. Je mnoho typov aplikácií, ktoré používajú obraz (*image*). Obraz alebo digitálny obraz je dátová štruktúra obsahujúca obrazové body a ďalšie obrazové dáta, ktoré poskytujú kontext na interpretáciu obrazu. Na prácu s grafickými dátami sa identifikuje šesť hlavných tried funkcií - analýza obrazu (*image analysis*), interpretácia obrazu (*image interpretation*), prezentácia obrazu (*image presentation*), spracovanie obrazu (*image processing*), snímanie obrazu (*image sensing*) a syntéza obrazu (*image synthesis, computer graphics*).



Obr. 1.5 Obrazy s rôznymi úrovňami (64, 32, 4 a 2) šedej farby

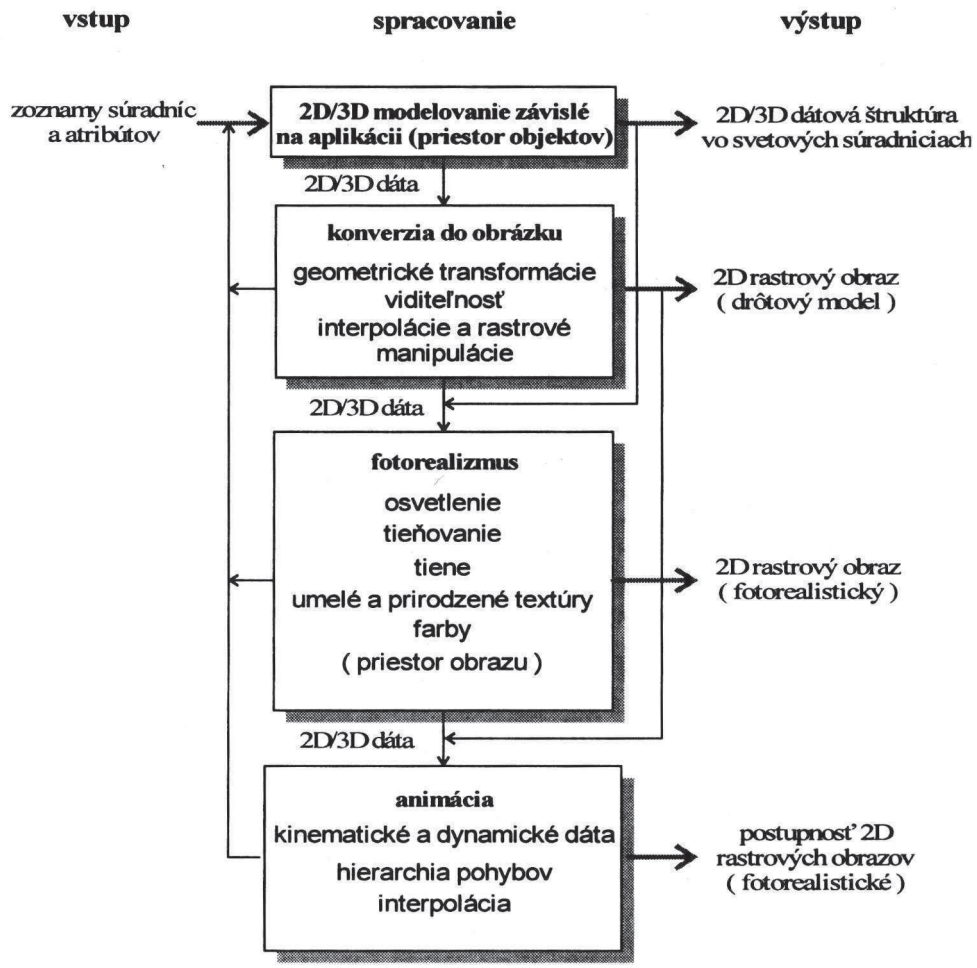
Historicky prvou aplikáciou spracovania obrazu bol v roku 1920 prvý **prenos obrazu** cez oceán pomocou podmorského telegrafného káblu medzi Londýnom a New Yorkom. Tento Bartlaneho systém prenášal najprv len 5 úrovní šedej farby, neskôr v roku 1929 ho rozšírili na 15 úrovní šedej. Obraz kódovali a zobrazovali špeciálnymi zariadeniami. Zásadné zlepšenie metód sa dosiahlo použitím počítačovej techniky. V roku 1964 výskumníci Jet Propulsion Laboratory vyvinuli metódu na **zlepšenie obrazu**, ktorý kozmická sonda Ranger 7 vysielala na Zem z Mesiaca. Neskôr zasa v projekte Mariner sa prijímali snímky z Marsu a prvé pristátie ľudí na Mesiaci sa prenášalo priamym prenosom záberov. Tieto postupy výrazne zlepšili techniky spracovania obrazu. Obrázok 1.5 ilustruje efekt znižovania počtu rezervovaných bitov pre jeden obrazový bod. Ak chceme dosiahnuť kvalitu čiernobieleho TV obrazu, musíme prenášať obraz s **rozlišovacou schopnosťou** 512*512 bodov a aspoň 128 úrovní jasu. Minimálne požiadavky na obraz, ktorý má byť spracovaný ľudským okom, sú 256*256 bodov a 64 úrovní jasu.

Funkčnosť spracovania obrazu popisuje obr. 1.6.



Obr. 1.6 Funkcie na spracovanie obrazu

Funkčnosť generovania obrázku resp. obrazu popisuje obr. 1.7.

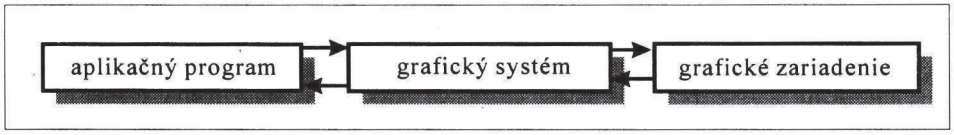


Obr. 1.7 Funkcie na vytváranie obrazu (pasívna počítačová grafika)

1.3 Štruktúra interaktívnej grafickej aplikácie

Slovom **systém** označujeme súbor prvkov a ich vzťahov, pričom relatívne stálejšie vzťahy nazývame štruktúrou systému. Subsystém je vymedziteľná časť systému. Systémy, ktoré budeme (nie vždy úplne) popisovať, patria medzi *cieľovo orientované systémy*. Ich cieľom je maximálne efektívne (rýchlo, úsporne, bezpečne, názorne a lacno) spracovať grafickú informáciu.

Základnú štruktúru aplikačného systému, využívajúceho (interaktívnu) počítačovú grafiku možno charakterizovať takto: **aplikačný program** (prípadne s aplikačnou databázou) spracúva model reálnych alebo abstraktných objektov a rozhoduje, čo zobraziť. Popíše teda pre **grafický systém** tvar a vzhľad objektu a grafický systém vykoná požadované zobrazenie na **zariadení**. Autor programu tak nemusí študovať detaily zariadení, aplikačný program je na zariadení nezávislý a prenositeľný na iné zariadenia.



Obr. 1.8 Základná štruktúra interaktívnej grafickej aplikácie

Analogicky, grafický systém odovzdáva aplikačnému programu vstupy od operátora, ktorý komunikuje s programom cez vstupné zariadenie. Jednou z hlavných myšlienok takéhoto konceptuálneho modelu je **oddelenie modelovania a zobrazovania**. Grafický systém sa stáva nezávislým od aplikácie, rastie jeho univerzálnosť a klesá zložitosť. O hranici medzi modelovaním a zobrazovaním, ktorá vždy vedie "na území" aplikačného programu, sa vedú spory, napr. kam patrí hierarchia skladania zložitejších objektov z jednoduchších. Grafický systém GKS modelovanie nemá, kým systém PHIGS obsahuje aj štruktúry a ďalšie nástroje na hierarchické modelovanie. Existuje aj názor, že kým sa neoddelí modelovanie od zobrazovania, ťažko hovoriť o grafickom systéme ako o univerzálnom nástroji na vývoj grafických aplikácií.

1.4 Vývoj počítačovej grafiky a spracovania obrazu

Vývoj počítačovej grafiky a spracovania obrazu bol podstatne ovplyvnený rozvojom hardveru a softveru. Prvé **obrazovky** vznikli v roku 1950, **svetelné pero** na určovanie grafických objektov sa objavilo okolo roku 1955. V polovici 60-tych rokov prišli **vektorové obrazovky** (atómom obrázku bol vektor, tj. vykreslená úsečka) a neskôr **pamäťové obrazovky**. Ďalším pokrokom bolo oddelenie zobrazovacej jednotky od základného počítača (inteligentný satelit). V polovici 70-tych rokov vynašli **rastrovú grafiku**, založenú na televíznej technológii. (Atómom obrázku je obrazový bod, pixel.) Dnes je k dispozícii veľmi dokonalý **2D vstup i výstup** a dômyselné zariadenia na 3D vstup (3D myš, ai.) i výstup (stereolitografia). Skúma sa zvuková komunikácia. Spracovanie jednotlivých médií (text, obraz, zvuk, video...) integruje **multimediálna technológia**.

Rozvoj softveru možno periodizovať v týchto etapách: **1. balíky podprogramov** pre konkrétne zariadenia, **2.** balíky vyššej úrovne, **nezávislé od zariadenia**. V polovici 70. rokov bola zrejmá potreba normalizovať grafické systémy a od roku 1977 prevzala koordináciu týchto snáh medzinárodná organizácia pre štandardizáciu - ISO (International Standardization Organization). **3.** etapou v rozvoji grafického softveru teda sú a budú medzinárodne prenositeľné **grafické systémy**; impulzom na túto stránku

vývoja grafického softveru je **prenositeľnosť** (*portability*). V praxi sa často používajú aj "firemné normy", napr. formáty kódovania dát **PCX** či **GIF**, jazyk **PostScript** firmy Adobe či grafický systém **OpenGL** firmy Silicon Graphics.

Často sa vývoj počítačovej grafiky a spracovania obrazu charakterizuje prehľadom úspešných aplikácií, napr. v leteckom a automobilovom priemysle, domácej zábave, animácii a filmových efektoch z Hviezdnych vojen či Terminátorov. Z perspektívnych oblastí sa zvykne uvádzať virtuálna realita, vizualizácia, sieťová komunikácia, fraktálne modelovanie, aplikácie v medicíne a geografických systémoch. Podstatou všetkých úspešných aplikácií bolo rozumné zvládnutie potrebného objemu dát, dôkladné štúdium problému a jeho kvalifikované algoritmické riešenie na adekvátnej hardverovej a softverovej platforme.

Vývoj počítačovej grafiky a spracovania obrazu súvisí s kvalitným vzdelaním. Diskutuje sa dokonca o tom, že počítačová grafika má patriť k základom úvodného informatického vzdelania. Ako sme už uviedli, nerobíme si nárok na úplnosť, iba na sprístupnenie základov prepotrebnej oblasti v slovenčine. Svetovú normu predstavuje učebnica, ktorej sa vo výklade budeme často pridržať - **Computer Graphics Principles and Practice**, ktorú napísali James D. Foley, Andries van Dam, Steven K. Feiner a John F. Hughes, [FOLE90]. Táto 1174-stranová kniha sa považuje za štandardnú učebnicu ("bibliu pre grafikov"), ktorá nadväzuje na prvé vydanie prvých dvoch autorov Fundamentals of Interactive Computer Graphics. Obe vydania vyšli v Addison Wesley v rokoch 1982 a 1990. Štruktúrou počítačovej grafiky ako vysokoškolského učiva podľa tejto knihy sa inšpiruje mnoho učebníc a univerzít. V niektorých smeroch však treba upozorniť na špecializovanejšie knihy, ktoré pokrývajú ďalšie dôležité oblasti: **Advanced Animation and Rendering Techniques** [WATT92], **Image Processing** [GONZ87], **Computational Geometry - An Introduction** [PREP85]. V zozname literatúry ďalšie uvedené učebnice a skriptá označujeme značkami U a S.

V tejto knihe nemožno pokryť počítačovú grafiku a spracovanie obrazu vyčerpávajúco. Už sme uviedli, že výklad možno tematicky rozčleniť do 4 celkov, ktoré však explicitne neuvádzame v obsahu.

Základy prinášajú základné pojmy, súvisiace s videním a spracovaním obrazovej informácie, stručný historický prehľad, geometrické transformácie v 2D a 3D, orezávanie a prieniky, reprezentáciu kriviek a plôch parametrickými funkciami (menovite kubické splajny). Podrobnejšie uvádzame problematiku rasterizácie čiarových a plošných grafických výstupných prvkov.

Spracovanie obrazu zavádza matematické metódy spracovania obrazu v spojitej rovine a prevod z Euklidovskej roviny do rastrovej, pokrýva klasické metódy Fourierovej transformácie, filtrovania, jasovej korekcie, segmentácie a obrysu, morfologických transformácií a skeletovania, identifikácie hrán.

Pre **vytváranie obrazu** (*image synthesis*) sa najprv analyzujú požiadavky a postupy geometrického modelovania, reprezentácie telies a ich hraníc metódou CSG (constructive solid geometry), potreby realistického zobrazovania, potom stratégie určovania viditeľných povrchov, osvetľovanie a tieňovanie, textúry a fraktály, tiene, priehľadnosť, odraz, fyzikálne osvetľovacie modely a metódy sledovania lúča (*ray tracing*) i radiačná metóda (*radiosity*). V tejto časti zavádzame hľadisko pozorovateľa, farbu a rôzne systémy popisu farieb.

V záverečnej časti **Užívateľské rozhranie a grafické systémy** rozoberáme technológiu interakcie, niektoré zaužívané interaktívne postupy, oknové systémy a ich nadstavby. Ďalej sa podrobne rozoberajú medzinárodné grafické normy a systémy, najmä norma PHIGS, kódovanie grafickej informácie a základy multimediálnej technológie.

Rámec tejto knihy neposkytuje priestor na niektoré najnovšie trendy. Pravdepodobne záverečným krokom k dokonalému realizmu nášho vnímania je integrácia fotorealistických obrázkov s informáciou pre ďalšie zmysly. Dokonalý vizuálny efekt možno dosiahnuť pomocou **na hlave upevnených displejov** (*head mounted display, HMD*), zariadenia na počúvanie zvuku sú bežné a informáciu pre hmat poskytuje **dátová rukavica** (*data glove*) alebo **dátový oblek**. S takýmito technickými prostriedkami a príslušným softverom už možno vytvoriť dokonalú ilúziu pohybu **vo virtuálnom svete** (*virtual world, virtual reality*), napr. medzi molekulami či galaxiami alebo v počítačovo rekonštruovaných už neexistujúcich budovách.

Jednou z najdôležitejších aplikačných oblastí počítačovej grafiky sú **návrhové a výrobné systémy CAD/CAM** (*Computer Aided Design, Computer Aided Manufacture*), o ktorých by ale musela byť samostatná kniha. Tejto veľkej témy sa dotkneme len okrajovo a nesystematicky tam, kde to bude potrebné kvôli kontextu nášho výkladu. Takisto vynechávame aj rozsiahlu problematiku vizualizácie (*scientific visualization*).

Najvhodnejšími zdrojmi na sledovanie perspektívnych oblastí výskumu sú časopisy **Computer Graphics, Computer Graphics Forum, ai.** a zborníky i tutoriály z dvoch vedúcich konferencií - SIGGRAPH a EUROGRAPHICS.

2

Transformácie v rovine a priestore

2.1 Úvod

V praktických úlohách často vytvárame alebo používame obraz určitého objektu, ktorý treba **vhodne transformovať**: zväčša vybrať určitú časť a prepočítať súradnice na dané výstupné zariadenie. V tejto kapitole ukážeme veľmi stručne vzájomnú súvislosť medzi maticami a lineárnymi transformáciami v rovine a odvodíme matice špeciálnych prípadov najčastejšie používaných rovinných a priestorových transformácií: posunutia, otočenia a škálovania. Ak bude treba, budeme vyjadrovať body a vektory v homogénnych súradniciach.

2.2 Matice

Medzi transformáciami a maticami je jednoznačný vzťah: každej lineárnej transformácii vieme priradiť určitý typ matice. Pri lineárnych transformáciách dvojrozmerných vektorových priestorov vystačíme s maticami typu 2×2. Pre afinné transformácie v rovine potrebujeme matice typu 3×3. Každý bod roviny budeme stotožňovať s vektorom typu 1×2 alebo typu 1×3. Pri vyjadrovaní bodov pomocou matíc 1×3 hovoríme o **homogénnych súradniciach** a príslušné priradenie je takéto: Bodu o súradniciach (x, y) priradíme $(x, y, 1)$. Zmysel zavedenia homogénnych súradníc si môžeme ukázať na jednoduchom príklade.

Príklad 1. Nájdime maticu pre posunutie v rovine zadané vektorom (t_x, t_y):

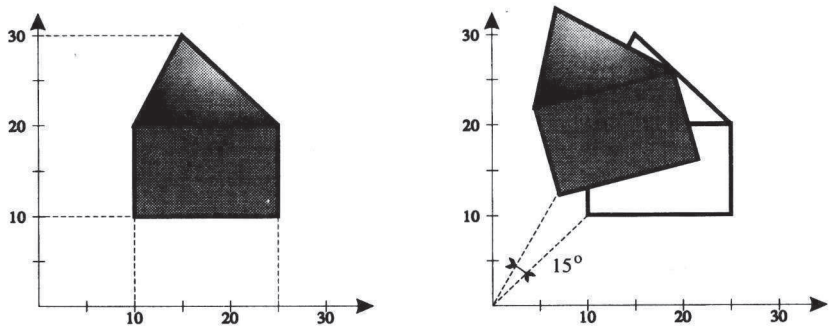
$$x' = x + t_x,\ y' = y + t_y.$$

Posunutie sa nedá vyjadriť maticou typu 2×2, tak aby sa transformácia reprezentovala maticovým násobením. V počítačovej grafike je zavedenie homogénnych súradníc veľmi častým spôsobom vyjadrovania. Formálny rozdiel býva iba v riadkovom resp. stĺpcovom zápise. V homogénnych súradniciach už môžeme rovnice posunutia prepísať do maticového riadkového zápisu

$$X' = (x', y', 1) = (x, y, 1).\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix} = X.A\,.$$

2.3 Otočenie a zmena mierky

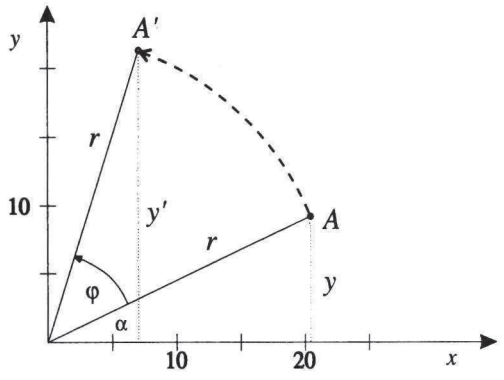
Transformácia objektu po kruhovej dráhe sa nazýva **otočenie**. Je určené uhlom a stredom otočenia (pevným bodom). Nasledujúci príklad na obr. 2.1 ukazuje, ako pôsobí otočenie so stredom v začiatku (sústavy súradníc) na objekt pri otočení o 15 stupňov.



Obr. 2.1 Ukážka otočenia pre daný objekt

Odvodíme vzťah pre vyjadrenie tohto zobrazenia. Objekt môžeme otočiť o ľubovoľný uhol φ vzhľadom k začiatku sústavy súradníc. Kedže sa zhodne otočí každý bod, stačí sledovať, čo sa deje s ľubovoľne vybratým bodom. Pri otočení o uhol φ nech bod $A(x, y)$ sa zobrazí do bodu $A'(x', y')$, pozri obr. 2.2. Najprirodzenejšie to vyjadríme použitím polárnych súradníc:

$$x = r.\cos\alpha\ , \qquad y = r.\sin\alpha, \qquad x' = r.\cos(\alpha+\varphi), \qquad y' = r.\sin(\alpha+\varphi)$$



Obr. 2.2 Otočenie bodu A do bodu A' o uhol φ

Úpravou posledných dvoch rovností (pomocou súčtu uhlov pre goniometrické funkcie sin(α+φ) a cos(α+φ)) môžeme vyjadriť x' a y':

$$x' = r.\cos(\alpha).\cos(\varphi) - r.\sin(\alpha).\sin(\varphi) = x.\cos(\varphi) - y.\sin(\varphi)$$

$$y' = r.\cos(\alpha).\sin(\varphi) + r.\sin(\alpha).\cos(\varphi) = x.\sin(\varphi) + y.\cos(\varphi)$$

Tým sme dostali transformačnú maticu otočenia o uhol φ v homogénnych súradniciach:

$$\begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Transformácia, pomocou ktorej môžeme zmeniť veľkosť objektu, sa nazýva **zmena mierky** alebo **škálovanie**. Tento typ zobrazenia všeobecne môžeme zapísať nasledovne:

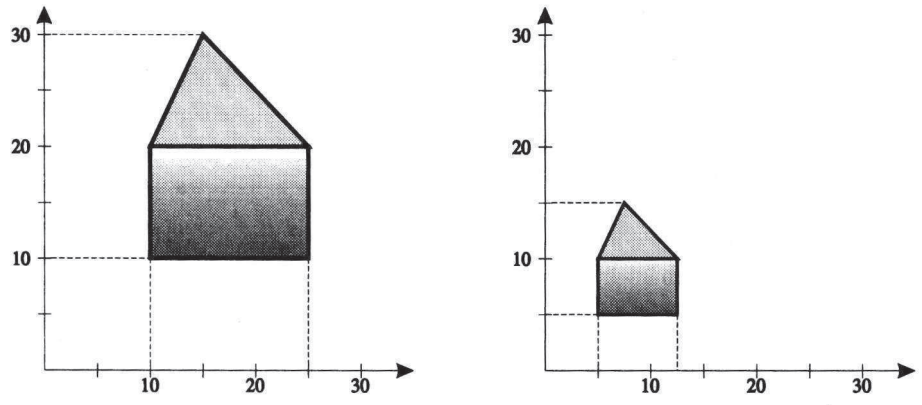
$$x' = s_x.x,$$

$$y' = s_y.y,$$

kde s_x, s_y sa nazývajú škálovacie faktory a zodpovedajúca transformačná matica má tvar:

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Poznamenajme, že aj pre regulárne škálovanie vždy existuje pevný bod, v našom prípade je to zatiaľ len začiatok súradníc. Neskôr ukážeme ako vyjadriť škálovanie pre ľubovoľný pevný bod. Na obrázku 2.3 vidíme ako zmena mierky pre hodnoty $s_x = 0.5$ a $s_y = 0.5$ pôsobí na zadaný objekt v tvare domčeka.



Obr. 2.3 Ukážka zmeny mierky pre daný objekt

2.4 Posunutie objektu a sústavy súradníc

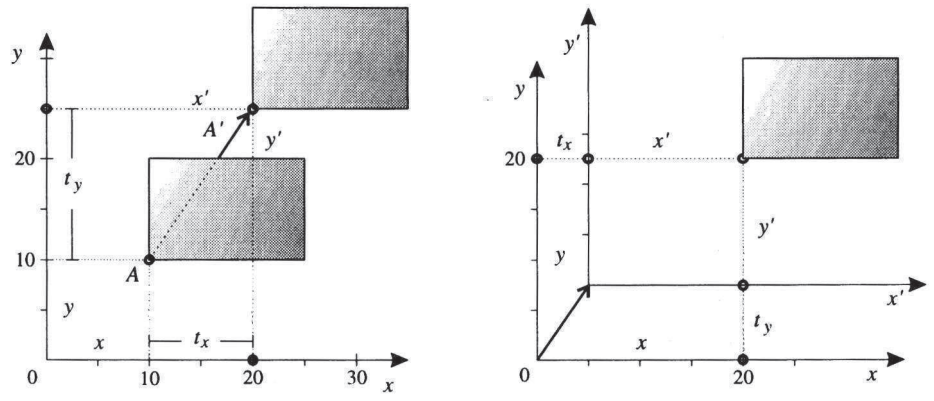
Posunutie je premiestenie objektu z jednej pozície do druhej. Poznamenajme, že na rozdiel od rotácie a škálovania nemá pevný bod. Posunutie v rovine je definované pomocou vektora (t_x, t_y). Zodpovedajúcu transformačnú maticu pre posunutie sme už uviedli v príklade 1.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

Môžeme si položiť otázku: Ako sa zmenia súradnice bodov pri posunutí sústavy súradníc? Na nasledujúcom obr. 2.4 vidíme, že ten istý bod má vzhľadom k posunutej sústave súradníc súradnice posunuté o opačný vektor $(-t_x, -t_y)$. Zodpovedajúca matica má potom tvar:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{pmatrix}$$

Táto matica $\boldsymbol{T'}$ je inverzná k matici posunutia $\boldsymbol{T}$. Obdobne by sme mohli odvodiť vzťahy pre otočenie a škálovanie sústavy súradníc. Všeobecne platí, že pre transformáciu objektov a transformáciu sústavy súradníc platí vzťah navzájom inverzného zobrazenia, pokiaľ sú tieto zobrazenia jednoznačné.



Obr. 2.4 Posunutie bodu a sústavy súradníc

2.5 Kompozícia 2-rozmerných transformácií

Ukážeme si, ako môžeme využiť násobenie matíc pri skladaní zobrazení. Napríklad, chceme vyjadriť zmenu mierky so stredom v ľubovoľnom pevnom bode $A(x, y)$. Budeme to riešiť tak, že uskutočníme za sebou tri zobrazenia:

1. Posunieme sústavu súradníc do bodu A.
2. Uskutočníme zmenu mierky v začiatku sústavy súradníc.
3. Posunieme späť bod A do pôvodného začiatku.

Každej tejto transformácii zodpovedá jedna transformačná matica. Výslednej zloženej transformácii zodpovedá nasledujúca matica:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{pmatrix} * \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ x(1-s_x) & y(1-s_y) & 1 \end{pmatrix}$$

Podobne otočenie so stredom v ľubovoľnom bode $A(x, y)$ vykonáme pomocou týchto troch transformácií:

1. Posunieme sústavu súradníc do bodu A.
2. Uskutočníme otočenie okolo začiatku sústavy súradníc o uhol φ.
3. Posunieme späť bod A do pôvodného začiatku.

Obdobne výslednú zloženú transformáciu vyjadríme násobením matíc:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{pmatrix} * \begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{pmatrix} =$$

$$\begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ x(1-\cos\varphi)+y\sin\varphi & y(1-\cos\varphi)-x\sin\varphi & 1 \end{pmatrix}$$

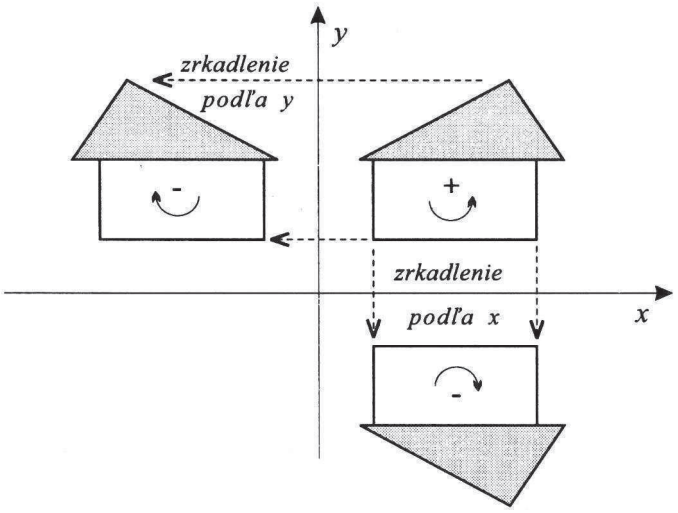
2.6 Iné transformácie

2.6.1 Súmernosť podľa priamky

Niektoré aplikácie počítačovej grafiky vyžadujú aj iné transformácie v rovine. V prípade, že uvažujeme len transformácie zachovávajúce začiatok sústavy súradníc, potom môžeme napísať matice 2×2 pre súmernosť (zrkadlenie) podľa osi x a y:

$$Z_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad Z_y = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

Táto podmatica sa nachádza vždy v ľavej hornej časti všeobecnej matice typu 3×3, vyjadrujúcej afinné transformácie roviny. Ide o špeciálny prípad škálovania, kde $s_x = -1$ alebo $s_y = -1$.



Obr. 2.5 Súmernosť podľa osi x a y

V obidvoch prípadoch je determinant matice záporný. To znamená, že transformácie **menia orientáciu** (vidno to na obrázku 2.5). Na túto skutočnosť treba dávať pozor. Ak totiž máme mnohouholník orientovaný proti smeru chodu hodinových ručičiek, potom takáto transformácia zmení orientáciu mnohouholníka na opačnú. Maticu súmernosti podľa ľubovoľnej priamky získame skladaním s príslušnými otočeniami a základnými súmernosťami podľa osí.

2.6.2 Skosenie

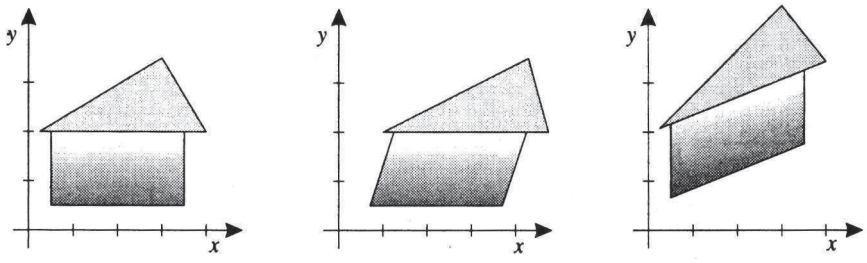
Tento typ transformácie vyvoláva dojem, akoby sa objekty po vrstvách posúvali. Rozoznávame dva základné typy:

- skosenie v smere x (mení sa súradnica x, súradnica y zostáva nezmenená)
- skosenie v smere y (mení sa súradnica y, súradnica x zostáva nezmenená)

Obidva typy majú jednoduchú transformačnú maticu:

$$S_x = \begin{pmatrix} 1 & 0 \\ s_x & 1 \end{pmatrix}, \qquad S_y = \begin{pmatrix} 1 & s_y \\ 0 & 1 \end{pmatrix}$$

Na nasledujúcom obrázku obr. 2.6 vidíme ako pôsobí skosenie na jednoduchý objekt



Obr. 2.6 Skosenie podľa x (s_x=0.4) a skosenie podľa y (s_y=0.4)

2.7 Zobrazenie okna na zobrazovacie pole

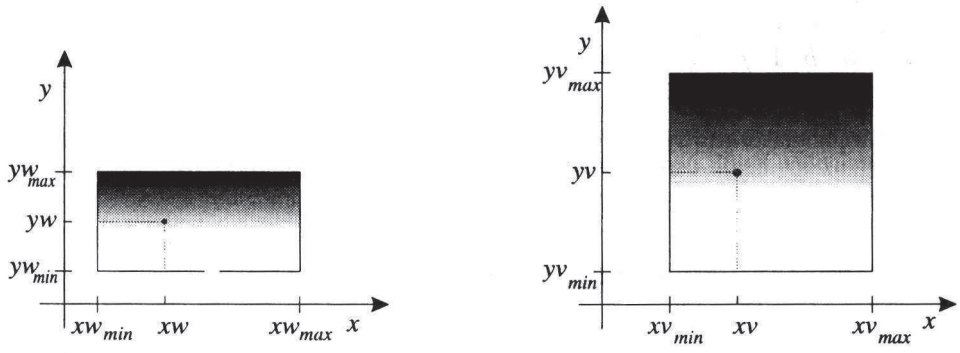
Užívateľ obvykle zadáva objekty vo svojich užívateľských súradniciach. Na zobrazenie **oblasti záujmu** si zvolí minimálne a maximálne súradnice v obidvoch smeroch, tzv. **okno**. Funkciu na definovanie okna viditeľnosti budeme označovať

$$window\ (xw_{min}, xw_{max}, yw_{min}, yw_{max}) .$$

Avšak toto okno užívateľ nemusí chcieť zobraziť na celú obrazovku (zobrazovaciu časť príslušného výstupného zariadenia). Preto je prirodzené zaviesť pojem **zobrazovacie pole** resp. **záber**. Je to tá časť, na ktorú sa bude transformovať okno. Funkciu pre definovanie zobrazovacieho poľa (záberu) označíme

$$viewport\ (xv_{min}, xv_{max}, yv_{min}, yv_{max}) .$$

Ako vyzerá transformačná matica, ktorá bude realizovať príslušné zobrazenie okna na zobrazovacie pole?



Obr. 2.7 Transformácia okna na zobrazovacie pole (záber)

Predpokladáme, že bod (xw, yw) sa zobrazí do bodu (xv, yv), pozri obrázok 2.7. Prirodzená požiadavka je, aby sa pri transformácii **zachovali pomery strán**. Preto podľa označenia z obrázku 2.7 požadujeme, aby platili tieto rovnosti

$$\frac{xw-xw_{min}}{xw_{max}-xw_{min}} = \frac{xv-xv_{min}}{xv_{max}-xv_{min}},$$

$$\frac{yw-yw_{min}}{yw_{max}-yw_{min}} = \frac{yv-yv_{min}}{yv_{max}-yv_{min}}.$$

Odtiaľ môžeme vyjadriť hodnoty xv a yv

$$xv = s_x \cdot (xw - xw_{min}) + xv_{min},$$

$$yv = s_y \cdot (yw - yw_{min}) + yv_{min},$$

kde

$$s_x = \frac{xv_{max}-xv_{min}}{xw_{max}-xw_{min}}, \qquad s_y = \frac{yv_{max}-yv_{min}}{yw_{max}-yw_{min}}.$$

V tomto poslednom vyjadrení s_x a s_y sú koeficienty pre zmenu mierky z okna na záber a xv_{min}, yv_{min} sú relatívne hodnoty posunu. Nakoniec rovnosti môžeme upraviť na tvar, kde pre transformáciu jednej súradnice máme len jednu operáciu súčtu a násobenia:

$$xv = s_x . xw + a,$$

$$yv = s_y . yw + b,$$

kde

$$a = -s_x.xw_{min} + xv_{min}, \quad b = -s_y.yw_{min} + yv_{min}.$$

Hľadaná matica je teda

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ a & b & 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ -s_x.xw_{min} + xv_{min} & -s_y.yw_{min} + yv_{min} & 1 \end{pmatrix}.$$

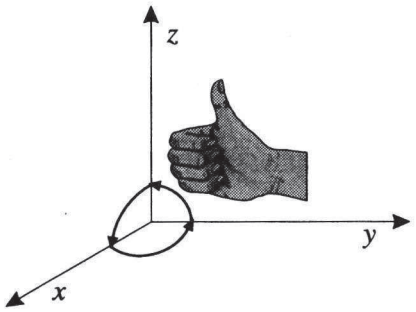
V grafických systémoch sa táto matica zostrojí automaticky po zadaní funkcií *window* a *viewport*. Kvôli úspore pamäti sa zvyčajne ukladajú len prvé dva stĺpce, pretože pre afinné transformácie je tretí stĺpec rovnaký.

2.8 Pravotočivá sústava súradníc v priestore

Afinné transformácie v rovine sme reprezentovali homogénnymi maticami typu 3×3. Podobne aj afinné transformácie v priestore budeme reprezentovať maticami typu 4×4. Každý bod v priestore so súradnicami (x, y, z) budeme reprezentovať v homogénnych súradniciach ako štvoricu $(x, y, z, 1)$.

Sústavu súradníc xyz v trojrozmernom priestore nazveme **pravotočivou**, ak pri pohľade z kladnej poloosi x, y, z pri otáčaní o kladne orientovaný uhol t.j. proti smeru chodu hodinových ručičiek o 90 stupňov nám bude prechádzať vždy jedna os do druhej podľa nasledujúcej tabuľky a obrázku 2.8:

Os otáčania	transformácia osi
x	y do z
y	z do x
z	x do y



Obr. 2.8 Prechod osí pre pravotočivú sústavu súradníc

V pravotočivej sústave súradníc lineárne nezávislé vektory $\boldsymbol{u}$, $\boldsymbol{v}$ a ich vektorový súčin $(\boldsymbol{u} \times \boldsymbol{v})$ vytvárajú spolu pravotočivú sústavu súradníc. Trojrozmernú afinnú transformáciu vyjadríme v homogénnych súradniciach ako maticu typu 4×4 vo všeobecnom zápise:

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ t_1 & t_2 & t_3 & 1 \end{pmatrix}.$$

Tento všeobecný zápis konkretizujeme pre špeciálne prípady 3 známych základných transformácií - zmenu mierky, posunutie a otočenie.

2.9 Zmena mierky a posunutie

Zmena mierky (*škálovanie*) v trojrozmernom priestore je prirodzeným zovšeobecnením dvojrozmerného škálovania. Matica škálovania s pevným bodom v začiatku sústavy súradníc a škálovacími faktormi pre jednotlivé osi s_x, s_y, s_z má nasledujúci zápis:

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Matica posunutia o vektor (t_x, t_y, t_z) má tvar:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{pmatrix}.$$

Podobne môžeme uvažovať o **zmene sústavy súradníc**, ako sme to urobili pre dvojrozmerné transformácie. Napríklad po zmene mierky súradnicových osí o s_x, s_y, s_z sa zmenia súradnice bodu v novej sústave súradníc a tomu zodpovedá inverzná matica škálovania:

$$x' = 1/s_x \,.\, x,$$
$$y' = 1/s_y \,.\, y,$$
$$z' = 1/s_z \,.\, z\,.$$

$$\begin{pmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

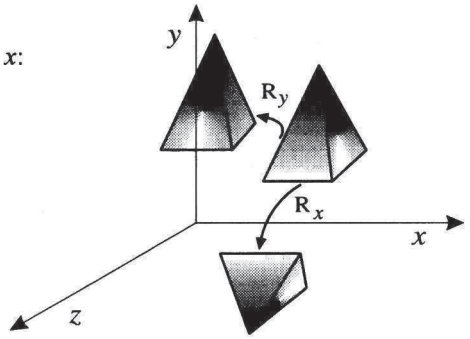
2.10 Otáčanie okolo súradnicových osí

Otáčanie v priestore je definované pomocou **orientovanej priamky** a **zadaného uhla**. Pre otáčanie v rovine xy si môžeme predstaviť, že uskutočňujeme otáčanie v priestore okolo kladne orientovanej osi z. Pri priestorovom otáčaní súradnica z ostáva nezmenená, a preto otáčanie v priestore môžeme vyjadriť podobne, ako sme vyjadrili otáčanie v rovine:

$$\mathbf{R}_z = \begin{pmatrix} \cos\varphi & \sin\varphi & 0 & 0 \\ -\sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Podobne vyjadríme otočenie okolo kladnej osi x:

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$



resp. otočenie okolo kladnej osi y:

$$\mathbf{R}_y = \begin{pmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Určenie znamienka pri funkcii $\sin\varphi$ vyplýva z tabuľky, ktorú sme určili pre pravotočivú sústavu súradníc. Pre ľavotočivú sústavu súradníc sa príslušné transformačné matice $\mathbf{R}_x$, $\mathbf{R}_y$, $\mathbf{R}_z$ zmenia len znamienkom pri funkcii $\sin\varphi$. Stĺpce aj riadky ľavej hornej podmatice 3×3 pre $\mathbf{R}_x$, $\mathbf{R}_y$, $\mathbf{R}_z$ tvoria v priestore ortogonálnu bázu vektorov.

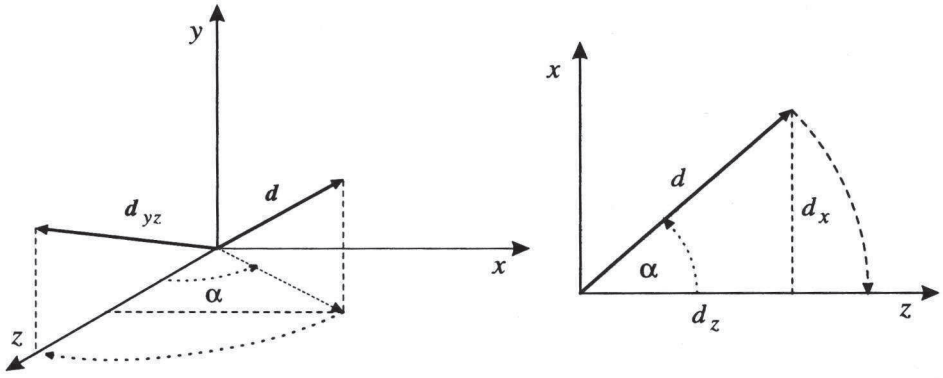
2.11 Otočenie okolo ľubovoľnej osi

Ak máme zadanú os otáčania pomocou orientovanej priamky p, prechádzajúcej cez začiatok sústavy súradníc t.j. smerovym vektorom $\boldsymbol{d} = (dx, dy, dz)$, potom otočenie okolo tejto osi o uhol φ uskutočníme zložením nasledujúcich transformácií :

1. otočenie okolo osi *y* tak, aby obraz vektora ***d*** ležal v rovine *yz*;
2. otočenie okolo osi *x* tak, aby výsledný obraz vektora ***d*** ležal na kladnej osi *z*;
3. otočenie okolo osi *z* o uhol φ;
4. otočenie okolo osi *x* o opačný uhol ako v bode 2;
5. otočenie okolo osi *y* o opačný uhol ako v bode 1.

Podrobnejšie rozpíšeme jednotlivé transformácie.

1. Na obrázku 2.9 vidíme také otočenie okolo osi *y*, ktoré zobrazuje vektor ***d*** do vektora $\boldsymbol{d}_{yz}$ v rovine *yz*.



*Obr. 2.9 Otočenie vektora **d** okolo osi y do roviny yz*

Otočenie okolo osi *y* o uhol -α zapíšeme maticou R_y. Funkcie sin a cos pre tento uhol otočenia upravíme nasledujúcimi rovnosťami (pozri obr. 2.9.):

$$\cos\alpha = dz/d \quad \text{a} \quad \cos(-\alpha) = dz/d\,,$$

$$\sin\alpha = dx/d \quad \text{a} \quad \sin(-\alpha) = -dx/d\,,$$

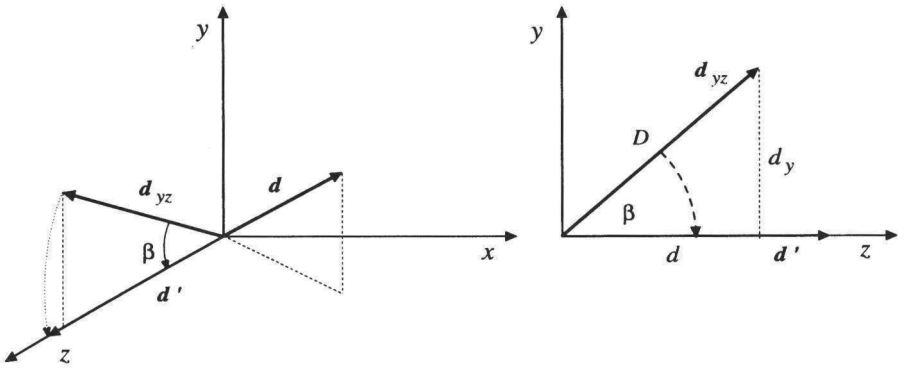
pričom $d = \sqrt{dx^2 + dz^2}$ je dĺžka priemetu vektora ***d*** v rovine *xz*. Otočenie okolo osi *y* môžeme zapísať maticou (porovnaj s predchádzajúcu maticou R_y)

$$R_y = \begin{pmatrix} dz/d & 0 & dx/d & 0 \\ 0 & 1 & 0 & 0 \\ -dx/d & 0 & dz/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

2. Otočenie okolo osi x o uhol β zapíšeme maticou $\mathbf{R}_x$. Pre uhol otočenia β platia nasledujúce rovnosti (pozri obr. 2.10, obraz vektora $\boldsymbol{d}_{yz}$ je vektor $\boldsymbol{d}'$, d je dĺžka priemetu vektora $\boldsymbol{d}$ v rovine xz):

$$\cos(\beta) = d/D \quad \text{a} \quad \sin(\beta) = -dy/D,$$

kde $D = \sqrt{dx^2 + dy^2 + dz^2}$ je dĺžka vektora $\boldsymbol{d}$.



Obr. 2.10 Otočenie okolo osi x

Otočenie okolo osi x môžeme zapísať maticou $\mathbf{R}_x$:

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & d/D & -dy/D & 0 \\ 0 & dy/D & d/D & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

3. Teraz otočíme okolo osi z o uhol φ ako sme požadovali na začiatku úlohy pre ľubovoľne zadanú os. Použijeme už predtým uvedenú transformačnú maticu $\mathbf{R}_z$:

$$\mathbf{R}_z = \begin{pmatrix} \cos\varphi & \sin\varphi & 0 & 0 \\ -\sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

4. Otočenie okolo osi x o opačný uhol $-\beta$ vyjadríme maticou inverznou k $\mathbf{R}_x$:

$$(\mathbf{R}_x)^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & d/D & dy/D & 0 \\ 0 & -dy/D & d/D & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

5. Otočenie okolo osi *y* vyjadríme maticou inverznou k $\mathbf{R}_y$:

$$(\mathbf{R}_y)^{-1} = \begin{pmatrix} dz/d & 0 & -dx/d & 0 \\ 0 & 1 & 0 & 0 \\ dx/d & 0 & dz/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Výsledná matica otočenia okolo osi je súčinom transformačných matíc:

$$\mathbf{R}_d = \mathbf{R}_y * \mathbf{R}_x * \mathbf{R}_z * (\mathbf{R}_x)^{-1} * (\mathbf{R}_y)^{-1}.$$

Podobne môžeme postupovať otočením vektora ***d*** najprv do roviny *xy* a potom okolo osi *z* do osi *x* a tak ďalej.

Otočenie okolo ľubovoľnej osi môžeme tiež vyjadriť pomocou vektorového súčinu. Transformačná matica otočenia okolo ľubovoľnej osi prechádzajúcej cez začiatok sústavy súradníc daná vektorom ***d*** a uhlom otočenia ϕ sa vyjadrí ako súčet matíc:

$$\mathbf{R}_d = \boldsymbol{d}'.\boldsymbol{d} + \cos\phi\,.\,(\mathbf{I} - (\boldsymbol{d}'.\boldsymbol{d})) + \sin\phi\,.(\mathbf{I}\times\boldsymbol{d}),$$

kde ***d'*** je transponovaný stĺpcový vektor a

$$\mathbf{I}\times\boldsymbol{d} = \begin{pmatrix} 0 & dz & -dy \\ -dz & 0 & dx \\ dy & -dx & 0 \end{pmatrix}.$$

2.12 Iné transformácie v priestore

Podobne ako sme uviedli pre transformácie v rovine, existujú transformácie v priestore, ktoré menia orientáciu t.j. menia pravotočivú sústavu súradníc na ľavotočivú. Uvažujme len tie transformácie, ktoré zachovávaju začiatok sústavy súradníc. Pre tieto nám stačí popísať ľavú hornú podmaticu 3×3. Pre nás budú zvlášť významné súmernosti podľa roviny.

1. Súmernosť podľa roviny *yz* sa vyjadrí nasledujúcou maticou:

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Podobne sa vyjadria aj súmernosti podľa roviny *xy* a *xz*.

2. Skosenie v smere roviny *xy* definujú koeficienty skosenia s_x a s_y a matica skosenia:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ s_x & s_y & 1 \end{pmatrix}.$$

3

Orezávanie a prienik

3.1 Úvod

Prirodzene, že ak máme definované okno a záber (zobrazovacie pole), potom máme definovanú transformáciu. Operátor môže požadovať od aplikačného programu, aby sa zobrazovali v okne určité detaily objektov, a tým sa niektoré časti obrázku môžu dostať mimo okna. Pri zobrazovaní okna alebo záberu musíme zrušiť príslušné časti obrázka, ktoré sú mimo okna alebo záberu. Túto operáciu nazývame **orezávanie** (*clipping*). Bude nás zaujímať postup orezávania výstupných grafických prvkov, ako sú bod, úsečka, mnohouholník a text. Je to špeciálny prípad všeobecného **prieniku** dvoch objektov, z ktorého jeden objekt je okno a druhý je grafický prvok.

Pri zobrazovaní bodov do okna je orezávanie jednoduché, pretože stačí overiť, či daný bod leží v okne alebo je mimo okna. Orezávanie mnohouholníkov a úsečiek je zložitejšie, pretože potrebujeme viac výpočtov. Pri vykresľovaní textu môžeme použiť viacero postupov.

3.2 Body v okne

Predpokladáme, že okno je zadané súradnicami x_{min}, x_{max}, y_{min}, y_{max}. Potom bod (x, y) je viditeľný len v tom prípade, ak sú splnené tieto nerovnosti:

$$x_{min} \leq x \leq x_{max},\ y_{min} \leq y \leq y_{max}\ .$$

Ak jedna z nerovností nie je splnená, potom bod nie je viditeľný, t.j. nebude sa zobrazovať.

3.3 Orezávanie úsečky

Pri orezávaní úsečky vo všeobecnom tvare potrebujeme viac porovnaní a výpočtov ako pre bod. Rýchlo vylúčime prípady, keď koncové body úsečky ležia vo vnútri okna a celá úsečka sa vykreslí. Pre úsečku s koncovými bodmi mimo okna je nutné preveriť, či sa pretína s oknom. Ak áno, musíme ju orezať. Jeden spôsob orezávania je vyhľadanie bodov prieniku úsečky s hranicami okna.

Rýchlosť operácie orezávania je v interaktívnej grafike dosť podstatná, pretože pre generovanie obrázka je nutné orezávať aj tisíce úsečiek. Efektívnejší algoritmus by mal

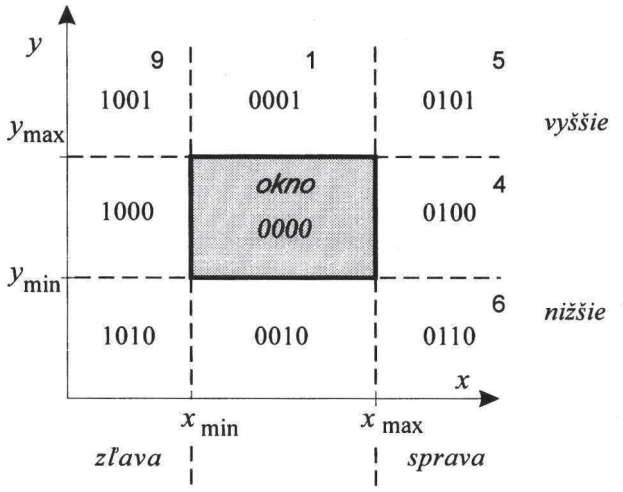
zo začiatku urobiť niektoré doplňujúce porovnania, aby sa zistilo, či je skutočne nutné vyhľadať body prieniku. Po prvé, môžeme vykresliť úsečky, pre ktoré koncové body ležia v okne. Po druhé, ak koncové body úsečky majú súradnicu y väčšiu ako y_{max} príslušného okna, potom celá úsečka leží nad oknom a úsečka sa nevykreslí, pretože je mimo okna. Takisto sa nevykreslia ani úsečky, ktoré ležia celé pod oknom, resp. vľavo od okna alebo vpravo od okna.

3.3.1 Algoritmus orezávania Cohena-Sutherlanda

Tento algoritmus rýchlo vylúči vyššie spomenuté prípady. Zvlášť rýchly je v prípade okna, ak obsahuje veľa úsečiek vo vnútri a taktiež pri takom okne, ak väčšina úsečiek je mimo okna. V týchto prípadoch sa úsečka podľa polohy buď celá zobrazí alebo nezobrazí.

Na začiatku algoritmu nastavíme 4-bitové hodnoty (kód) pre obidva koncové body úsečky podľa polohy vzhľadom na okno. Pre každú hraničnú priamku okna nastavíme jeden bit podľa toho, či leží bod v danej polrovine (pozri obr. 3.1). Pravidlo upresníme podľa polohy bodu vzhľadom na okno:

1. bit - bod leží vyššie od okna ($x_{max} < x$);
2. bit - bod leží nižšie od okna ($x_{min} < x$);
3. bit - bod leží sprava od okna ($y_{max} < y$);
4. bit - bod leží zľava od okna ($y_{min} < y$).



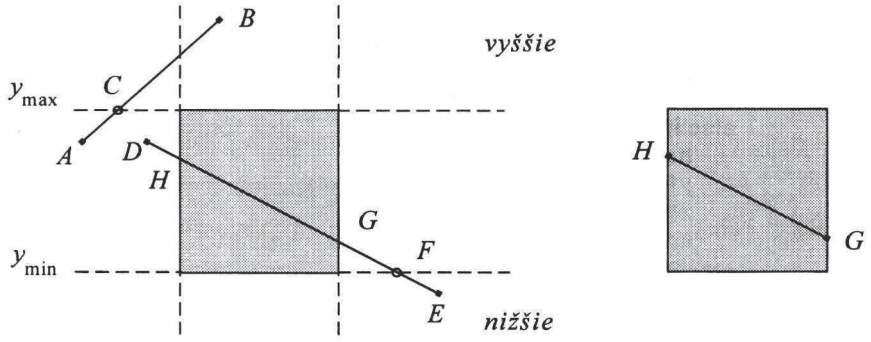
Obr. 3.1 Kódovanie jednotlivých častí roviny

Bod sa nachádza vo vnútri okna, ak jeho kód je rovný (0000). Úsečka je celá v okne, ak oba koncové body majú kód rovný (0000). Ak je celá úsečka dole, hore, vľavo alebo vpravo od okna, potom je celá úsečka mimo okna. Ľahko sa identifikuje podľa kódov koncových bodov, pretože je nastavený bit na rovnakom mieste. V tomto prípade stačí

pre body overiť, či ich logický bitový súčin je rôzny od kódu (0000). Naopak, ak je logický súčin kódov rovný 0000, tak úsečku nemôžeme vylúčiť, pretože ešte môže mať prienik s oknom. Preto pri každej iterácii overíme úsečku vzhľadom na hranicu okna (ľavú, pravú, hornú a dolnú). Ak ju pretína, potom časť úsečky po orezaní leží mimo okna a druhá časť sa ďalej skúma. Treba podotknúť, že táto časť úsečky nemusí ležať celá v okne, a preto musíme postup opakovať.

Ukážeme si na príklade, ako sa budú orezávať úsečky *AB* a *DE* (pozri obr. 3.2). Najprv zistíme kódy bodov *A* a *B*. Úsečku nemôžeme jednoduchým testom vylúčiť, a preto musíme počítať prienik najprv s hornou hranicou okna. Úsečka *AB* sa rozdelí na dve časti *AC* a *CB*, časť *CB* vylúčime. V ďalšom opakovanom kroku vylúčime jednoduchým testom zostávajúcu časť úsečky *AC*, a preto sa úsečka *AB* nezobrazí. Pre úsečku *DE* musíme hľadať prienik najprv s dolnou hranicou okna a rozdeliť ju na úsečky *DF* a *FE*. Po orezaní ostane úsečka *DF*. Po druhom opakovaní testov orežeme sprava a dostaneme úsečku *DG*. Postupne orežeme úsečku *DG* zľava a ostane nám úsečka *HG*, ktorá sa nachádza celá v okne. Ako vidíme na obr. 3.2 úsečku *HG* vykreslíme.

Poradie hrán okna pre jednu iteráciu môžeme vybrať ľubovoľne. Napríklad v nasledujúcom algoritme zodpovedá poradie bitov v kóde a postupnosti orezávania: zhora, zdola, sprava, zľava.



Obr. 3.2 Príklady orezania niektorých úsečiek

V nasledujúcej časti zavedieme označenie a popis funkcií pre algoritmus zapísaný v pseudokóde. Súradnice (x_{min}, x_{max}, y_{min}, y_{max}) definujú okno, vzhľadom na ktoré budeme orezávať úsečky. Koncové body úsečky sú $P1=(x1, y1)$ a $P2=(x2, y2)$.

Procedúra *outcod* nastaví bity do kódu podľa súradníc daného bodu vzhľadom na okno. Boolovská premenná *accept* nás informuje, či daná úsečka sa má vykresľovať. Premenná *done* slúži na ukončenie cyklu **repeat**..**until**. Funkcia *and* dáva logický súčin kódov. V 3. kroku algoritmus zisťuje, či úsečka nie je mimo okna podľa spomenutých kritérií a v 4. kroku, či úsečka je vnútri okna. Procedúra *swap* vymieňa body a ich kódy v 5. kroku. Postupne od 6. až po 9. krok zisťujeme, či úsečka je celá zhora, zdola, sprava alebo zľava od okna. Ak áno, potom vykonáme potrebné orezanie úsečky do okna.

Algoritmus Cohen-Sutherlanda

```
Procedure Clipping;
begin
1.  accept:= false;                                  { nastav - P1P2 sa nevykresľuje }
    outcod (x2, y2, cd2);                            { zistíme kód bodu P2 }
2.  repeat
        outcod (x1, y1, cd1);                        { zistíme kód bodu P1 }
3.      if and (cd1, cd2) ≠ 0 then done:= true       { 1. jednoduchý test - mimo okna}
        else
4.          begin
            if (cd1=0 and cd2=0) then                { 2. jednoduchý test - vnútri okna}
                        begin accept:= true;         { žiadaj vykreslenie v kroku 10. }
                              done:= true end
            else
5.              begin
                if cd1=0 then swap(P1, P2);          { zameníme body, aby 1. bol von}
6.              if cd1 ∈ (1, 5, 9) then              { orezávanie zhora }
                    begin
                    x1:= x1 + (x2-x1)*(ymax-y1)/(y2-y1);
                    y1:= ymax ;
                    end
7.              else if cd1 ∈ (2, 6, 10) then        { orež zdola }
                    begin
                    x1:= x1 + (x2-x1)*(ymin-y1)/(y2-y1);
                    y1:= ymin ;
                    end
8.              else if cd1 ∈ (4, 5, 6) then         { orež sprava }
                    begin
                    y1:= y1 + (y2-y1)*(xrnax-x1)/(x2-x1);
                    x1:= xmax ;
                    end
9.              else if cd1 ∈ (8, 9, 10) then        { orež zľava }
                    begin
                    y1:= y1 + (y2-y1)*(xmin-x1)/(x2-x1);
                    x1:= xmin ;
                    end
                end             { od kroku 5 }
            end             { od kroku 4 }
    until done
10. if accept then draw(P1, P2);                     { vykresli úsečku }
end.
```

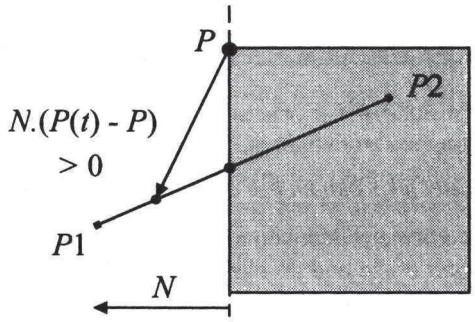
3.3.2 Parametrické orezávanie

Nevýhodou Cohen-Sutherlandovho algoritmu orezávania bolo, že sa orezávalo aj v prípade, keď to nebolo nutné. To znamená, že niektoré úsečky sme orezávali štyrikrát. V roku 1978 uviedli Cyrus a Beck algoritmus [CYRU78], ktorý odstraňoval tento nedostatok a orezával úsečku maximálne dvakrát, a to pri vstupe do okna a pri výstupe z okna. Neskôr v roku 1984 uviedli Liang a Barsky [LIAN84]ešte lepšie upravený algoritmus. Ukážeme základnú myšlienku algoritmu Cyrus-Becka.

Označíme D smerový vektor úsečky $P1P2$ a zapíšeme úsečku v parametrickom tvare:

$$P(t) = P1 + t.D, \qquad (1)$$

kde parameter $t \in \langle 0, 1 \rangle$. Podľa obr. 3.3 označme bod na hrane P a normálový vektor hraničnej priamky okna N.



Obr. 3.3 Skalárny súčin identifikuje, či je bod zvonku alebo zvnútra

Ak bod úsečky $P(t)$ sa nachádza mimo okna, potom normálový vektor N s vektorom $(P(t) - P)$ zviera uhol menší ako 90° a skalárny súčin je kladný. Ak bod $P(t)$ leží na hrane e, potom môžeme vypočítať parameter t:

$$N.(P(t) - P) = 0.$$

Použijúc (1) zapíšeme nasledujúci vzťah pre bod na hrane:

$$N.(P1 + t.\ D - P) = N.(P1 - P\ ) + t\ N.D,$$

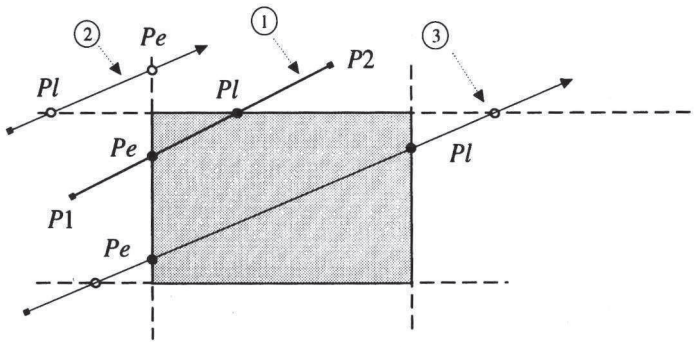
$$t = \frac{N.(P1-P)}{-N.D}. \qquad (2)$$

Skalárny súčin $N.D$ je rovný nule, ak úsečka je rovnobežná s hranou e. Označíme bod vstupu Pe (*potentially entering*) a bod výstupu Pl (*potentially leaving*) úsečky. Za hrany postupne volíme hraničné úsečky e_i okna a ich normálové vektory N_i. Z predchádzajúcich úvah vyplýva:

1. ak $N_i.D > 0$, potom úsečka vstupuje bodom Pe, (3)

2. ak $N_i.D < 0$, potom úsečka vystupuje bodom Pl.

Na obrázku 3.4 prvá úsečka má prienik práve v dvoch bodoch s oknom, časť úsečky $PePl$ je určená parametrami te a tl. Tieto parametre počítame zo vzťahu (2). Zohľadňujúc pravidlo (3) pri výpočte parametrov úsečky sa môže stať, že pre jej parametre platí $te > tl$. To je presne prípad druhej úsečky. Takúto úsečku nevykresľujeme. Na tretej úsečke vidíme, že musíme pre parameter te vyberať maximum a naopak pre tl minimum.



Obr. 3.4 Prienik úsečiek s oknom pre algoritmus Cyrus-Beck

V nasledujúcej časti uvedieme pseudokód algoritmu Cyrus-Becka podľa uvedeného označenia. Funkcia *clip_point* oreže bod do okna. V cykle **for** (3. krok algoritmu) volíme postupne hrany okna e_i a ich koncový bod P_i pre výpočet parametra *te* a *tl*.

Algoritmus Cyrus-Becka

```
Procedure Cyrus-Beck;
begin
1.  if (P1 = P2) then clip_point;                 { úsečka je bod, orež bod }
    else
    begin
2.      te = 0;                                   { príprava parametrov }
        tl = 1;
3.      for e_i (i = 1 to 4)                      { pre každú hranu okna }
            begin
            nd = N_i.D;
4.          if nd ≠ 0 then  t = N.(P1-P_i)/nd;    { výpočet parametra }
            if nd > 0 then  te = max (te, t)      { uprav parameter potenc. vstupu }
        if nd < 0 then  tl = min (tl, t)          { uprav parameter potenc. výstupu }
            end
5.      if te > tl then return nil;               { úsečka mimo okna }
        else
    return   (P(te), P(tl));                      { výstup - orezaná úsečka }
    end
end
```

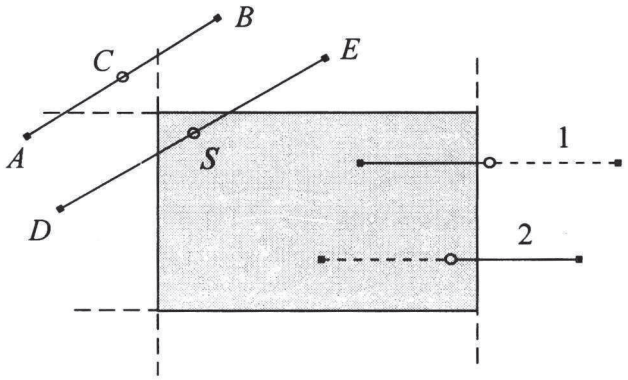
Existujú aj iné algoritmy orezávania. Ich hlavný dôraz sa obvykle kladie na väčší počet jednoduchých aritmetickych výpočtov.

3.3.3 Algoritmus postupného delenia

V algoritme Cohena-Sutherlanda sa na výpočet súradnice používa násobenie a delenie. Pokiaľ procesor nemá pre násobenie a delenie v pohyblivej rádovej čiarke špeciálne inštrukcie, potom je výhodnejšie z časových dôvodov použiť nasledovný algoritmus. Jeho základný princíp spočíva v prevedení úsečky z užívateľských súradníc na celočíselné súradnice, a potom postupnom delení úsečky na polovicu.

V tomto algoritme môže počet iterácií prevýšiť predchádzajúci algoritmus, avšak samotné výpočty sú podstatne rýchlejšie. Obdobne definujeme pre body ich 4-bitové hodnoty (kódy). V prvom kroku iterácie predpokladáme pre úsečku, ktorú nemôžeme ani vylúčiť ani celú zobraziť, že je len jeden priesečník úsečky s oknom. Súradnice stredu úsečky ľahko vypočítame aj bez operácie delenia, a to pomocou súčtu a bitového posunu (delenie 2) ako $(x1+x2)/2$, resp. $(y1+y2)/2$. Pre každú časť úsečky rozdelenú na polovicu overujeme, či nemôžeme niektorú z nich vylúčiť jednoduchým testom, či leží nad alebo pod oknom, vpravo alebo vľavo od neho. Stratégia metódy spočíva v opakovaní delenia úsečky na polovicu. V každom kroku sa obvykle jedna polovica úsečky ponechá pre ďalšie skúmanie a druhá vylúči.

Vysvetlíme jednoduché príklady. Na obr. 3.5 úsečka *AB* má stred úsečky *C* mimo okna. Po rekurzívnom zavolaní zistíme, že obidvoma časťami *AC* a *CB* sa už nebudeme ďalej zaoberať, lebo sa nachádzajú jednoznačne mimo okna. Pre prvú úsečku na obr. 3.5 po rozdelení na polovicu budeme skúmať len ľavú polovicu, lebo pravá je mimo okna (nezobrazí sa). Pre druhú úsečku budeme skúmať len pravú polovicu, pretože ľavá časť je celá v okne a zobrazí sa.



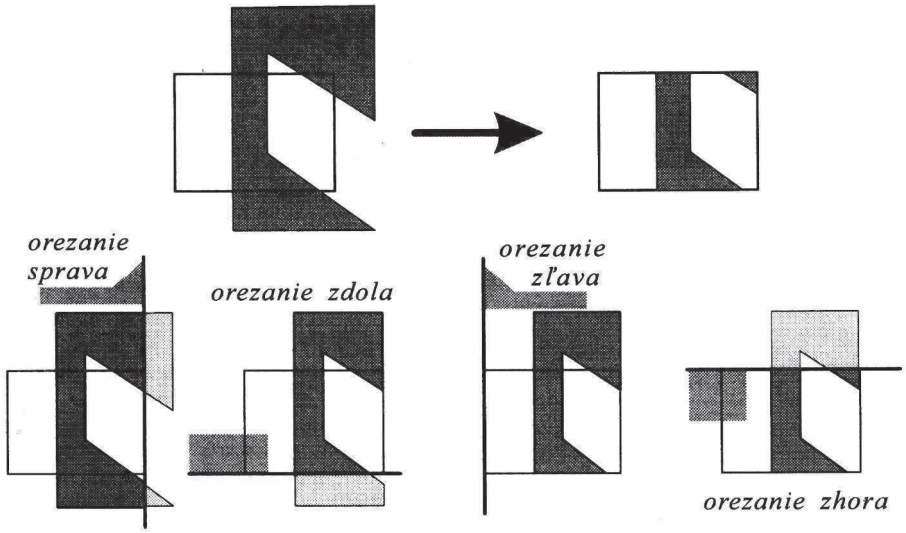
Obr. 3.5 Príklady orezania niektorých úsečiek delením na polovicu

Pri tomto postupe môže nastať zdanlivo problém, ako v prípade úsečky *DE* na obr. 3.5, že stred úsečky leží v okne, hoci krajné body sú mimo okna. Je zrejmé, že nemôžeme vylúčiť ani jednu úsečku *DS* ani *SE*. Ponúka sa jednoduché riešenie, môžeme rekurzívne riešiť osobitne úsečku *DS* a *SE* (t.j. najprv jednu časť a potom druhú časť). Opakovanie tejto situácie (po rekurzívnom zavolaní nevypadne z riešenia ani jedna časť úsečky) už nemôže viackrát nastať. To znamená, že v binárnom delení úsečky sa riešia maximálne dve cesty binárneho stromu (častejšie len jedna).

V najhoršom prípade sa algoritmus skončí za log (n) krokov, kde n je maximálna dĺžka strany okna. Napríklad, ak pripustíme celočíselné súradnice z intervalu $\langle 0, 1023 \rangle$, maximálne po 10 krokoch nájdeme orezanú úsečku.

3.4 Prienik polroviny s mnohouholníkom

Základná myšlienka Sutherland-Hodgmanovho algoritmu orezávania mnohouholníka do okna spočíva v tom, že mnohouholník orezávame postupne jednotlivými priamkami tvoriacimi hranicu okna (pozri obr. 3.6).

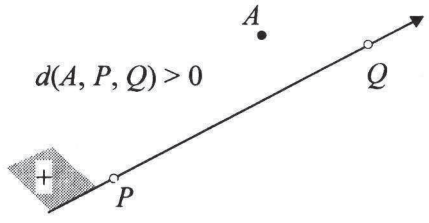


Obr. 3.6 Postupné orezávanie do okna

Preto vyšetrujeme prienik polroviny danej oknom s mnohouholníkom. Predpokladáme, že máme zadanú orientovanú priamku PQ a kladne orientovaný mnohouholník. Tento mnohouholník je zadaný vrcholmi $A(1)$, ..., $A(n)$ a priamka bodmi P, Q. Skôr ako budeme riešiť prípad prieniku, zadefinujeme funkciu troch bodov $d(A, P, Q)$ nasledujúcim predpisom:

$$d(A, P, Q) = xa.(yp\text{-}yq) + ya.(xq\text{-}xp) + xp.yq - yp.xq,$$

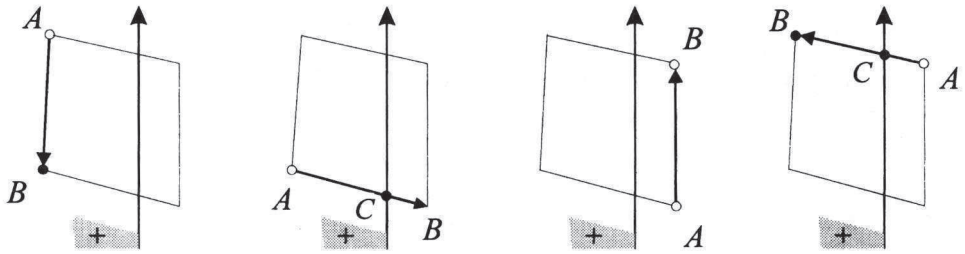
kde $A = (xa, ya)$, $P = (xp, yp)$, $Q = (xq, yq)$. Znamienko hodnoty d určuje polohu bodu A k orientovanej priamke PQ. Na obr. 3.7 máme zobrazenú situáciu, kde body naľavo od PQ majú hodnotu $d(A, P, Q)$ kladnú a napravo od PQ majú hodnotu $d(A, P, Q)$ zápornú.



Obr. 3.7 Poloha bodu A vzhľadom k orientovanej priamke PQ

Zaujíma nás prienik polroviny s mnohouholníkom. Skúmame polohu vrcholov mnohouholníka k polrovine PQ. Označíme hodnoty $s(i) = d(A(i), P, Q)$, čím dostaneme vo vrcholoch $A(i)$ znamienka $s(i)$. Na určenie prieniku nám stačí vyšetriť tie hrany, pre ktoré hodnoty $s(i)$ a $s(i+1)$ sú rôzne, čo indikuje, že vrcholy $A(i)$, $A(i+1)$ sú oddelené priamkou PQ. Pre daný mnohouholník budeme vytvárať nový mnohouholník tak, že budeme postupne na výstup zapisovať len tie vrcholy, ktoré ležia v danej polrovine. Koncový bod vyšetrovanej hrany, ktorá celá leží v polrovine, zapíšeme do zoznamu nových vrcholov. V prípade hrany mimo danej polroviny, nezapíšeme žiaden bod.

Označme vyšetrovanú hranu AB a $r = d(A, P, Q)$, $s = d(B, P, Q)$. Ostávajú dva prípady, buď hrana vychádza z polroviny alebo vchádza do polroviny. Bod prieniku hrany s priamkou PQ označme ako C. Ak hrana vychádza (prípad obr. 3.8 b), potom do zoznamu vrcholov zapíšeme bod C. Ak hrana vchádza (prípad obr. 3.8 d), potom do zoznamu vrcholov zapíšeme body C aj B.



a) b) c) d)

Obr. 3.8 Klasifikácia úsečky AB vzhľadom na polpriamku

Pre 3 možné hodnoty znamienka r a s (+, 0, -) nastáva 9 možností, ktoré nasledujúce kritériá rozlišujú z hľadiska výstupu. Môžeme použiť tieto kritériá pre zápis bodov (porovnaj časti obrázku 3.8):

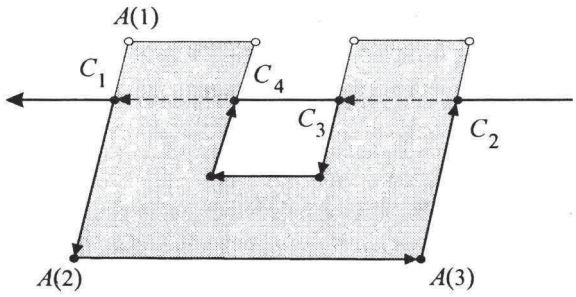
1. ak ($r > 0$) a ($s > 0$), potom zapíšeme koncový bod B; (časť a)

2. ak ($r > 0$) a ($s <= 0$), potom zapíšeme prienikový bod C; (časť b)

3. ak ($r <= 0$) a ($s > 0$), potom zapíšeme body B a C; (časť d)

4. inak nezapíšeme žiaden bod. (časť c)

Prienikom mnohouholníka s polrovinou môže byť nielen jeden, ale aj viac mnohouholníkov (pozri na obr. 3.6). Preto našou úlohou je nielen určiť nové body, ale tiež tieto body usporiadať do zoznamov určujúcich mnohouholníky. Môžeme to riešiť tak, že pre vychádzajúcu hranu určíme najbližšiu vchádzajúcu hranu. Ponúka sa nám jednoduché riešenie: všetky prienikové body C usporiadame v orientovanom smere priamky PQ. Najbližšiu vchádzajúcu hranu určíme podľa tohto usporiadania bodov C. Pre vyhľadanie najbližšieho bodu je výhodné vytvoriť smerníkovú štruktúru.



Obr. 3.9 Znázornenie smerníkov pre prienik mnohouholníka s priamkou

Pre každý bod mnohouholníka $A(i\text{-}1)$ budeme mať smerník označený ako $next(A(i\text{-}1)) = A(i)$. V programe to môžeme zrealizovať poľom $A(n, 3)$, kde pre každý bod tretia zložka bude obsahovať index nasledujúceho bodu. V prípadoch, že úsečky vchádzajú alebo vychádzajú z danej polroviny, nevieme smerníky jednoznačne určiť. Presnejšie, ak hrana $A(i\text{-}1)A(i)$ vychádza, potom vieme zadať len $next(A(i\text{-}1)) = C$ a $next(C)$ nevieme. V prípade, že hrana $A(i\text{-}1)A(i)$ vchádza, vieme zadať $next(C) = A(i)$, ale nevieme, z ktorého bodu sa dostaneme do bodu C.

Preto až po vyhľadaní všetkých prienikových bodov, smerníkovú štruktúru jednoducho uzavrieme. Usporiadame všetky body C na priamke PQ a tieto pospájame do nasledujúcich dvojíc. Je to názorne vidieť na obrázku 3.9, kde pre nové body C_1, C_2, C_3 a C_4 sme dodali tieto nové smerníky

$$next(C_4) = C_1 \text{ a } next(C_2) = C_3 . \qquad (4)$$

Nakoniec prehľadávame smerníkovú štruktúru, a tak vytvárame nové mnohouholníky.

V nasledujúcej časti algoritmus rozpíšeme. Vrcholy mnohouholníka sú zapísané na začiatku poľa $A(m, 3)$, pričom dimenzia m je väčšia ako počet vrcholov n. Za posledným bodom budeme ukladať nové body prieniku. Priamka je zadaná bodmi P a Q. Do poľa $t()$ budeme ukladať parametre t, ktoré určíme pri prieniku priamky PQ s hranou mnohouholníka. V poli $m()$ budeme mať usporiadané indexy podľa hodnôt parametra $t()$, aby sme mohli nakoniec určiť smerníky pre nové body. Do poľa $B(m,2)$ ukladáme za sebou mnohouholníky.

Procedúra *intersection* vypočíta bod prieniku hrany $A(i\text{-}1)A(i)$ mnohouholníka s hraničnou priamkou PQ polroviny. Spočítaný bod prieniku odloží procedúra *write* na koniec matice A a parameter t do poľa $t()$. *Next* je smerník pre cyklickú štruktúru pre vytvorenie mnohouholníka.

Algoritmus prieniku mnohouholníka s polrovinou

```
Procedure Clipping;
begin
        { 1. časť : vyhľadáme body prieniku }

1.  s:= d(A(1), P, Q);
    k:= 0;                                  { index pre body C }
2.  for i:= 2 to n do
```

```
    begin
3.      r:= s;                                  { zameníme d pre A(i-1) }
        s:= d(A(i), P, Q);                      { pripravíme d pre A(i) }
4.      if r > 0 and s > 0 then next(A(i-1)):= i;   { prípad a) - potvrď smerník }
5.      else if r > 0 and s <= 0 then begin
                                  intersection(C);  { počítaj prienik A(i-1)A(i) s PQ }
                                   write(C, A, t);  { zapíš bod C na (n+k) miesto A a t() }
                                   next(A(i-1)):= n+k;  { prípad b) - smerník na C }
                                        end;
6.      else if r <= 0 and s > 0 then begin
                                intersection(C, t);  { počítaj prienik A(i-1)A(i) s PQ }
                                write(C, A, t);      { zapíš bod C na (n+k) miesto A a t() }
                                next(A(n+k)):= i;    { prípad d) - smerník z bodu C }
                                        end;
7.  end

        { 2. časť : uzatvoríme smerníkovú štruktúru }

8.  { Do matice A sa zapísalo k nových bodov C a do poľa t() parametre priamky PQ }
    { Usporiadame hodnoty v poli t() a zároveň podľa nich uložíme indexy do poľa m()};
9.  { Podľa vybraných dvojíc uzavrieme smerníkovú štruktúru v poli A() }
    for i:= 1 to k { step 2 } do
    begin
10.     next(A(n+m(i)):= n+m(i+1);              { pozri obr. 3.9 a vzťah (4) }
    end

        { 3. časť : vytvoríme nové mnohouholníky }

    j:= 0;                              { priprav index do poľa B pre nové mnohouholníky }
    repeat
11. i:= 1;                                  { začni prezerať od začiatku pole A() }
        repeat
12.     i:= i+1;                            { vyhľadaj nenulový smerník }
        until ( next(A(i)) ≠ 0 or i = n )   { opakuj pokiaľ ešte máme nenulový smerník }
13. i1:= i;                                 { priprav 1. index pre nový mnohouholník }
14. if i1 ≠ n then                          { vyhľadaj v cykle }
            repeat                          { celý nový mnohouholník }
15.         B(j):= A(i); temp:= i;          { odlož vrcholy do poľa B }
                i:= next(A(i)); next(A(temp)):= 0;
            until (i=i1);
    until (i1 = n)
end.
```

3.5 Prienik dvoch mnohouholníkov

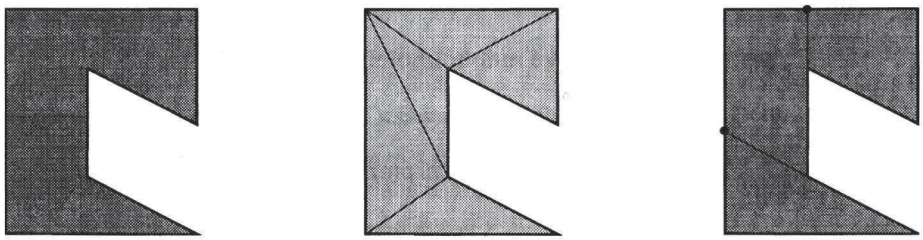
Ak máme dva mnohouholníky a jeden z nich je *konvexný*, potom úlohu môžeme previesť na predchádzajúci prípad. **Konvexný mnohouholník** sa dá popísať pomocou prieniku polrovín, ktoré sú dané jeho hraničnými priamkami (podobne ako na obr. 3.6).

Použijeme predchádzajúci algoritmus postupne pre jednotlivé polroviny. Výsledný jeden mnohouholník alebo viac výsledných mnohouholníkov tvoria prienik týchto dvoch mnohouholníkov.

Pre **dva nekonvexné mnohouholníky** je zrejmé, že tento postup nemôžeme aplikovať. Ukážeme hlavnú myšlienku, ako postupovať v takomto prípade. Jeden z týchto nekonvexných mnohouholníkov rozložíme na **zjednotenie niekoľkých konvexných mnohouholníkov**. Potom sa stačí opäť vrátiť k prieniku mnohouholníka s konvexnými časťami druhého mnohouholníka a nakoniec ich zjednotiť.

Geometricky jednoduchšie riešenie rozkladu mnohouholníka na konvexné časti je urobiť jeho **rozklad na trojuholníky**. Avšak počet trojuholníkov môže byť neúmerne veľký, a preto je zaujímavé minimalizovať počet konvexných častí. Takýto algoritmus je efektívny, ale jeho praktické využitie je problematické pre jeho algoritmickú náročnosť.

Ukážeme pomerne jednoduché riešenie: budeme **odstraňovať nekonvexné uhly**. Pre každý vrchol mnohouholníka, v ktorom je uhol väčší ako 180 stupňov, nájdeme prienik jednej hraničnej polpriamky tvoriacej tento uhol s najbližšou hranou mnohouholníka (obr. 3.10). Takýmto postupom sa bude zmenšovať počet nekonvexných uhlov, až nakoniec dostaneme rozklad na konvexné mnohouholníky. Tento postup má však jeden nedostatok, pri rozklade nám budú pribúdať ďalšie vrcholy, ktoré neboli na vstupe, čo spôsobuje rast asymptotickej zložitosti algoritmu.



Obr. 3.10 Rozklad mnohouholníka na konvexné časti

3.6 Prístupy k orezávaniu textu

Text je možné orezávať niekoľkými spôsobmi. Záleží na tom, či znaky reprezentujeme ako **plošné** alebo **čiarové** prvky. Plošné sú definované pomocou bitovej mapy a čiarové sú definované po úsečkách. Ak by sme každé písmeno orezávali po úsečkách alebo krivkách, ktoré ho vytvárajú, dostali by sme presné zobrazenie textu v okne, ale časovo náročné. Niekedy je výhodnejšie využívať hardverové generovanie znakov, pri ktorom sa musíme pozerať pri orezávaní textu na každý znak ako na nedeliteľný objekt. Každý znak je obsiahnutý v minimálnom obdĺžniku, ktorý nazveme **obálkou znaku**. Obyčajne určitý bod tejto obálky (napríklad stred) sa vezme za referenčný bod, podľa ktorého sa určí, či príslušný znak je v okne alebo nie. Podľa toho sa príslušný znak zobrazí alebo nezobrazí. Môžeme prijať na zobrazenie podmienku: "obálka znaku celá leží v okne".

Tretí, najjednoduchší spôsob je orezávanie textu podľa slov alebo celého riadku, ktorý považujeme za nedeliteľný. V takomto prípade sa buď celé slovo zobrazí, alebo nezobrazí (pozri obr. 3.11). Môžeme si vybrať podmienku, že celý obdĺžnik obsahujúci slovo musí ležať v okne.

Obr.3.11 Orezávanie textu podľa rôznych podmienok

3.7 Záver

Doteraz sme predpokladali, že okno má strany rovnobežné so súradnicovými osami. Ako riešiť situáciu pre otočené okno? Je možné počítať prieniky úsečky s priamkami, ktoré tvoria hranicu okna. Alebo je možné počítať s tým, že sa dajú využiť rýchlejšie algoritmy pre priamky rovnobežné so súradnicovými osami. Miesto pôvodného okna sa definuje obálka, ktorá obsahuje definované okno. Lenže, ak má úsečka prienik s obálkou, potom musíme ešte zistiť presne body prieniku s pôvodným oknom. Ak pre záber sú hraničné priamky rovnobežné s osami, potom pre natočené okno je výhodnejšie orezávať úsečky až v zábere.

Často sa vyskytujú dve analogické úlohy:

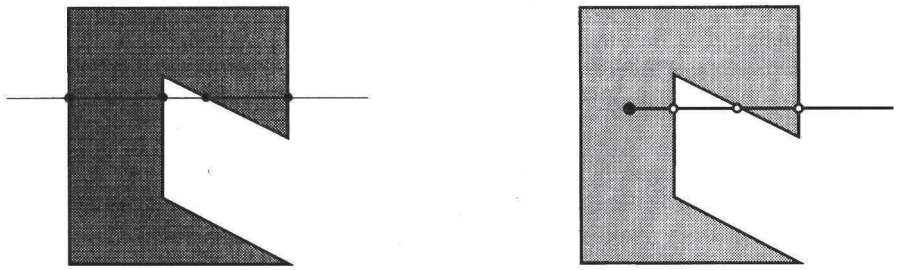
1. vyhľadanie vnútorného bodu mnohouholníka,

2. určenie, či daný bod je vnútorný.

Pre konvexné mnohouholníky sú obidve úlohy jednoduché. Napríklad ťažisko konvexného mnohouholníka je vnútorným bodom. Daný bod P je vnútorný pre kladne orientovaný konvexný mnohouholník, ak pre každú hranu AB platí, že hodnota $d(P, A, B)$ je kladná.

Pre nekonvexný mnohouholník musíme vybrať takú priamku, ktorá má spoločné body s mnohouholníkom a pritom neprechádza ani cez jeden jeho vrchol. Na tejto priamke vyhľadáme všetky body prieniku s mnohouholníkom a podľa usporiadania na priamke ich spájame do dvojíc. Medzi každou takouto dvojicou bodov sa nachádzajú vnútorné body mnohouholníka (pozri obrázok 3.12 a). Pre daný bod vyhľadáme takú polpriamku z daného bodu, aby neprechádzala cez vrchol mnohouholníka. Určíme počet spoločných

bodov mnohouholníka s polpriamkou. Ak je tento počet nepárny, potom daný bod je vnútorný (obr. 3.12 b).



a) b)

Obr. 3.12 Určenie vnútorného bodu mnohouholníka

4

Krivky a plochy

4.1 Vyjadrenie kriviek a plôch

V počítačovej grafike sa stretávame s rôznymi spôsobmi zadávania kriviek a plôch. V tejto časti si ukážeme niektoré spôsoby zadania:

- **analytické zadanie** krivky a plochy (explicitný, implicitný a parametrický spôsob),
- **interpolačné zadanie** krivky a plochy (zadané sú body, ktorými má krivka alebo plocha prechádzať),
- **aproximačné zadanie** krivky a plochy (priblížime sa k zadaným bodom krivky alebo plochy, ako napr. u Bézierovej alebo B-splajnovej (*B-spline*) krivky a plochy).

V geometrickom modelovaní sa stretávame jednak s grafmi funkcií dvoch reálnych premenných, ale omnoho častejšie sú v praxi technické objekty popísané parametrickými vyjadreniami kriviek a plôch (karosérie automobilov, trupy lodí, lopatky turbín).

Kvalitatívnou zmenou pri návrhových systémoch CAD bol prechod na využívanie racionálnych Bézierových kriviek a plôch. Pomocou nich sa dajú jednotne generovať aj klasické geometrické objekty ako sú kružnice, kužeľosečky a pod., pritom však poskytujú veľmi silný nástroj pre tvarovanie technických kriviek a plôch. V tejto kapitole sa však nebudeme nimi zaoberať, pretože presahujú jej rámec. Podrobnosti možno nájsť v práci [BARS88] a [FARI88].

4.2 Geometrické modelovanie kriviek

V tejto časti sa sústredíme na vyjadrovanie kriviek. Z interpolačných kriviek spomenieme Fergusonovu kubiku a splajnové interpolačné krivky. Z aproximačných kriviek Bézierove krivky a B-splajnové krivky.

4.2.1 Analytický spôsob zadania kriviek

Na analytický popis krivky môžeme využiť vyjadrenie explicitné, implicitné, či parametrické. V počítačovej grafike sa práve najviac využíva parametrické zadanie, pretože dáva jednotiace hľadisko pre rovinné aj priestorové krivky. Ukážeme si na jednoduchom príklade rôzne vyjadrenia elipsy.

Príklad 1. Uvažujme, že máme zadanú elipsu so stredom v začiatku sústavy súradníc, s poloosou a na osi x a b na osi y. Jej zodpovedajúce vyjadrenia sú:

- **explicitné**

$$y = b \cdot \sqrt{1-(x^2/a^2)}\,, \text{ a } \quad y = -b \cdot \sqrt{1-(x^2/a^2)}\,, \quad x \in \langle -a, a \rangle$$

- **implicitné**

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

- **parametrické**

$$x = a \cdot \cos t, \quad y = b \cdot \sin t, \quad t \in \langle 0, 2\pi \rangle.$$

Vo všeobecnosti explicitné vyjadrenie krivky je zadané funkciou v tvare $y = f(x)$. Implicitné zadanie krivky máme v tvare $F(x, y) = 0$ a parametrické vyjadrenie je $x = x(t), y = y(t)$.

4.2.2 Interpolačné krivky

V numerickej matematike sa používa interpolácia hlavne polynómami, pretože sa obvykle s nimi pracuje jednoduchšie. Napríklad, na tomto princípe pracujú algoritmy pre výpočet určitého integrálu. V počítačovej grafike interpolácia slúži hlavne na zostrojovanie kriviek pre zadané určujúce body. Interpolačná krivka prechádza týmito diskrétnymi bodmi. Problémom býva určiť hodnotu parametra, pre ktorý získame daný určujúci bod interpolačnej krivky.

Interpolačná krivka je teda zadaná určujúcimi (spojovacími) uzlovými bodmi P_0, P_1,..., P_n a hodnotami parametrov t_0, t_1, ..., t_n pre tieto body. Potom pre bodové (niekedy tiež nazývané vektorové) parametrické vyjadrenie:

$$\boldsymbol{P} = \boldsymbol{P}(t), \quad t \in \langle t_0, t_n \rangle,$$

kde pre $i = 0,..., n$, platí $P_i = \boldsymbol{P}(t_i)$.

Hodnoty parametra t_i sa väčšinou určujú automaticky. Ak je rozdiel $t_{i+1} - t_i$ *konštantný*, hovoríme o **uniformnej parametrizácii** interpolačnej krivky. Najčastejšie sa v tomto prípade volí $t_i = i$. Pre neuniformnú parametrizáciu sa často používa vzťah:

$$t_{i+1} = t_i + k.|P_{i+1} - P_i|,$$

kde k je určitá konštanta, t.j. prírastok parametra medzi dvoma uzlami je úmerný dĺžke príslušnej tetivy krivky. Neuniformná parametrizácia sa používa hlavne v tom prípade, ak určujúce body sú nerovnomerne rozložené. Interpolačná krivka sa spravidla vytvára z jednotlivých oblúkov. Základom pre konštrukciu interpolačnej krivky po oblúkoch je **Fergusonova kubika**.

Fergusonova kubika

Nech sú dané body P_i a P_{i+1}. Označme ďalej $\boldsymbol{P}'_i$ a $\boldsymbol{P}'_{i+1}$ dotykové vektory hľadanej krivky v týchto bodoch. Hľadáme polynomické funkcie tretieho stupňa určujúce krivku s koncovými bodmi P_i, P_{i+1} a dotykovými vektormi v nich. Nech daným bodom zodpovedajú hodnoty parametra t_i a t_{i+1}. Predpokladajme, že h_0, h_1, h_2, h_3 sú polynomické

funkcie tretieho stupňa premennej $s = t - t_i$, pomocou ktorých vyjadríme Fergusonovu kubiku v parametrickom tvare:

$$\boldsymbol{P}(t) = h_0(s).P_i + h_1(s).P_{i+1} + h_2(s).\mathbf{P}'_i + h_3(s).\mathbf{P}'_{i+1}. \qquad (1)$$

Z podmienok $P_i = \boldsymbol{P}(t_i)$, $P_{i+1} = \boldsymbol{P}(t_{i+1})$, $\boldsymbol{P}'_i = \boldsymbol{P}'(t_i)$, $\boldsymbol{P}'_{i+1} = \boldsymbol{P}'(t_{i+1})$ určíme hodnoty koeficientov pri jednotlivých mocninách premennej u funkcií h_0, h_1, h_2, h_3. Aby sme jednoduchšie vyjadrili funkcie h_0, h_1, h_2, h_3, položme pre rozdiel parametrov $k = t_{i+1}-t_i$. Potom:

$$h_0(s) = \frac{2}{k^3}s^3 - \frac{3}{k^2}s^2 + 1, \qquad h_1(s) = -\frac{2}{k^3}s^3 + \frac{3}{k^2}s^2,$$

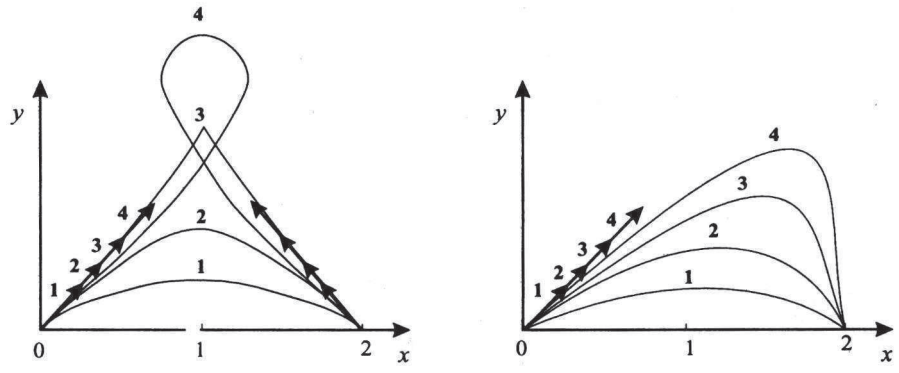
$$h_2(s) = \frac{1}{k^2}s^3 - \frac{2}{k}s^2 + s, \qquad h_3(s) = \frac{1}{k^2}s^3 - \frac{1}{k}s^2.$$

Pre uniformnú parametrizáciu (t.j. pre parameter krivky transformovaný do jednotkového intervalu, čiže pre $t \in \langle 0, 1\rangle$) používame miesto označených funkcií h_0, h_1, h_2, h_3 funkcie g_0, g_1, g_2, g_3. Pre tieto nové funkcie dostávame po dosadení do vyššie uvedených rovností a $k = t_{i+1}-t_i = 1$ nasledujúce vyjadrenia:

$$g_0(t) = 2t^3 - 3t^2 + 1, \qquad g_1(t) = -2t^3 + 3t^2,$$

$$g_2(t) = t^3 - 2t^2 + t, \qquad g_3(t) = t^3 - t^2.$$

Tieto krivky použil po prvýkrát v počítačovej grafike J. C. Ferguson pre konštruovanie kriviek v leteckom priemysle. Je treba zdôrazniť, že tvar Fergusonovej kubiky závisí od dĺžky dotykových vektorov. Na obr. 4.1 a) sú vykreslené štyri Fergusonove kubiky pre úmerne zväčšujúce sa dotykové vektory v obidvoch krajných bodoch. Ak meníme len dotykový vektor v začiatku, dochádza ku zmenám krivky podľa obr. 4.1 b). Podrobne o Fergusonových krivkách referuje tiež knižka [DRS_91].



a) b)

Obr. 4.1 Zmena dotykových vektorov Fergusonovej kubiky

Splajnové krivky

Splajnové krivky patria k často používaným interpolačným krivkám v technickej praxi. Sú matematickým modelom správania sa pružného latkového krivítka, ktoré v minulosti používali konštruktéri trupov lodí, keď z niekoľkých významných profilov

trupov lodi odvodzovali (interpolovali) ďalšie profily. Z hľadiska praxe majú zásadný význam **kubické splajnové krivky**.

Výklad preto obmedzíme len na túto skupinu splajnov. Najprv budeme definovať pojem kubickej splajnovej funkcie: Nech je daná postupnosť hodnôt $x_0 < x_1 < ... < x_n$ a im zodpovedajúce hodnoty $y_0, y_1, ..., y_n$, ktoré určujú body v rovine. Pod pojmom kubická splajnová funkcia $f(x)$ rozumieme zloženú funkciu, pre ktorú platí $f(x_i) = y_i$. Pritom funkcia f na každom z intervalov $\langle x_i, x_{i+1}\rangle$ je polynómom tretieho stupňa a má spojitú prvú a druhú deriváciu na celom intervale (x_0, x_n).

Zadanými bodmi (x_i, y_i) však nie je kubická splajnová funkcia určená jednoznačne, lebo pre n kubických polynómov majúcich $4n$ koeficientov, máme iba $4n$ - 2 podmienok na ich určenie. Presnejšie krajné body jednotlivých úsekov $2n$ podmienok, zo spojitosti prvej derivácie n-1 podmienok a zo spojitosti druhej derivácie n-1 podmienok. Je zrejmé, že na určenie kubickej splajnovej funkcie musíme zvoliť ešte ďalšie 2 podmienky. Spravidla sa zadávajú hodnoty prvej derivácie v začiatočnom a koncovom bode krivky alebo hodnoty druhej derivácie v týchto bodoch. Špeciálne sa často volí "podmienka voľného konca", t.j. nulová druhá derivácia v krajných bodoch.

Splajnovou krivkou pre dané určujúce body $P_0, P_1, ..., P_n$ a dané hodnoty parametra $t_0<t_1< ... < t_n$ rozumieme krivku $\boldsymbol{P} = \boldsymbol{P}(t)$, $t \in \langle t_0, t_n\rangle$, pre ktorú každá zo zložiek vektorovej funkcie $\boldsymbol{P}(t)$ je kubickou splajnovou funkciou. Zadanie hodnôt parametra pre určujúce body sa väčšinou realizujú istým algoritmom. Napríklad pre uniformnú parametrizáciu volíme $t_i = i$ a pre neuniformnú parametrizáciu postupne vzdialenosti krajných bodov oblúka.

Výpočet koeficientov kubickej splajnovej krivky uskutočníme tak, že z podmienok hladkosti a ďalších dodatočných podmienok určíme dotykové vektory krivky v určujúcich bodoch a potom jednotlivé oblúky splajnovej krivky vypočítame ako Fergusonove kubiky podľa predchádzajúcich vzťahov.

Označme $\boldsymbol{P}'_i$, i=0, ..., n hľadané dotykové vektory kubickej splajnovej krivky v uzlových bodoch. Vzhľadom na požiadavku spojitosti druhej derivácie vo vnútorných uzlových bodoch získame po úpravách vzťahy:

$$\frac{1}{k_i}P'_i + (\frac{2}{k_i} + \frac{2}{k_{i+1}})P'_{i+1} + \frac{1}{k_{i+1}}P'_{i+2} = \frac{3}{k_{i+1}^2}P_{i+2} + (\frac{3}{k_i^2} - \frac{3}{k_{i+1}^2})P_{i+1} - \frac{3}{k_i^2}P_i,$$

kde $i = 0, ..., n$-2.

Ďalšie dve rovnosti pre výpočet dotykových vektorov sa spravidla odvodia z niektorej z nasledujúcich volieb alebo podmienok :

1. Zvolíme $\boldsymbol{P}'_0$ a $\boldsymbol{P}'_n$ dotykové vektory v začiatočnom a koncovom bode krivky. Potom má uvedená sústava už len jediné riešenie.

2. Zadáme vektory $\boldsymbol{a}$ a $\boldsymbol{b}$ druhých derivácií v začiatočnom a koncovom bode krivky. V tomto prípade doplníme sústavu o rovnice

$$2P'_0 + P'_1 = \frac{3}{k_0}(P_1 - P_0) - \frac{k_0}{2}a,$$

$$P'_{n-1} + 2P'_n = \frac{3}{k_{n-1}}(P_n - P_{n-1}) - \frac{k_{n-1}}{2}b.$$

Prirodzený kubický splajn definujeme dodatočnou podmienkou $\boldsymbol{a} = \boldsymbol{b} = \boldsymbol{0}$.

3. Uzavretie krivky dopĺňa podmienky o ďalšie dve rovnice, ktoré získame použitím uvedenej formuly pre $i = n\text{-}1$ a $i = n$ s tým, že od indexov väčších než n odpočítame $n+1$.

Sústavu n rovníc o n neznámych riešime Gaussovou eliminániou. Matica sústavy je pre všetky zložky vektorov rovnaká, preto môžeme urobiť priamy chod Gaussovej eliminácie naraz pre rôzne pravé strany. Matica sústavy je diagonálne dominantná a pre prípad a) a b) je navyše táto matica trojdiagonálna.

Príklad 2. Určíme priestorový kubický splajn pre 4 body $P_0 = (0, 0, 0)$, $P_1 = (1, 0, 0)$, $P_2 = (1, 1, 1)$, $P_3 = (0, 0, 1)$. Uvažujeme uniformnú parametrizáciu a hľadáme prirodzenú splajnovu krivku.

Z predchádzajúcich vzťahov a podmienky $\boldsymbol{a} = \boldsymbol{b} = \mathbf{0}$ a $k_i = 1$ pre každé i, vyplývajú rovnice:

$$\begin{aligned} 2.\boldsymbol{P}_0 + \boldsymbol{P}_1 &= 3(P_1 - P_0) \\ \boldsymbol{P}_0 + 4.\boldsymbol{P}_1 + \boldsymbol{P}_2 &= 3(P_2 - P_0) \\ \boldsymbol{P}_1 + 4.\boldsymbol{P}_2 + \boldsymbol{P}_3 &= 3(P_3 - P_1) \\ \boldsymbol{P}_2 + 2.\boldsymbol{P}_3 &= 3(P_3 - P_2) \end{aligned}$$

a rozšírenú maticu pre Gaussovu elimináciu môžeme zapísať v tvare:

$$\left(\begin{array}{cccc|ccc} 2 & 1 & 0 & 0 & 3 & 0 & 0 \\ 1 & 4 & 1 & 0 & 3 & 3 & 3 \\ 0 & 1 & 4 & 1 & -3 & 0 & 3 \\ 0 & 0 & 1 & 2 & -3 & -3 & 0 \end{array}\right).$$

Riešením sústavy dostaneme dotykové vektory v určujúcich bodoch, napr. dotykový vektor $\boldsymbol{P}_3 = (-2.08, -2.78, -0.58)$. Jednotlivé oblúky dostaneme dosadením do Fergusonovej kubiky (1).

4.2.3 Aproximačné krivky

Pod aproximačnou krivkou rozumieme takú krivku, pre ktorú je daných niekoľko bodov charakterizujúcich jej tvar, ale nepožaduje sa, aby krivka nutne týmito bodmi prechádzala. Väčšinou sa požaduje, aby krivka minimalizovala nejakú podmienku (napr. súčet štvorcov vzdialeností od daných bodov). V počítačovej grafike použili ako prví pre konštruovanie kriviek a plôch riadiaci systém bodov P. E. Bézier a P. de Casteljau. Návrh krivky sa realizuje pomocou riadiaceho polygónu, t.j. lomenej čiary, ktorá je priradená krivke a určuje tvar tejto krivky. Zmenami polohy vrcholov riadiaceho polygónu meníme tvar krivky. Niektoré návrhové systémy umožňujú užívateľovi pomocou grafického vstupu interaktívne s počítačom navrhnúť príslušnú krivku, bez toho aby návrhár musel stanoviť jej rovnicu.

V tomto odstavci si všimneme aspoň základné vlastnosti Bézierových a B-splajnových kriviek. V obidvoch prípadoch budeme vychádzať z toho, že pre uvažovanú krivku je daný riadiaci polygón s vrcholmi $V_0, \ldots, V_n$.

Bézierove krivky

Teória Bézierových kriviek vznikla v rokoch 1959 - 1962 v súvislosti s uplatnením číslicovo riadených obrábacich strojov vo francúzskych automobilkách. Neskôr sa ukázalo, že za matematický základ tejto teórie môžeme zobrať tzv. Bernsteinove polynómy. Bézierova krivka je opísaná rovnicou

$$P(t) = \sum_{i=0}^{n} V_i B_{i,n}(t), \quad t \in \langle 0, 1 \rangle$$

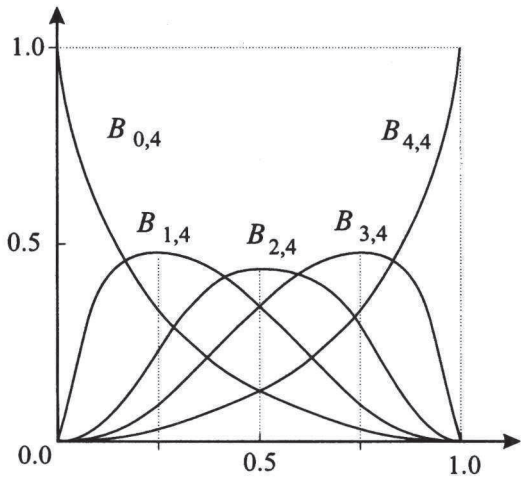
kde V_i sú polohové vektory vrcholov riadiaceho polygónu.

$B_{i,n}(t)$ sú Bernsteinove polynómy:

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i},$$

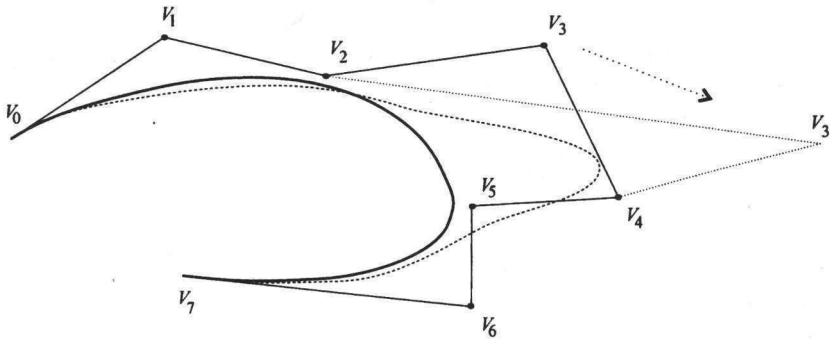
kde definitoricky položíme $\binom{0}{0} = 1$.

Obrázok 4.2 ilustruje systém Bernsteinových polynómov pre n = 4. Polynóm $B_{i,4}$ nadobúda maximálnu hodnotu na intervale $\langle 0, 1 \rangle$ v bode $1/4*i$, pre $i = 0, 1, 2, 3, 4$.



Obr. 4.2 Bernsteinove polynómy 4. stupňa

Na obr. 4.3 je pre daný riadiaci polygón V_0, V_1,..., V_7 vykreslená príslušná Bézierova krivka. Čiarkovane je vyznačený zmenený riadiaci polygón a k nemu príslušná modifikovaná Bézierova krivka.



Obr. 4.3 Zmena riadiaceho polygónu Bézierovej krivky

Sformulujeme niektoré vlastnosti Bézierových kriviek :

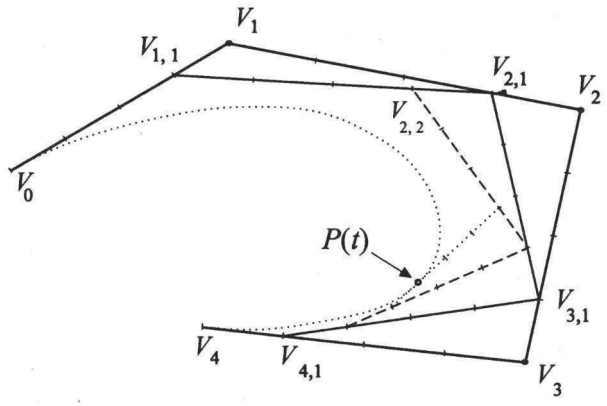
1. Začiatočným bodom Bézierovej krivky je bod V_0 a krivka sa v tomto bode dotýka priamky V_0V_1. Podobne koncovým bodom krivky je bod V_n a krivka sa v tomto bode dotýka priamky $V_{n-1}V_n$.

2. Bézierova krivka je krivkou stupňa n (pre daných n+1 bodov). Polynómi stupňa n, pre veľké n, sú numericky nestabilné a takisto aj Bezierová krivka stupňa n. Z toho dôvodu sa začali používať splajnové krivky nižšieho stupňa.

3. Na obr. 4.4 je ukázaná podstata tzv. algoritmu Casteljau. Určenie bodu krivky $\boldsymbol{P}(t)$ pre parameter t vykonáme postupným iteratívnym delením úsečiek riadiaceho polygónu pomerom závislým od parametru t. Na obrázku 4.4 sú znázornené tieto úsečky, kde pomer delenia je pre parameter $t = 0.75$ a $n = 4$. V každom i-tom priblížení definujeme body

$$V_{j,i} = (1-t).V_{j-1,i-1} + t.V_{j,i-1},$$

pre $i = 1, 2, 3, 4$ a $j = i, ..., 4$ a hľadaný bod je $\boldsymbol{P}(t) = V_{4,4}$.



Obr. 4.4 Algoritmus de Casteljau

B-splajnové krivky

Nevýhodou Bézierových kriviek je, že sa u nich so zvyšujúcim počtom vrcholov riadiaceho polygónu vo všeobecnosti zvyšuje i stupeň krivky. Táto nevýhoda sa síce dá obísť segmentáciou krivky (pozri príklad 2), ale tento postup je náročný na výpočet z dôvodu zložitého riešenia sústavy rovníc. Preto sa v geometrickom modelovaní začali používať v niektorých aplikáciách B-splajnové krivky. B-splajnová krivka stupňa k je určená riadiacim polygónom $V_0,..., V_n$ a hodnotami parametrov $t_0 \leq ... \leq t_m (n \leq m)$ (obvykle hovoríme o uzlovom vektore alebo parametrizačnom vektore). V hodnotách parametrov je zaistený aj "rozbeh a dobeh" výpočtov, t.j. zadanie podmienok v začiatočnom a koncovom bode krivky. B-splajnovú krivku stupňa k definujeme predpisom:

$$P(t) = \sum_{i=0}^{n} V_i . N_{i,k}(t).$$

Bázové funkcie $N_{i,k}(t)$ sú definované rekurentne:

$$\text{a) } N_{i,1}(t) = \begin{cases} 1 & \text{pre } t \in \langle t_i, t_{i+1} \rangle, \\ 0 & \text{inde}, \end{cases}$$

$$\text{b) } N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t), \text{ pre } k > 1.$$

Pre B-splajnovú krivku sa dá zmodifikovať algoritmus de Casteljau a je známy ako de Boorov algoritmus.

Môžeme definovať nielen racionálne Bézierove krivky ale aj racionálne B-splajnové krivky. Racionálna (neuniformná) B-splajnová krivka je okrem riadiaceho polygónu určená stupňom k, uzlovým vektorom a tzv. váhami jednotlivých vrcholov riadiaceho polygónu. Pre jednotkové váhy získame B-splajnovú krivku. Racionálne B-splajnové krivky a plochy sa využívajú pre univerzálnosť dátovej reprezentácie kriviek a plôch v návrhových systémoch CAD pre rôzne technické aplikácie.

Špeciálnym prípadom B-splajnovej krivky je Coonsov B-splajn, čo je uniformná kubická (k=3) B-splajnová krivka. Pre riadiaci polygón V_0, ..., V_n môžeme vyjadriť oblúk Coonsovej B-splajnovej krivky:

$$P_j(t) = \frac{1}{6} \sum_{i=0}^{3} V_{j+i} . C_i(t - j), \quad t \in \langle j, j+1 \rangle, \quad j = 0, ..., n-3,$$

kde

$$C_0(s) = -s^3 + 3.s^2 - 3.s + 1 = (1-s)^3, \qquad C_1(s) = 3.s^3 - 6.s^2 + 4, \qquad (2)$$
$$C_2(s) = -3.s^3 + 3.s^2 + 3.s + 1, \qquad C_3(s) = s^3.$$

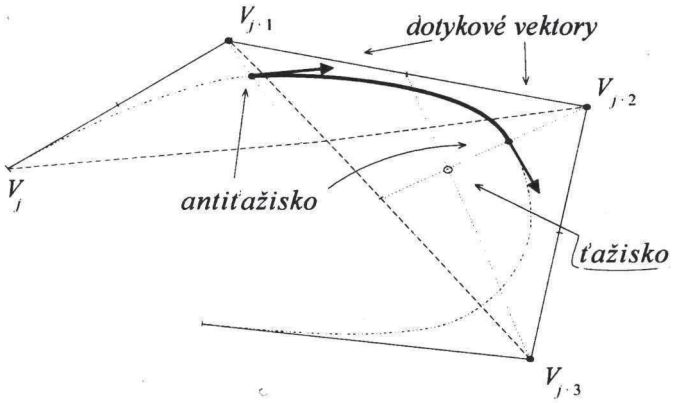
Určíme začiatočný a koncový bod oblúku Coonsovho B-splajnu. Pre funkcie C platí:

$$C_0(0) = 1,\ C_1(0) = 4,\ C_2(0) = 1 \text{ a } C_3(0) = 0.$$

Z toho dôvodu pre koncový bod j-teho segmentu platí:

$$P_j(j) = \frac{1}{6}(V_j + V_{j+1} + V_{j+2}) .$$

Vidíme, že tento bod leží na ťažnici trojuholníka $V_j V_{j\cdot 1} V_{j\cdot 2}$ v jednej tretine od vrchola $V_{j\cdot 1}$ (tento bod nazveme **antiťažisko**). Podobný vzťah platí aj pre koncový bod oblúka v trojuholníku $V_{j\cdot 1} V_{j\cdot 2} V_{j\cdot 3}$. Na obr. 4.5 je oblúk Coonsovho B-splajnu znázornený medzi týmito bodmi. Výhoda Coonsovho B-splajnu je v tom, že krivka je tretieho stupňa (t.j. stupeň žiadneho segmentu nie je väčší ako 3), a naviac je nezávislá na počte vrcholov riadiaceho polygónu. Zmena polohy vrcholu riadiaceho polygónu má vplyv len na lokálnu zmenu maximálne štyroch oblúkov krivky.



Obr. 4.5 Krajné body oblúka Coonsovho B-splajnu

Je pomerne jednoduché ukázať, že Coonsov B-splajn je kubickou uniformnou splajnovou krivkou pre spojovacie body v antiťažiskách jednotlivých trojuholníkov riadiaceho polygónu. Na to stačí vypočítať vektor prvej a druhej derivácie v krajných bodoch oblúka a ukázať, že tieto vektory majú spoločné oblúky (pozri obr. 4.5).

4.3 Geometrické modelovanie plôch

Podobne ako u kriviek uvedieme základné tvary určenia plochy. Pôjde o analytické vyjadrenie plôch, o interpolačné a aproximačné plochy. Pre interpolačné plochy uvedieme bilineárnu a bikubickú Coonsovu záplatu. Pre aproximačné Bézierovu a B-splajnovú plochu.

Základné informácie o splajnových plochách sú obsiahnuté v knižke [DRS_91], podrobnosti obsahujú špeciálne publikácie o splajnových funkciách a plochách [BART87].

4.3.1 Analytický popis plôch

Explicitnou rovnicou možno popísať plochy, ktoré sú grafmi funkcií dvoch premenných. Napr. rovnica $z = x.y$ popisuje tzv. hyperbolický paraboloid.

Implicitné rovnice nie sú vždy pre potreby počítačovej grafiky a geometrického modelovania vhodné. Príkladom implicitnej rovnice je rovnica

$$x^2 + y^2 + z^2 = r^2,$$

ktorá popisuje guľovú plochu s polomerom r a so stredom v začiatku sústavy súradníc. Pre naše potreby je vhodnejšie použitie parametrických funkcií. Je treba si uvedomiť, že tú istú plochu možno vyjadriť rôznymi sústavami parametrických funkcií.

Parametrické vyjadrenie plochy má tvar:

$$x = x(u, v),\ y = y(u, v),\ z = z(u, v),$$

kde parametre u a v sú z intervalov I_u a I_v. Z tohto parametrického zadania ľahko získame vektorovú reprezentáciu $\boldsymbol{r} = \boldsymbol{r}(u, v)$ prípadne aj bodové vyjadrenie:

$$P = P(u, v).$$

4.3.2 Interpolačné plochy

Interpolačné plochy sa objavujú v praxi všade tam, kde máme k dispozícii systém bodov alebo kriviek plochy a potrebujeme určiť ďalšie body alebo krivky plochy. Interpolačná plocha môže byť určená systémom bodov alebo jedným či dvomi systémami kriviek tejto plochy. Okrem toho možno na interpolačné plochy klásť doplňujúce požiadavky, ako je predpísaná hodnota vektorov parciálnych derivácií, a pod.

Pokiaľ je pre plochu známa množina bodov, predpokladá sa väčšinou, že tieto body ležia nad obdĺžnikovou sieťou (v rovine xy alebo aspoň v priestore parametrov). Jednou z metód interpolácie je v tomto prípade opäť použitie interpolácie po častiach s využitím splajnových funkcií. **Záplatou** spravidla rozumieme časť plochy ohraničenej 4 hraničnými krivkami. V prípade, že máme zadané len dve hraničné krivky $a_0(v)$ a $a_1(v)$, $v \in I_v$, môžeme zvyšné dve dourčiť úsečkami. Pokiaľ sú hodnoty parametra pre tieto krivky v rôznych intervaloch, prevedieme v prípade potreby afinnou transformáciou zmenu parametra jednej z kriviek tak, aby obidve krivky mali parameter z rovnakého intervalu. Potom najjednoduchší spôsob zadania záplaty je, ak spojíme úsečkou všetky zodpovedajúce body kriviek $a_0(v)$ a $a_1(v)$, $v \in I_v$.

Záplata je potom určená vzťahom:

$$P(u,v) = a_0(v) \cdot (1-u) + a_1(v) \cdot u \qquad v \in I_v,\ u \in \langle 0, 1 \rangle .$$

Je zrejmé, že systém viacerých kriviek $a_i(v)$ môžeme pomocou jednotlivých záplat pokryť a zložiť do výslednej plochy. Kvalita tejto interpolačnej plochy (môže ísť o povrch nástroja, formy, a pod.) je veľmi nízka. Záplaty sa pozdĺž spoločnej krivky nenapájajú hladko. Hovoríme, že nie je zaistené hladké napojenie susedných záplat. V technickej praxi sa s touto interpolačnou plochou stretávame napr. pri interpolácii plochy danej veľkým počtom kriviek $a_i(v)$. Výhodou je, že parametrické krivky pre $v = konšt.$ sú po častiach tvorené úsečkami, čo je vhodné pre prípravu NC programov pre frézovanie, a pod.

Ďalej sa budeme zaoberať prípadom interpolácie, kedy sú známe dva systémy parametrických kriviek plochy, ktoré vytvárajú krivky jednej aj druhej sústavy. Uvažujeme jednu záplatu takejto plochy a predpokladajme pre jednoduchosť, že táto záplata má parametre $u \in \langle 0, 1 \rangle$ a $v \in \langle 0, 1 \rangle$. Všeobecný prípad prevedieme na tento tvar vhodnými transformáciami parametra. Konštruujeme teda záplatu plochy určenej okrajom (štyrmi

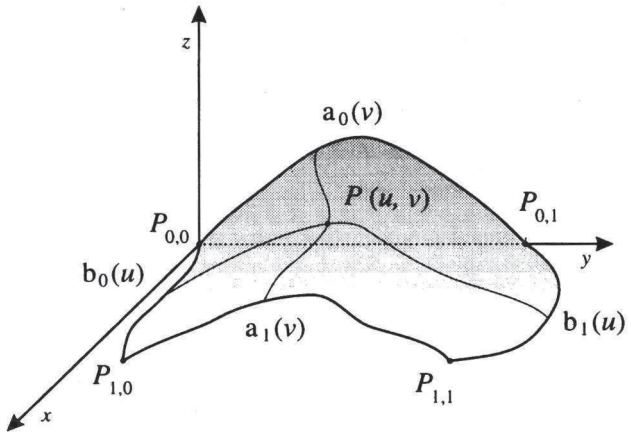
oblúkmi kriviek so spoločnými bodmi v rohoch) pre dvojice parametrov z jednotkového štvorca.

Protiľahlými stranami okraja záplaty nech sú krivky $a_0(v)$ a $a_1(v)$, resp.$b_0(u)$ a $b_1(u)$. Polohové vektory rohov záplaty označme $P_{0,0}$, $P_{0,1}$, $P_{1,0}$ a $P_{1,1}$ (pozri obr. 4.6).

Mapovacou maticou záplaty nazveme maticu

$$\mathbf{M} = \begin{pmatrix} P_{0,0} & a_0(v) & P_{0,1} \\ b_0(u) & P(u,v) & b_1(u) \\ P_{1,0} & a_1(v) & P_{1,1} \end{pmatrix},$$

kde $P(u, v)$ je polohový vektor všeobecného bodu záplaty.



Obr. 4.6 Okraje záplaty plochy

Bilineárna Coonsova záplata

Bilineárna Coonsova záplata je určená rovnicou:

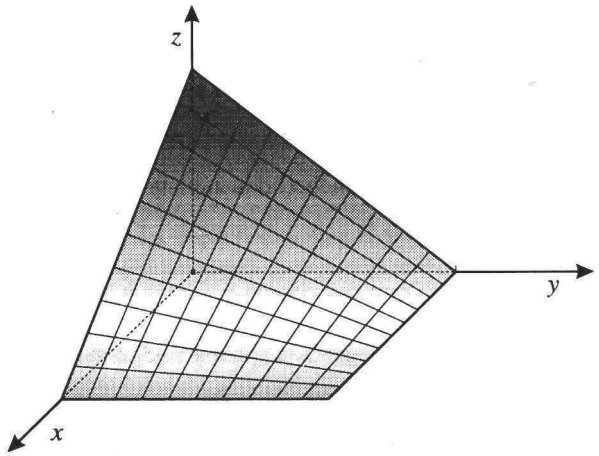
$$\begin{pmatrix} 1-u & -1 & u \end{pmatrix} \mathbf{M} \begin{pmatrix} 1-v \\ -1 \\ v \end{pmatrix} = 0,$$

kde lineárne funkcie 1- u, u a 1-v, v vystupujú v úlohe tzv. **blending funkcií** (*blending function*) a parametre u a v sú z jednotkového intervalu. Ak vytvárame záplatu nad jednotkovým štvorcom v rovine xy, môžeme vektory v mapovacej matici nahradiť iba ich treťou zložkou a miesto parametra u a v použiť priamo premenné x a y. Ľahko možeme dokázať, že ak sú protiľahlé dve strany okraja lineárnej záplaty úsečkami, je výsledná plocha priamkovou plochou, t.j. napríklad, ak

$$b_0(u) = (1-u)\cdot P_{0,0} + u\cdot P_{1,0}, \qquad b_1(u) = (1-u)\cdot P_{0,1} + u\cdot P_{1,1},$$

potom parametrické krivky záplaty $P(u,v) = b_0(v) \cdot (1-u) + b_1(v) \cdot u$ pre v = *konšt.* sú taktiež úsečkami (pozri obr. 4.7). Rovnica záplaty je v tomto prípade tvaru:

$$P(u,v) = b_0(v) \cdot (1-u) + b_1(v) \cdot u$$



Obr. 4.7 Bilineárna Coonsova záplata

Pomocou bilineárnych Coonsových záplat nie je zaistené hladké spojenie (plátovanie). Použitie týchto záplat je teda hlavne pri návrhovaní jednotlivých plôch a uplatnenie v technickej praxi majú hlavne tam, kde je žiadúce, aby pri interpolácii (pokiaľ je to možné) vznikali priamkové plochy.

Bikubická Coonsova záplata

Coons použil prvý raz záplatu s kubickými tvarovacími funkciami v leteckom priemysle pri návrhu trupu lietadla v roku 1967. Bikubická záplata sa od bilineárnej líši len spôsobom tvarovania, t.j. miesto funkcií (1-u) a u, resp. (1-v) a v sú použité kubické funkcie. Z podmienky, aby plocha obsahovala okraj záplaty, sa dá ukázať, že za tvarovacie funkcie možno zobrať funkcie F_0 a F_1, ktoré sme odvodili pre krivky.

Rovnicu zovšeobecnenej bilineárnej záplaty môžeme zapísať v tvare:

$$\begin{pmatrix} F_0(u) & -1 & F_1(u) \end{pmatrix} \mathbf{M} \begin{pmatrix} F_0(v) \\ -1 \\ F_1(v) \end{pmatrix} = 0 ,$$

kde $F_0(s) = 2.s^3 - 3.s^2 + 1$, $F_1(s) = -2.s^3 + 3.s^2$ sú dva zo 4 Hermitových polynómov tretieho stupňa.

Podstatnou črtou bikubických Coonsových záplat je, že zaisťujú napojenie záplat a za určitých podmienok plátovanie. Ak okraje obidvoch záplat v bodoch $P_{0,1}$ a $P_{1,1}$ majú spoločnú dotyčnicu, potom obidve záplaty majú v bodoch spoločnej krivky $b_1(u)$

spoločnú dotykovú rovinu. Pokiaľ teda je plocha daná dvomi systémami hladkých kriviek, potom pomocou bikubických Coonsových záplat je automaticky zaistené, že výsledná plocha je hladká.

Ďalšie typy Coonsových záplat

Technická prax si postupne vyžadovala ďalší rozvoj Coonsovej teórie interpolačných plôch. V niektorých aplikáciách nie sú k dispozícii okrajové krivky záplat, ale sú známe okrem rohov záplat dotykové vektory okrajových kriviek záplat v týchto bodoch. Okrajové krivky potom možeme získať ako Fergusonove kubiky a takto určenú bikubickú záplatu tiež nazývame **Fergusonovou záplatou**. Je určená dvanástimi vektormi, preto sa napr. v [DRS_91] označuje ako *dvanásťvektorová záplata*. Všeobecnejšou variantou je tzv. *šestnásťvektorová záplata*, ktorá je oproti dvanásťvektorovej záplate naviac určená vektormi druhých zmiešaných parciálnych derivácií v rohoch záplaty. To znamená, že nedajú sa voliť tieto vektory ľubovoľne. Tieto typy Coonsových plôch sa používajú napr. v systéme F-MILL, resp. INCAD 3D pre prípravu NC programov obrábacích strojov.

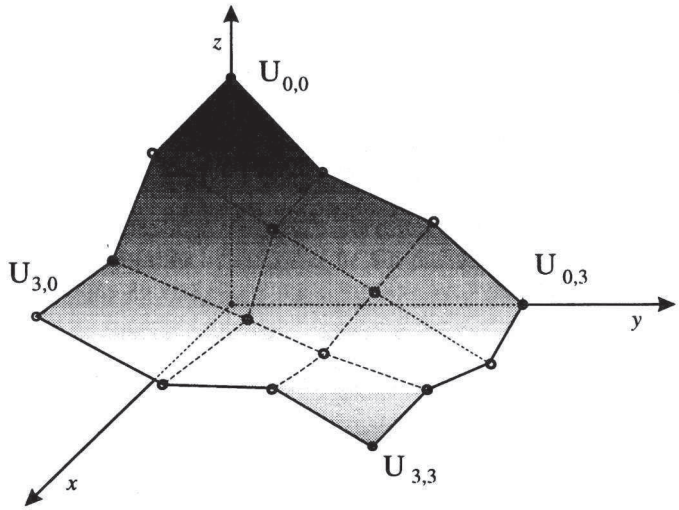
Problémy sa objavujú, ak chceme Coonsovu plochu hladko napojiť na plochu, ktorá je daná analyticky alebo vznikla iným interpolačným postupom. Na to vyvinuli teóriu všeobecných Coonsových plôch, pre ktoré je možné zadať priebeh vektora "priečnej" derivácie pozdĺž okrajovej krivky. Ale celkovo možno povedať, že vyššie typy Coonsových plôch vyžadujú veľmi kvalifikovaného návrhára. Vývoj teórie geometrického modelovania tvarovo zložitých objektov smeroval preto k uplatneniu aproximačných plôch, hlavne B-splajnových plôch.

4.3.3 Aproximačné plochy

Teória aproximačných plôch v geometrickom modelovaní je založená na podobných základoch ako teória aproximačných kriviek. Tu sa zmienime o Bézierových a B-splajnových plochách.

Riadiaca sieť aproximačnej plochy je určená maticou polohových vektorov **bodov siete** (nazývame ich tiež uzlami siete). **Sieťou** U budeme v ďalšom výklade rozumieť sieť $(n+1)\times(m+1)$ bodov

$$U = \begin{pmatrix} U_{0,0} & U_{0,1} & \dots & U_{0,m} \\ U_{1,0} & U_{1,1} & \dots & U_{1,m} \\ \dots & \dots & \dots & \dots \\ U_{n,0} & U_{n,1} & \dots & U_{n,m} \end{pmatrix}$$



Obr. 4.8 Riadiaca sieť

Na obr. 4.8 sú vyznačené väčšími krúžkami uzlové body siete a silnejšou čiarou rohové strany siete.

Bézierova plocha

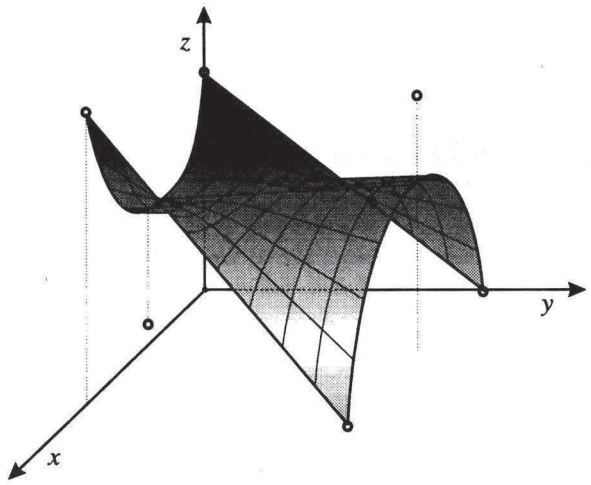
V roku 1970 sa začal vo francúzskej automobilke Renault používať systém UNISURF, ktorý na modelovanie tvaru karosérie využil plochy určené riadiacou sieťou podľa návrhu P. Béziera. Bézierova plocha pre sieť **U** je určená parametrickým vyjadrením:

$$P(u,w) = B_n(u)\,\mathbf{U}\,B_m^T(w), \quad u \in \langle 0,1\rangle, w \in \langle 0,1\rangle, \text{ kde}$$

$$B_k(s) = [B_{0,k}(s), B_{1,k}(s), ..., B_{k,k}(s)],$$

t.j. B_k je vektorová funkcia parametru s, ktorej zložkami sú hodnoty jednotlivých Bernsteinových polynómov stupňa k v bode s. Bézierova plocha prechádza rohovými bodmi siete a okrajové krivky plochy sú Bézierovými krivkami určenými okrajmi siete (pozri obr. 4.9).

Podobne ako pre Bézierove krivky i pre Bézierove plochy sa dá upraviť a formulovať algoritmus de Casteljau pre konštrukciu bodu plochy, ktorý zodpovedá daným hodnotám parametrov u_0 a v_0. Algoritmus de Casteljau aplikujeme najprv na stĺpce matice **U**, čím vznikne polygón ${}^nU_0,...,{}^nU_m$, na ktorý potom opäť aplikujeme algoritmus de Casteljau. Výsledkom je určenie bodu s polohovým vektorom P (u_o, v_o). Dá sa dokázať, že je možné v matici **U** použiť algoritmus de Casteljau na riadky a po vykonaní tohto algoritmu na získané body prídeme k rovnakému konečnému výsledku.



Obr. 4.9 Bézierova plocha

V odstavci venovanom Bézierovým krivkám sme sa zmienili o vytváraní krivky po častiach a o racionálnych Bézierových krivkách. Všetky tieto postupy sú používané i pre Bézierove plochy.

B - splajnová plocha

Podobne ako sme definovali Bézierovu plochu možno zaviesť i B-splajnovú plochu. V tomto prípade je možné okrem riadiacej siete voliť ešte stupeň bázových polynómov v obidvoch parametroch u a v. Značnú tvarovú variabilitu a všeobecnosť potom ponúkajú hlavne racionálne B-splajnové plochy.

Podrobnejšie si teraz všimnime len špeciálne B-splajnové plochy - Coonsove B-splajnové plochy. Plocha pre danú sieť **U** je určovaná po častiach týmto predpisom

$$^{i,j}\mathrm{P}(u,v) = \tfrac{1}{36}\mathrm{C}(u-i)\,{}^{i,j}\mathbf{U}\,\mathrm{C}^{T}(v-j),$$

kde $u \in \langle i, i+1\rangle, v \in \langle j, j+1\rangle$, $\mathrm{C}(s) = [C_0(s), C_1(s), C_2(s), C_3(s)]$ je vektorová funkcia, ktorej zložky sú dané predpisom (2) a ${}^{i,j}\mathbf{U}$ je podmatica 4×4 v matici **U**, t.j.

$$\mathbf{U} = \begin{pmatrix} \mathrm{U}_{i,j} & \mathrm{U}_{i,j+1} & \mathrm{U}_{i,j+2} & \mathrm{U}_{i,j+3} \\ \mathrm{U}_{i+1,j} & \mathrm{U}_{i+1,j+1} & \mathrm{U}_{i+1,j+2} & \mathrm{U}_{i+1,j+3} \\ \mathrm{U}_{i+2,j} & \mathrm{U}_{i+2,j+1} & \mathrm{U}_{i+2,j+2} & \mathrm{U}_{i+2,j+3} \\ \mathrm{U}_{i+3,j} & \mathrm{U}_{i+3,j+1} & \mathrm{U}_{i+3,j+2} & \mathrm{U}_{i+3,j+3} \end{pmatrix}$$

Pre záplatu Coonsovej B-splajnovej plochy určíme jej okrajové krivky. Napr. pre $u = i$ stanovíme, že okrajová krivka je oblúkom B-splajnovej krivky pre riadiaci polygón s vrcholmi Q_k, $k = j,\ldots,j+3$ v antiťažiskách trojuholníka $U_{i,j}U_{i+1,j}U_{i+2,j}$. Pre Coonsov B-splajn nemôžeme tak jednoducho ako pre Bézierove plochy použiť parametrizáciu na intervale $\langle 0,1\rangle \times \langle 0,1\rangle$.

5

Rasterizácia kriviek a oblastí

5.1 Úvod

Vývoj rastrovej grafiky je podmienený rýchlym vývojom mikroelektroniky, najmä mikroprocesorov a pamätí. Historicky prvé vektorové obrazovky boli nahradené rastrovými obrazovkami. Tieto vytvárajú obraz z bodov rastra, a preto vzrástol význam rastrového zobrazovania. V tejto časti popíšeme *algoritmy rastrovej grafiky*. Prevod základných grafických výstupných prvkov (úsečky, kružnice, krivky, oblasti a textového reťazca) do postupnosti obrazových bodov nazývame **rasterizáciou**.

Funkcie a procedúry na rasterizáciu sa v grafických programoch vyvolávajú veľmi často (napríklad pri každom novom generovaní obrazu), a preto musia byť rýchle. Okrem toho grafický výstup by mal spĺňať prirodzenú požiadavku vizuálnej kvality, čo je často v protiklade s rýchlosťou zobrazenia. Celkove pri výbere algoritmov pre grafický systém sa rozhodujeme podľa typu aplikácie a hardveru. Už pri strednej rozlišovacej schopnosti (t.j. okolo 512 obrazových bodov v jednom smere) dávame prednosť rýchlym algoritmom. Zrýchlenie môžeme dosiahnuť použitím špeciálnych grafických procesorov, ktoré súčasne s behom aplikačného programu transformujú výstupné grafické prvky do rastrovej formy.

5.2 Rastrový rozklad úsečky

Uvedieme dva algoritmy generovania úsečiek do rastrovej formy. Pri rozklade úsečky budeme predpokladať, že oba koncové body majú celočíselné súradnice ($x1$, $y1$), ($x2$, $y2$).

5.2.1 Jednoduchý prírastkový algoritmus

Analytické vyjadrenie priamky, ktorá nie je rovnobežná s osou y, vyjadrujeme v tvare:

$$y = m.x + b,$$

kde m je smernica priamky a b posun na osi y. Koncové body úsečky určujú priamku s parametrami m a b:

$$m = \frac{y2-y1}{x2-x1} \qquad \text{a} \qquad b = \frac{x2 \cdot y1 - x1 \cdot y2}{x2-x1} .$$

Aby sme pre výpočet súradnice y zadanej úsečky nemuseli používať analytické vyjadrenie priamky, budeme uvažovať pravidelný prírastok dx pre súradnicu x. Takto súradnice nasledujúceho bodu úsečky získame priamo zo súradníc predchádzajúceho bodu. Špeciálne pre prírastok $dx = 1$ (prírastok 1 volíme preto, aby sme mali v každom stĺpci obrazovky zobrazený bod) dostaneme pre súradnicu y prírastok m:

$$x' = x + dx = x + 1 ,$$

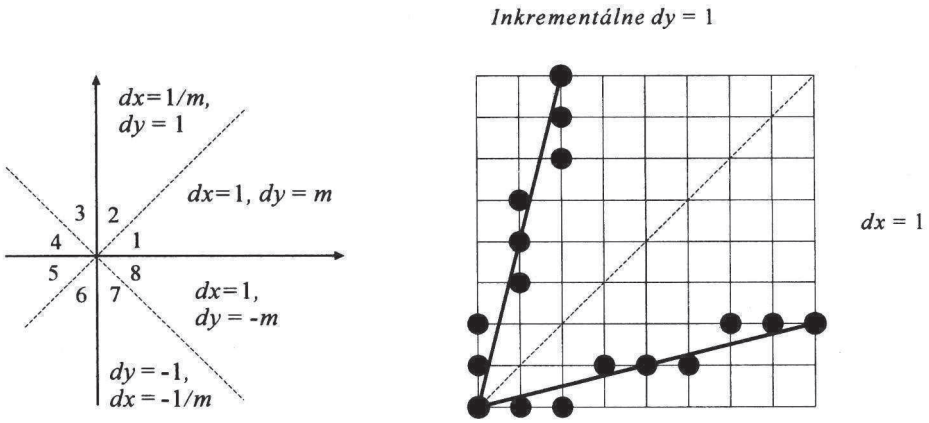
$$y' = m.x' + b = m.x + b + m = y + m.$$

Ak inkrementujeme súradnicu o $dy = 1$, dostaneme pre súradnicu x prírastok $1/m$:

$$x' = 1/m.y' - 1/m.b = 1/m.y + 1/m - 1/m.b = x + 1/m.$$

V prírastkovom algoritme, známom ako DDA *(Digital Differential Analyzer)* , nepoužívame operáciu násobenia. Súradnicu y musíme počítať v reálnej aritmetike a zaokrúhliť ju do celočíselných súradníc rastrových bodov.

Pre smernicu $m > 1$ je prírastok súradnice y väčší ako prírastok x, a preto vykresľovaná úsečka má väčšie medzery medzi jednotlivými vykresľovanými bodmi. Preto treba zameniť úlohu súradníc x a y tak, aby sa jednotkový prírastok bral pre súradnicu y a x budeme inkrementovať o $1/m < 1$. Na obrázku 5.1 znázorňujeme rovinu s oktantami a zodpovedajúcimi prírastkami.



Obr. 5.1 Prírastky pre niektoré oktanty a dve úsečky v 1. a 2. oktante

Pretože samotný algoritmus je podobný pre jednotlivé oktanty, uvedieme nasledujúci program, ktorý realizuje prírastkový algoritmus v 1. oktante. Označíme začiatočný bod $P = (x1, y1)$ a koncový bod $Q = (x2, y2)$. Procedúra *swap* vymieňa súradnice bodov úsečky P a Q. Používame procedúru *write_pixel*, ktorá zapíše hodnotu *colour* do pamäti obrazu podľa súradníc x a y. Funkcia *round* zaokrúhli súradnicu do rastra na celočíselné hodnoty. Funkcia *error* hlási chybu.

Prírastkový algoritmus úsečky DDA (Digital Differential Analyzer)

```
procedure DDA_line ( x1, y1,                    { začiatočný bod P }
                     x2, y2,                    { koncový bod Q }
                     colour :integer );         { farba vykreslenej úsečky }

var dx, dy, x, y, m: real;
begin
   if x1 > x2 then swap;                        { zameň body, aby prvý bod bol ľavý }
   if x1 < x2 then
      begin
      dx:= x2 - x1;
      dy:= y2 - y1;
      m:= dy/dx;
      y:=y1;
      for x:= x1 to x2 do
         begin
         write_pixel (x, round(y), colour);     { zobraz bod (x, y) }
         y:= y + m                              { inkrementuj súradnicu y }
         end
      end
   else if y1 = y2 then write_pixel (x1, y1, colour)   { zapíš len jeden bod }
   else error ;
end.
```

Nasledujúca procedúra *line* ukazuje postup generovania úsečky pre všetky oktanty vhodnou zámenou súradníc a voľbou príslušných prírastkov. V časti generovanie pre 1. oktant zavoláme napríklad funkciu *DDA_Line1*, ktorú sme uviedli. Podobne by sme odvodili *DDA_Line2* aj pre 2. oktant. Taktiež môžeme vybrať aj inú funkciu, ktorá bude generovať body úsečky v príslušnom oktante.

Prírastkový algoritmus úsečky pre všetky oktanty

```
Procedure line ( x1, y1,                        { začiatočný bod }
                 x2, y2,                        { koncový bod }
                 colour : integer );            { farba vykreslenej úsečky }

var dx, dy, x, y, m: real;
begin
   dx:= x2 - x1;
   dy:= y2 - y1;
   if abs(dx) > abs(dy) then
      begin                                     { oktanty 1, 4, 5, 8 }
      if x1 > x2 then  swap;                    { zameň body, aby prvý bod bol vľavo }
      if y1 < y2 then  Yinc:= 1;
                  else  Yinc:= -1;
         begin                                  { generovanie úsečky pre 1. oktant }
         DDA_Line1(x1, y1, x2, y2, colour);
         end;
      end;
   else begin                                   { abs(dx)<abs(dy), t.j. oktanty 2, 3, 6, 7 }
```

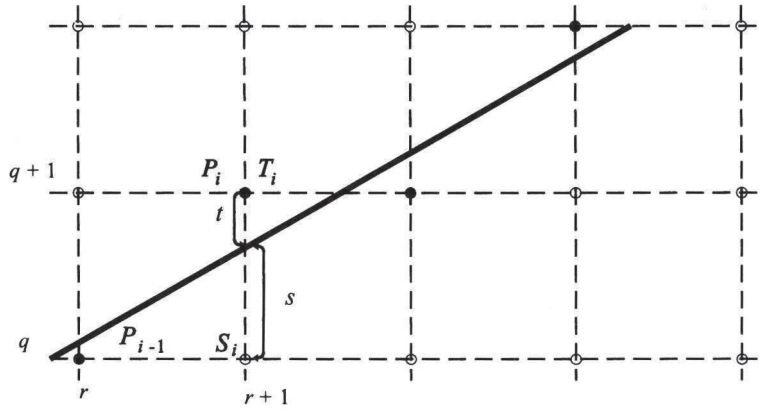
```
    if y1 > y2 then swap;                    { zameň body, aby prvý bod bol vľavo }
    if x1 < x2 then Xinc:= 1;
               else Xinc:= -1;
        begin                                { generovanie pre 2. oktant }
        DDA_Line2(x1, y1, x2, y2, colour);
        end;
end.
```

5.2.2 Bresenhamov algoritmus pre úsečku

V algoritme DDA sa počíta súradnica y (resp. x) v reálnej aritmetike a musíme ju zaokrúhliť. **Bresenhamov algoritmus** postupuje tak, aby sa práve operácie v pohyblivej rádovej čiarke nepoužívali.

Predpokladajme, že smernica m je z intervalu $\langle 0, 1 \rangle$, čiže sklon úsečky od osi x je v rozpätí od 0 do 45 stupňov. V algoritme postupujeme od bodu $P(i\text{-}1)$ k bodu $P(i)$ s prírastkom v smere osi x ($dx = 1$). Úsečka pretína mriežku medzi dvoma susednými bodmi vo vertikálnom smere. Označme horný bod ako $T(i)$ (horný úsek t) a dolný bod $S(i)$ (dolný úsek s) podľa obr. 5.2.

Zavedieme riadiacu premennú d úmernú rozdielu úsekov (s-t). Ak je $s < t$, tak bod $S(i)$ je bližšie k priamke a vyberieme ho ako nasledujúci bod úsečky. Inak ($s \geq t$) je bližšie bod T(i). Medzi posunom horizontálne alebo diagonálne rozhoduje znamienko d.



Obr. 5.2 Rozklad úsečky do rastra

Odvodíme algoritmus. Nech $x1 < x2$, potom pri rozklade úsečky budeme postupovať z bodu ($x1$, $y1$) do bodu ($x2$, $y2$). Najprv posunieme prvý bod do začiatku sústavy súradníc. Vtedy začiatočný bod má súradnice (0, 0) a koncový bod (dx, dy), kde $dx = x2 - x1$ a $dy = y2 - y1$. Rovnica priamky má v tom prípade tvar $y = (dy/dx).x$. Označíme si súradnice bodov po posunutí podľa obrázku 5.2:

$$P(i\text{-}1) = (x_{i-1}, y_{i-1}) = (r, q), \qquad S(i) = (r+1, q), \qquad T(i) = (r+1, q+1).$$

Nasledujúci bod $P(i)$ vyberáme medzi bodom $S(i)$ a bodom $T(i)$, t.j. z bodu $P(i\text{-}1)$ horizontálne alebo diagonálne. Ak je bližšie k úsečke bod $T(i)$, potom zvolíme posun diagonálne, inak horizontálne. Rozdiel $(s\text{-}t)$ rozhoduje o výbere bodu takto :

1. pre $s\text{-}t < 0$, vyberieme nasledujúci bod $S(i)$;

2. pre $s\text{-}t \geq 0$, vyberieme nasledujúci bod $T(i)$.

Pre jednotlivé vzdialenosti platí:

$$s = (dy/dx).(r+1) - q \ , \quad t = q+1 - (dy/dx).(r+1),$$

$$s - t = 2.(dy/dx).\ (r+1) - 2.q - 1.$$

Po odstránení operácie delenia dostaneme:

$$dx.(s\text{-}t) = 2.(r.dy\text{-}q.dx) + 2.dy - dx.$$

Hodnota dx je kladná, lebo $x_1 < x_2$, a preto o výbere môže rozhodovať aj posledne uvedený výraz. Označíme si ľavú stranu $dx.(s\text{-}t)$ ako riadiacu premennú $d(i)$ a dosadíme pre $r = x_{i-1}$ a $q = y_{i-1}$:

$$d(i) = 2.x_{i-1}\ .dy - 2.y_{i-1}\ .dx + 2.dy - dx. \qquad (1)$$

Ak zvýšime index o 1, dostávame pre riadiacu premennú nasledujúci tvar:

$$d(i+1) = 2.x_i\ .dy - 2.y_i\ .dx + 2.dy - dx.$$

Odčítaním posledných dvoch rovníc dostaneme:

$$d(i+1) - d(i) = 2.dy.(x_i - x_{i-1}) - 2.dx.(y_i - y_{i-1}).$$

Predpokladáme, že používame prírastkový krok 1, t.j. $x_i - x_{i-1} = 1$, teda posledný rozdiel môžeme upraviť na tvar:

$$d(i+1) = d(i) + 2.dy - 2.dx.(y_i - y_{i-1}).$$

Ak $d(i) < 0$, tak vyberieme bod $S(i)$. V tom prípade $y_i = y_{i-1}$ a

$$d(i+1) = d(i) + 2.dy\ . \qquad (2)$$

Ak $d(\mathrm{i}) \geq 0$, tak vyberieme bod $T(i)$. V tomto prípade $y_i = y_{i-1} + 1$ a

$$d(i+1) = d(i) + 2.(dy - dx). \qquad (3)$$

Tým sme popísali iteratívny výpočet $d(i+1)$ z predchádzajúcej hodnoty $d(i)$, čiže výber bodu $P(i)$. Začiatočnú hodnotu dostaneme pre $i=1$ dosadením do (1):

$$d(1) = 2.dy - dx. \qquad (4)$$

Pre výpočet uvedených hodnôt (2)-(4) treba minimálny počet aritmetických operácii: súčet, rozdiel a násobenie dvoma. Ostatné prípady dostaneme analogicky buď zámenou bodov alebo výmenou súradníc. Pre zaujímavosť uvedieme vzorce riadiacej premennej $d(i)$ pre m z intervalu $\langle -1, 0 \rangle$.

Ak $d(i) < 0$, tak vyberáme bod $S(i)$. V tom prípade $y_i = y_{i-1}$ a $d(i+1) = d(i) - 2.dy$.

Ak $d(i) \geq 0$, tak vyberáme bod $T(i)$. Platí $y_i = y_{i-1} + 1$ a $d(i+1) = d(i) - 2.(dy + dx)$.

Nasledujúci program realizuje zjednodušený Bresenhamov algoritmus v 1. oktante, t.j. pre smernicu z intervalu $\langle 0, 1 \rangle$. Označíme začiatočný bod $P=(x1, y1)$ a koncový bod $Q=(x2, y2)$. Procedúra *write_pixel* zabezpečuje zápis hodnoty *colour* do pamäti obrazu podľa súradníc x a y.

Bresenhamov algoritmus úsečky

```
procedure Bres_line ( x1, y1,                    { začiatočný bod }
                      x2, y2,                    { koncový bod }
                      colour : integer );        { farba vykreslenej úsečky }

var dx, dy, d, incr1, incr2, x, y, m  : integer;
begin
   dx:= abs(x2 - x1);   dy:= abs(y2 - y1);
   d:= 2*dy - dx;                                { začiatočná hodnota d }
   incr1:= 2*dy;                                 { prírastok pre d < 0 horizontálne }
   incr2:= 2*(dy-dx);                            { prírastok pre d ≥ 0 diagonálne }
    if x1 > x2 then begin x:= x2 ; y:= y2 ; xend:= x1 end
               else begin x:= x1 ; y:= y1 ; xend:= x2 end;
   write_pixel (x, y, colour);                   { vykresli 1. bod }
   while x < xend do
                  begin
                  x:= x + 1 ;
                  if d < 0 then d:= d + incr1    { vyber bod S(i), horizontálne }
                           else begin
                                y:= y + 1 ;
                                d:= d + incr2;   { vyber bod T(i), diagonálne }
                                end
                  write_pixel (x, y, colour);   { vykresli bod }
                  end
end.
```

5.3 Rastrový rozklad kružnice

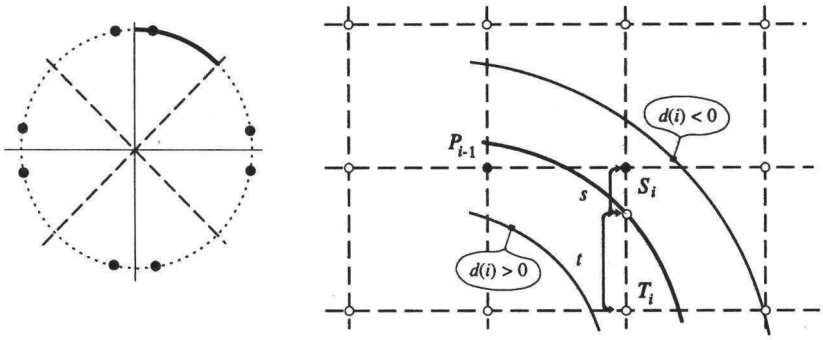
Uvedieme zovšeobecnenie predchádzajúceho postupu na rozklad kružnice. Rozklad kružnice zrýchlime, ak pre kružnicu so stredom v začiatku sústavy súradníc využijeme symetrie. Ak bod (x, y) je bodom kružnice, potom na kružnici leží 7 ďalších bodov:

$$(y, x), (y, -x), (x, -y), (-x, -y), (-y, -x), (-y, x), (-x, y).$$

Preto stačí spočítať len 1/8 kružnice, napríklad oblúk od 45 do 90 stupňov (t.j. pre hodnoty x od 0 až po $r/\sqrt{2}$). Pretože dotyčnica ku kružnici vo vyšetrovaných bodoch zviera s osou x uhol od 0 po -45 stupňov, budeme postupovať v kladnom horizontálnom smere alebo v diagonálnom smere dole (podobne ako pre úsečku so smernicou od -1 po 0 t.j. v 8. oktante, obr. 5.3 a)).

V každom kroku algoritmu vyberáme rastrový bod $P(i)$ najbližšie ku kružnici. Pre určenie odchýlky môžeme použiť kritérium:

$$d(P(i)) = (x_i^2 + y_i^2) - r^2 \ .$$



a) b)

Obr. 5.3 Symetria kružnice a rozklad kružnice do rastra podľa hodnôt s a t

Základná myšlienka spočíva vo výbere najbližšieho bodu pomocou riadiacej premennej (obdobne ako u Bresenhamovho algoritmu pre úsečku). Pri výbere riadiacej premennej sa snažíme minimalizovať chybu odchýlky. Označme dva rastrové body, ktoré prichádzajú do úvahy:

$S(i) = (x_{i-1} + 1, y_{i-1})$ t.j. horizontálny posun (doprava),

$T(i) = (x_{i-1} + 1, y_{i-1} -1)$ t.j. diagonálny posun (doprava dole).

Nech riadiaca premenná je definovaná vzťahom:

$$d(i) = d(S(i)) + d(T(i)) \ .$$

Analyzujme kritérium pre výber bodu. Ak kružnica prechádza nad obidvoma bodmi $S(i)$ a $T(i)$ (t.j. body sú vo vnútri kružnice), vidíme, že hodnota $d(i) < 0$. Naopak, ak body $S(i)$ a $T(i)$ ležia zvonku kružnice, potom hodnota $d(i) > 0$. Na obrázku 5.3 b) vidíme (kružnicový oblúk s menším polomerom) aj zaujímavý prípad: bod $S(i)$ je zvonku kružnice ($d(S(i)) > 0$) a bod $T(i)$ je vo vnútri kružnice ($d(T(i)) < 0$). V tomto prípade vyberieme bod podľa znamienka hodnoty $d(i)$. To znamená, že rozhodne, ktorá hodnota v absolútnej hodnote je väčšia, a to priamo súvisí s dĺžkami s a t na obr. 5.3. Nakoniec môžeme zhrnúť kritérium pre výber bodu a zmenu riadiacej premennej takto:

1. Ak $d(i) \geq 0$, tak vyberáme bod $T(i)$. Pretože sa posúvame v diagonálnom smere dole, dostávame sa výberom bodu skôr do vnútra kruhu a tým sa hodnota riadiacej premennej zmenšuje smerom do záporných hodnôt.

2. Ak $d(i) < 0$, tak vyberáme bod $S(i)$. Pretože sa posúvame v horizontálnom smere, dostávame sa výberom bodu skôr von z kruhu a tým sa hodnota riadiacej premennej zväčšuje smerom do kladných hodnôt.

Po rozpísaní riadiacej premennej a zavedení substitúcie pre ďalší iteračný krok dostávame :

$$d(i) = (x_{i-1} + 1)^2 + y_{i-1}^2 - r^2 + (x_{i-1} + 1)^2 + (y_{i-1} - 1)^2 - r^2$$

$$= 2.x_{i-1}^2 + 4.x_{i-1} + 2 + 2.y_{i-1}^2 - 2.y_{i-1} + 1 - 2.r^2$$

$$d(i+1) = 2.x_i^2 + 4.x_i + 2 + 2.y_i^2 - 2.y_i + 1 - 2.r^2$$

Po odčítaní posledných dvoch rovníc $d(i+1) - d(i)$ máme :

$$d(i+1) - d(i) = 2.(x_i^2 - x_{i-1}^2) + 4.(x_i - x_{i-1})$$

$$+ 2.(y_i^2 - y_{i-1}^2) - 2.(y_i - y_{i-1}) .$$

Po zvážení každej z dvoch možností dostávame pre premennú $d(i)$:

1. Ak $d(i) \geq 0$, tak $x_i = x_{i-1} + 1$ a $y_i = y_{i-1} - 1$ a upravíme $d(i+1) = d(i) + 4.x_{i-1} - 4.y_{i-1} + 10$.

2. Ak $d(i) < 0$, tak $x_i = x_{i-1} + 1$ a $y_i = y_{i-1}$ a upravíme $d(i+1) = d(i) + 4.x_{i-1} + 6$.

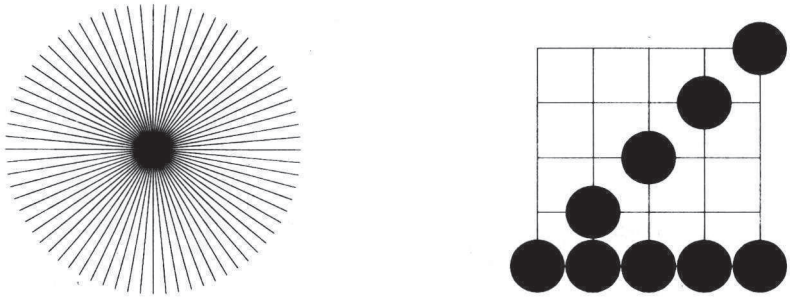
Hodnotu $d(1)$ dostávame dosadením bodu $(0, r)$ do základného vzťahu $d(1) = 3 - 2.r$.

Uvedené posledné tri vzťahy tvoria základ algoritmu, ktorý navrhol J. Mischener. Podobne možno postupovať aj pre elipsu, kde prírastky tiež už nebudú konštantné ako pri úsečke. Pri rozklade elipsy treba vypočítať namiesto oktantu kvadrant, lebo elipsa už nie je 8-symetrická. Štruktúra algoritmu na rozklad úsečky, kružnice a elipsy je však veľmi príbuzná.

5.4 Vyhladzovanie (antialiasing) a hrúbka čiary

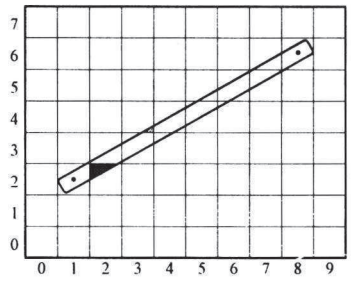
Alias v grafike znamená neželaný optický jav, kaz obrazu. Ak vykreslíme trs úsečiek (napr. polomerov kružnice) s malými uhlovými odchýlkami, objavia sa priečne pruhy, vzniknuté kumuláciou diagonálnych krokov pri rozklade úsečiek. Diagonálny krok v štvorcovom rastri je $\sqrt{2}$ -krát dlhší ako horizontálny krok pre dva obrazové body: obraz úsečky na týchto miestach je jasovo redší a oko vníma skok - obraz úsečky nie je hladký (obr. 5.4)

Pri vyšších nárokoch na rozklad úsečky, aby bol jej obraz čo najrovnejší, postupujeme pomocou metód vyrovnávania a vyhladzovania, ktoré prvýkrat použili v grafike Catmull a Crow. Hlavná myšlienka spočíva v tom, že obrazový bod má na displeji nenulovú plochu, a preto má mať aj zobrazený bod v priestore modelu tiež nenulovú plochu. Dôsledkom toho je, že viditeľné úsečky v reálnom svete majú nenulovú hrúbku (t.j. nie sú to abstraktné matematické úsečky s nulovou hrúbkou).



Obr. 5.4 Aliasing úsečiek

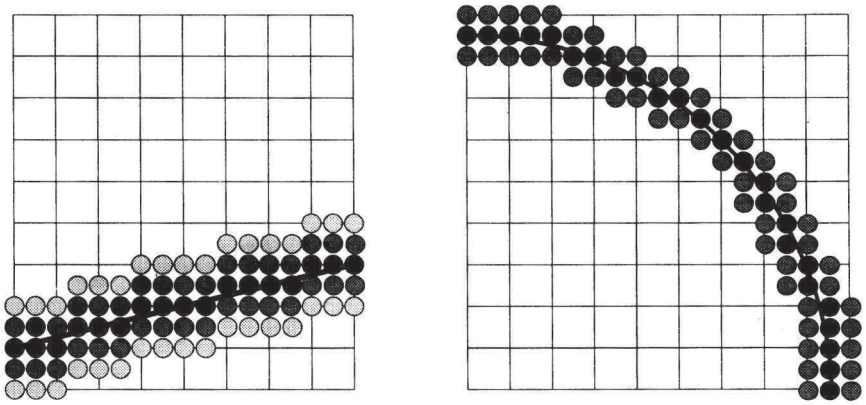
Na obrázku 5.5 je zobrazená úsečka nenulovej hrúbky, položená do rastra. Rastrová sieť je posunutá tak, aby mrežové body ležali vo vnútri každého obdĺžnika siete a nie v priesečníkoch mriežky. Každý obrazový bod je zobrazený obdĺžnikom.



Obr. 5.5 Antialiasing úsečky

Obrazový bod prekrývajúci sa s úsečkou danej hrúbky bude mať intenzitu zobrazenia úmernú ploche ich prieniku. Na tomto príklade vidíme, že pre bielu úsečku na čiernom pozadí má bod (2, 2) úsečky v spoločnom prieniku približne 30% svojej celkovej plochy (teda vysvietime ho 30% intenzitou z maximálnej intenzity) a bod (3, 4) približne len 10% (teda 10% intenzitou). Algoritmy na výpočet spoločnej plochy sú náročné na strojový čas (kvôli zložitosti počítania prienikov a nevyhnutným výpočtom v reálnej aritmetike). Crow a Fuchs[] rozpracovali efektívny spôsob výpočtu pre hladkú úsečku, ale napriek tomu proces výpočtu ostáva ešte pomalý a vo všeobecnosti je pre interaktívnu prácu neprijateľný. Základná myšlienka vyhladzovania sa dá previesť podobne aj pre kružnicu a iné krivky, avšak výpočet je zložitejší.

V grafických systémoch môžeme často zadávať aj **hrúbku čiary** pre úsečky, kružnicu a iné krivky. Rastrové systémy povoľujú zadať hrúbku čiary nepárnym číslom. Pri generovaní čiary ako napríklad pre úsečku alebo kružnicu pri inkrementácii x súradnici, budeme vykresľovať zároveň po stĺpcoch opakujúce pixle zhora aj zdola (5.6).



Obr. 5.6 Hrubé čiary pre vykreslenie úsečky a kružnice

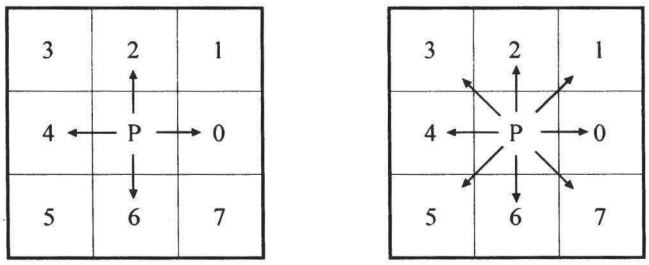
5.5 Vypĺňanie oblasti

V tejto časti popíšeme algoritmy na zobrazovanie oblastí v rastrovej grafike. **Oblasťou** rozumieme množinu bodov, ktoré sú vzájomne súvisle pospájané. V praktických úlohách oblasti môžu vzniknúť rôznymi spôsobmi. Motiváciou sú objekty, na ktorých zobrazenie sú vhodnejšie plošné útvary. Oblasť môžeme vytvoriť špecifikovaním všetkých jej bodov, z ktorých pozostáva, alebo hranicou, ohraničujúcou určitú množinu bodov prípadne aproximovanou pomocou mnohouholníka.

V mnohých grafických aplikáciách operátor interaktívne vytvára oblasť, ktorá sa zvyčajne vykreslí zadanou farbou. Niekedy treba naviac vyplniť časti oblastí rôznymi farbami, preto operátor interaktívne vytvorí potrebné hranice a výberom vnútorného bodu a zadanou farbou spustí vypĺňanie oblastí (obr. 5.8).

5.5.1 Základné pojmy

Na výklad niektorých algoritmov vypĺňania budeme potrebovať pojem **susednosti**. Uvažujme obraz zadaný v bitovej mape, ktorého prvky zodpovedajú jednotlivým obrazovým bodom (pixlom). Vo všeobecnosti každému bodu P prislúcha jeho osem susedov, ktorí sú očíslovaní 0,..,7 (pozri obrázok 5.7). Susedov 0, 2, 4 a 6 nazývame priamymi susedmi bodu P (ináč tiež 4-susedia). Títo 4-susedia majú v základnom rastri s bodom P spoločnú hranu.



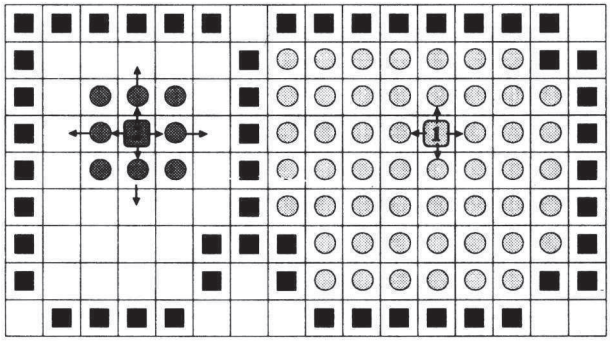
Obr. 5.7 Susednosť (4 a 8) v štvorcovom rastri pre bod P

Môžeme sa dostať k nim štyrmi smermi: vľavo, vpravo, dole, hore. Susedov 1, 3, 5 a 7 nazývame nepriamymi susedmi bodu P a ležia na diagonálach od bodu P. Všetkých susedov (priamych aj nepriamych) nazývame 8-susedmi bodu P. Oblasti poznáme: **4-súvislé** a **8-súvislé**. Každé dva body 4-súvislej (resp. 8-súvislej) oblasti môžeme spojiť postupnosťou 4-susedných (resp. 8-susedných) bodov z tejto oblasti. Mnohé algoritmy pre 8-súvislé oblasti môžeme použiť aj pre 4-súvislé oblasti, ale naopak musíme robiť úpravy v algoritme. Ako si ukážeme, mnohé algoritmy sa dajú pomerne ľahko prepísať z jedného typu súvislej oblasti do druhého typu súvislej oblasti.

Oblasť zadaná svojími vnútornými bodmi sa nazýva **vnútorne definovaná**. Všetky body oblasti majú v tomto prípade zadanú hodnotu. Algoritmy, ktoré pracujú s takýmito oblasťami sa nazývajú vnútorne vypĺňajúce algoritmy. Oblasť zadaná hranicou sa nazýva **hranične definovaná**. Body hranice, majú predpísanú hodnotu a vypĺňajúce algoritmy, ktoré pracujú s takýmito oblasťami, sa nazývajú **hranične vypĺňajúce algoritmy**.

5.5.2 Jednoduché rekurzívne algoritmy

Veľmi jednoduchý vlnový algoritmus rekurzívne priradí bodom 4-súvislej vnútorne definovanej oblasti (farbou *old_colour*) hodnotu *new_colour* (obr. 5.8). Procedúra *read_pixel* zistí farbu obrazového bodu (x, y).



Obr. 5.8 Vypĺňanie oblastí farbou zadaného vnútorného bodu

Algoritmus vlnového vypĺňania *Flood_fill*

```
procedure Flood_fill_4 ( x, y,                { začiatočný vnútorný bod vypĺňania oblasti }
                         old_colour,                          { stará farba oblasti }
                         new_colour : integer);               { nová farba oblasti }
begin
  if read_pixel (x,y) = old_colour then
    begin
        write_pixel (x, y, new_colour);
        Flood_fill_4 (x, y-1, old_colour, new_colour);
        Flood_fill_4 (x, y+1, old_colour, new_colour);
        Flood_fill_4 (x-1, y, old_colour, new_colour);
        Flood_fill_4 (x+1, y, old_colour, new_colour);
    end
end.
```

Algoritmus zistí, či sme už vyšetrovali vnútorný bod (x, y). Ak nie, potom má hodnotu *old_colour* a zmeníme túto hodnotu a rekurzívne vyšetríme jeho 4-susedov. Ak nemá hodnotu *old_colour*, nerobíme nič. Tento algoritmus môžeme jednoducho modifikovať aj pre 8-súvislé vnútorne definované oblasti tak, že vyšetríme rekurzívne pre daný bod jeho 8-susedov. Podobne postupuje jednoduchý algoritmus pre hranične definovanú oblasť:

Algoritmus vypĺňania do hraničných bodov *Bound_fill_4*

```
procedure Bound_fill_4 ( x, y,                    { začiatočný bod vypĺňania oblasti }
                         bound_colour,              { farba hranice oblasti }
                         new_colour: integer);      { nová farba oblasti }
begin
 if read_pixel (x, y) ≠ bound_colour and read_pixel (x, y) ≠ new_colour
 then begin
        write_pixel (x, y, new_colour);
        Bound_fill_4 (x, y-1, old_colour, new_colour);
        Bound_fill_4 (x, y+1, old_colour, new_colour);
        Bound_fill_4 (x-1, y, old_colour, new_colour);
        Bound_fill_4 (x+1, y, old_colour, new_colour);
        end
end.
```

Myšlienka algoritmu *Bound_fill_4* sa podobá na *Flood_fill_4*. Nestačí testovať, či bod (x, y) je časťou oblasti. Treba dva testy, či je vo vnútri oblasti a či už má novú farbu. Túto procedúru môžeme tiež pomerne ľahko modifikovať pre *Bound_fill_8*.

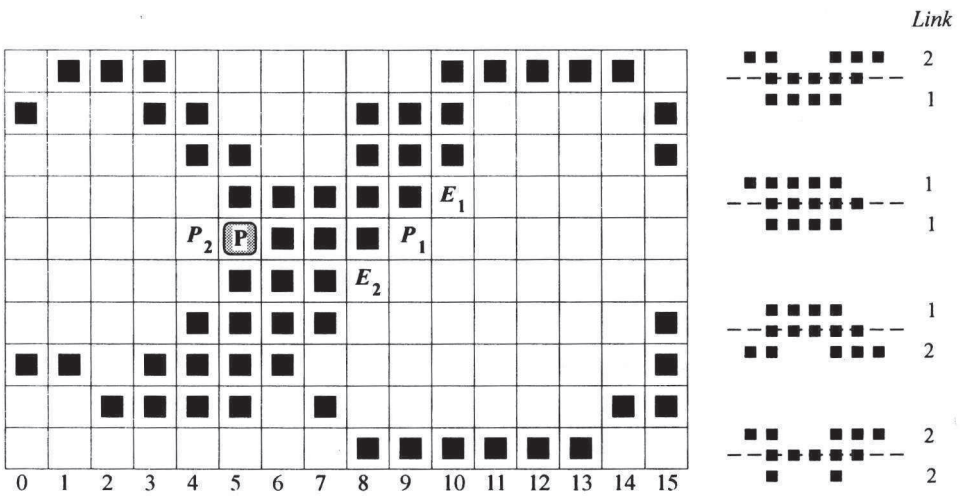
Na obr. 5.9 máme obrysom definovanú hranicu oblasti. Vidíme, že vnútro oblasti má komponenty súvislosti iné pre 8-súvislosť a 4-súvislosť. Preto musíme podľa typu súvislosti vnútra oblasti správne zvoliť *Bound_fill_8* alebo *Bound_fill_4*.

Obr. 5.9 Vypĺňanie podľa 4-súvislosti a 8-súvislosti

5.5.3 Vypĺňanie podľa parity

Jeden zo základných postupov počítačovej grafiky je určenie vnútorného bodu mnohouholníka. Bod je vnútorný, ak polpriamka rovnobežná s osou x vychádzajúca z tohto bodu pretne mnohouholník v nepárnom počte hrán. Skúsime túto analógiu spojitej geometrie (euklidovskej roviny) pretransformovať pre vypĺňanie oblasti v rastrovej grafike.

Najprv popíšeme procedúru *Link_4*, ktorá zistí lokálne správanie sa hranice oblasti v danom bode (riadku). Táto procedúra má dva vstupné parametre *x*, *y*, t.j. súradnice určujúce prvý ľavý hraničný bod P danej hranice. Zaujíma nás správanie sa hranice vzhľadom na počet intervalov v riadkoch nad a pod, ktoré súvisia s daným bodom P. Uvedený algoritmus môžeme modifikovať, aby sme určili výstupné parametre body E_1, E_2, P_1, P_2, t.j. hraničné body hranice nad, pod (E_1, E_2) a tiež body pravý, ľavý (P_1, P_2) ako to vidíme na obrázku 5.10 .



Obr. 5.10 Určenie pomocných bodov pomocou procedúry Link_4

Algoritmus určenia súvislosti hranice

```
procedure Link_4 ( x, y : integer;          { začiatočný bod oblasti }
            var  nad,                       { počet intervalov nad   }
                 pod : integer );           { a intervalov pod pre daný bod P }
begin
   1. Nastav počet nad, pod na nulu.
   2. Ak bod(x-1, y+1) patrí hranici, potom zvýš nad.
   3. Ak bod(x-1, y-1) patrí hranici, potom zvýš pod.
   4. Pokiaľ bod(x, y) patrí hranici, potom 5 - 7.
   begin
      5. Ak bod(x, y+1) patrí hranici a bod(x-1, y+1) nepatrí, potom zvýš nad.
      6. Ak bod(x, y-1) patrí hranici a bod(x-1, y-1) nepatrí, potom zvýš pod.
      7. Zvýš x - t.j. posuň sa v riadku doprava
   end;
   8. Ak bod(x, y+1) patrí hranici a bod(x-1, y+1) nepatrí, potom zvýš nad.
   9. Ak bod(x, y-1) patrí hranici a bod(x-1, y-1) nepatrí, potom zvýš pod.
end.
```

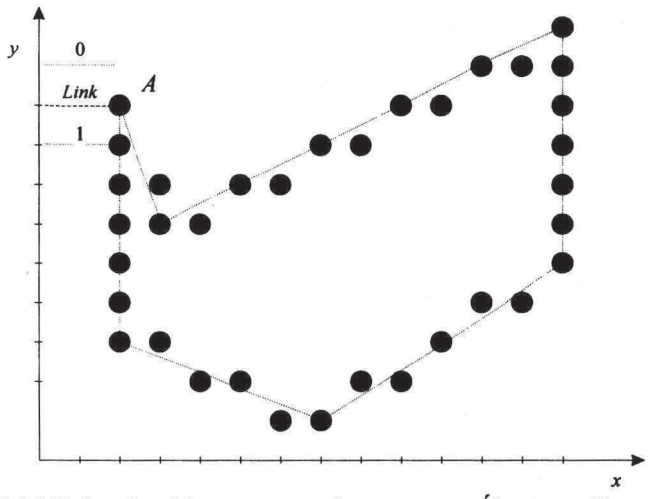
Procedúra na vypĺňanie podľa parity zisťuje, či procedúra *Link_4* (pre prvý ľavý bod hranice) vráti počet intervalov *nad* a *pod*. Prípustné počty intervalov budú len (1, 1) - postupujúca hranica, (0, 2) - lokálne maximum a (2, 0) - lokálne minimum. Ak je len prípustná situácia, potom sa podľa párnosti premennej *par* zavolá procedúra *Fill_Pixel* na nastavenie hodnoty pre vyplnenie daného bodu. Príznak *error_flag* sa nastavuje pre nekorektne zadanú oblasť. Oblasť je uložená v matici h[*m*, *n*] hodnotou *bound_colour*. Po procedúre *Link_4* sa hodnota *x* zmení tak, aby bod P nepatril hranici.

Algoritmus vypĺňania podľa parity *Parity_fill*

```
procedure Parity_fill;
var x, y, Xmax, Ymax, par, error_flag,
    nad, pod, bound_colour, ab: integer;
begin
   for y:= 1 to Ymax do                                  { pre všetky riadky }
   begin
   par:= 0; error_flag:= 0;                              { inicializácia premenných *}
   nad:= 0; pod:= 0;   x:= 1;
   while x <= Xmax do
      begin
      if (h[x, y] ≠ bound_colour) then
         begin
         if (Odd(par)) then Fill_Pixel (x, y);           { vyplň *}
                            x:= x + 1;                   { posuň *}
         end
      else
         begin
         Link_4 (x, y, nad, pod);                  { zisti lokálne správanie hranice *}

         if ((nad=1) and (pod=1)) then par:= par + 1;
                                       ab:= nad + pod;
         if ((ab ≠ 0) or (ab ≠ 2)) then error_flag:= 1;
         end
      end;
   end;
   if (error_flag ≠ 0) then Title ('Error in boundary');   { vypíš chybovú správu }
end.
```

Algoritmus **vypĺňania podľa parity** je vhodný len pre korektne zadanú hranicu. Na obr. 5.11 máme znázornenú jednoduchú oblasť, ktorú sme získali rozkladom mnohouholníka. V bode *A* je veľmi ostrý uhol, a preto procedúra *Link* nám vráti hodnoty *nad* = 1 a *pod* = 0. Čo je pre algoritmus neprípustná hranica. Existujú modifikácie tohto algoritmu, ktoré však vyžadujú podrobnejšiu analýzu [PAVL82].

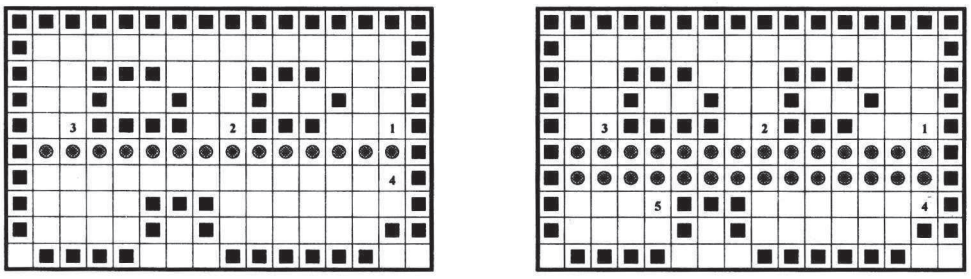


Obr. 5.11 Nekorektná hranica pre algoritmus vypĺňania podľa parity

5.5.4 Zmenšenie hĺbky rekurzie

Popísané rekurzívne procedúry vypĺňania môžu priveľkou hĺbkou rekurzívneho volania preplniť systémový zásobník. Existujú oveľa efektívnejšie algoritmy, ktoré zmenšujú hĺbku rekurzie. Takýto základný algoritmus operuje s horizontálnymi úsekmi vo vnútri oblasti (ohraničené z oboch strán bodmi s hodnotou *bound_colour*). Úseky sa vyhľadajú v cykle, pričom sa vytvárajú od prvého bodu smerom doľava.

Algoritmus najprv vytvorí horizontálny úsek od prvého pixla. Potom vytvorí všetky úseky nad a pod vytvoreným riadkom a každý prvý pravý bod dá do zásobníka. V ďalšej iterácii sa vyberie bod zo zásobníka a proces sa opakuje. Tento postup podľa Smitha ilustruje obrázok 5.12. Procedúra *Link_4* alebo *Link_8* nám vráti informáciu o riadku nad a pod. Dá sa upraviť tak, aby nám vrátil aj požadované obrazové body (na obr. 5.12 pre prvý riadok body 1, 2, 3 a 4).



Obr. 5.12 Vypĺňanie oblasti po riadkoch

5.6 Rastrový rozklad mnohouholníka

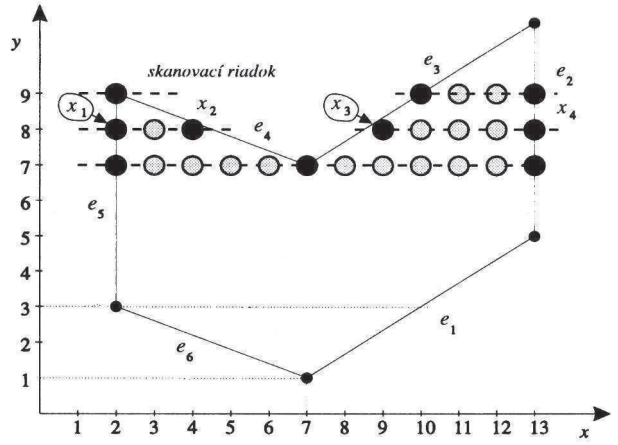
Často treba uskutočniť rasterizáciu mnohouholníka t.j. vyplniť oblasť, ktorej hranicu tvorí mnohouholník. O hranici vtedy máme kompletnú geometrickú informáciu, čo využijeme na konštrukciu efektívneho algoritmu. V tomto prípade môžeme uskutočniť rozklad postupne po riadkoch a preniesť ich do obrazovej pamäti. Základná idea takého prístupu spočíva v tom, že vieme problém rozložiť na vypĺňanie pre jeden riadok. Takýto typ algoritmov nazývame skanovacie algoritmy.

Všimnime si mnohouholník na obr. 5.11. Napadne nás, že môžeme postupovať rozkladom hrán mnohouholníka a využiť niektorý algoritmus vypĺňania. Zaplnenie oblasti pri rozklade mnohouholníka je možné použiť len v určitom postupe. Preto sa na začiatku pamäť obrazu naplní hodnotami *old_colour* a potom sa transformujú hrany mnohouholníka do rastrovej formy s hodnotami *bound_colour*. Nakoniec sa vyhľadá niektorý vnútorný bod a zavolá sa procedúra *Bound_fill_4* pre vyplnenie oblasti. Tento spôsob však nie je efektívny, a preto uvedieme iný spôsob rasterizácie mnohouhlníka.

Medzi vypĺňaním oblasti a rasterizáciou mnohouholníka je podstatný rozdiel. Namiesto oblasti zadanej hodnotami bodov v pamäti obrazu je oblasť definovaná vrcholmi mnohouholníka, pričom sa nepredpokladá zadanie hodnôt v pamäti obrazu.

5.6.1 Riadkový rozklad (skanovací algoritmus)

Algoritmus rozkladu popísaný v tomto odseku odstraňuje mnohé ťažkosti doterajších postupov. Na obrázku 5.13 máme zobrazený mnohouholník s troma skanovacími riadkami. Potrebujeme napríklad určiť, ktoré obrazové body sa nachádzajú vo vnútri mnohouholníka pre skanovací riadok y=8 (na obrázku 5.13 sú to body pre x = 2 - 4 a 9 - 13) a definovať im príslušnú hodnotu vnútra oblasti. Tento proces opakujeme pre každý skanovací riadok prechádzajúci daným mnohouholníkom. Takýmto postupom dostaneme úplny rastrový rozklad mnohouholníka.



Obr. 5.13 Skanovací riadok pre rozklad mnohouholníka do rastra

Dva susedné body majú často vzhľadom na mnohouholník rovnakú polohu (sú vnútri alebo mimo). V takýchto situáciách pozorujeme jav **koherentnosti**, t.j pri zmene od

jedného riadku k druhému sa vo väčšine prípadov situácia vzhľadom na mnohouholník podstatne nezmení. Budeme hľadať tie prípady, pri ktorých sa kvalitatívne zmení situácia t.j. napríklad zmení sa počet úsekov na skanovacom riadku.

Na skanovacom riadku 8 na obrázku 5.13 sa vo vnútri mnohouholníka nachádzajú dva úseky, ktoré je možné zafarbiť (označiť) nasledujúcimi troma fázami:

1. Nájdu sa všetky priesečníky skanovacej priamky so všetkými hranami mnohouholníka. To sú hrany e_2, e_3, e_4, e_5.
2. Usporiadame priesečníky podľa x-ovej súradnice. Na obr. 5.13 sú to súradnice označené x_1, x_2, x_3, x_4 .
3. Zafarbíme všetky body, ktoré sa nachádzajú vždy medzi dvojicami za sebou usporiadaných priesečníkov.

V prvom kroku sa hľadajú priesečníky hrán so skanovacou priamkou. Vodorovné hrany nemôžu mať priesečník jeden bod, a preto ich vykreslíme a zďalšieho skúmania vylúčime. Ďalšie dve fázy sa zdajú byť algoritmicky jednoduché.

Po usporiadaní súradníc môže vzniknuť nasledujúci problém. Čo máme robiť, ak je nepárny počet bodov prieniku pri výbere bodov pre vyplnenie? Uvažujme napríklad riadok $y = 3$, ktorý pretína hrany e_5, e_6 a e_1 v bodoch s x-súradnicou 2, 2 a 10. Podľa obrázku sa nám zdá, že tento bod (2, 3) máme počítať len jedenkrát za obidve hrany e_5 a e_6, ktoré ho obsahujú. Avšak na druhej strane pre skanovací riadok $y = 1$ je len jedna x-súradnica 7, a preto tento vrchol mnohouholníka potrebujeme počítať dvakrát za obidve hrany e_6 a e_1, ktoré ho obsahujú. Posledný prípad sa odlišuje od predchádzajúceho tým, že mnohouholník v tomto bode nadobúda lokálne minimum a v predchádzajúcom prípade nie. Ináč povedané, predchádzajúci a nasledujúci vrchol mnohouholníka sa nachádza v tej istej polrovine, ktorá je určená skanovacou priamkou (v tomto prípade počítame vrchol dvakrát, pretože je lokálny extrém mnohouholníka) alebo vrcholy sa nachádzajú v rôznych polrovinách (a v tomto prípade počítame vrchol len jedenkrát, pretože je vo vrchole monotónne spojenie). Všeobecný postup algoritmu riadkového rozkladu môžeme sformulovať takto:

Algoritmus Scan_Line

1. Skrátime zdola hrany naväzujúce vo vrchole monotónneho spojenia.
2. Vylúčime vodorovné hrany.
3. V cykle od minimálnej po maximálnu súradnicu y mnohouholníka:
  3.1 Nájdeme priesečníky skanovacej priamky so všetkými hranami.
  3.2 Usporiadame priesečníky podľa x-ovej súradnice.
  3.3 Vykreslíme všetky body, ktoré sú medzi dvojicami za sebou.
4. Vykreslíme hranicu mnohouholníka.

Tento algoritmus nie je efektívny. Preto ho v nasledujúcej časti upravíme.

5.6.2 Koherentnosť a tabuľka hrán

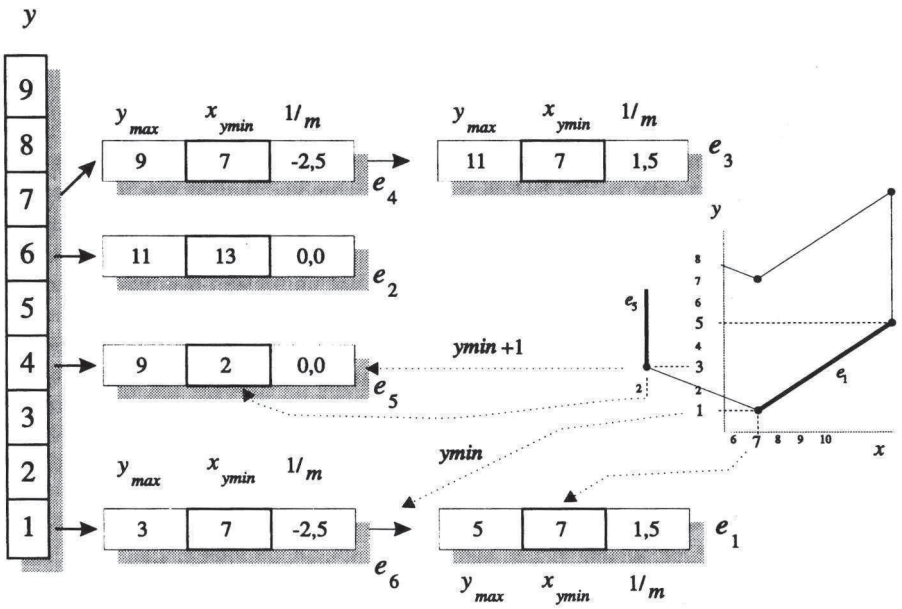
Ako sme už skôr spomenuli, najprv sa nájdu priesečníky skanovacej priamky so všetkými hranami mnohouholníka. Tento výpočet je pomalý, pretože treba každú hranu

porovnávať so skanovacou priamkou. Všimneme si, že pri hľadaní priesečníkov nás môžu zaujímať len niektoré hrany. Ak hrana pretína i-ty skanovací riadok, potom obvykle pretína aj $(i+1)$ skanovací riadok (*koherencia*). Pri prechode od jedného riadku k druhému môžeme vypočítať novú x-súradnicu priesečníka so skanovacím riadkom nasledovne:

$$x(i+1) = x(i) + 1/m ,$$

kde m je smernica úsečky (t.j. $m = dy/dx$ a $dy = 1$, takže $1/m = dx$).

Pre každý skanovací riadok budeme brať do úvahy len tie hrany, ktoré sa s ním pretínajú. Tieto budú v tabuľke aktívnych hrán (TAH). Pri prechode k nasledujúcemu riadku nové x-súradnice vypočítame podľa predchádzajúce vzorca. Všetky nové hrany, ktoré sa pretínajú s týmto riadkom budú zaradené do TAH. Tie hrany, ktoré sa už nepretínajú, budú z TAH vylúčené.

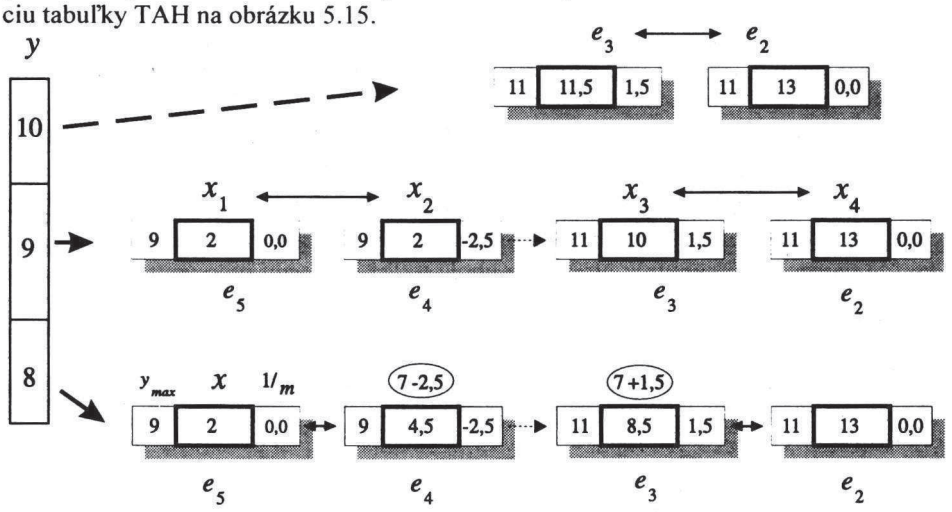


Obr. 5.14 Tabuľka hrán pre daný mnohouholník

Kvôli rýchlosti výpočtov sa vytvorí tabuľka hrán (TH), v ktorej sú všetky hrany usporiadané podľa svojej minimálnej y-súradnice. Môžeme ju vytvoriť **hašovacou tabuľkou** podľa rastra v smere osi y. Táto tabuľka sa obvykle vytvorí pomocou usporiadaných zoznamov. Pre každý skánovací riadok je vytvorený zoznam tých hrán, ktoré dosahujú minimálnu y-súradnicu v tomto riadku. Pre každú hranu odložíme do TH:

1. maximálna y-súradnicu hrany ($ymax$),
2. x-súradnica, zodpovedajúca bodu s minimálnou y-súradnicou hrany,
3. prevrátená hodnota smernice ($1/m$).

Každá trojica sa vloží usporiadane do zoznamu podľa y-súradnice. Skrátime zdola hrany naväzujúce vo vrchole monotónneho spojenia, a preto tieto hrany vložíme ich do zoznamu TH pre $y_{min}+1$. V zozname trojice držíme usporiadané podľa x. Každý prvok TH obsahuje dve nemenné hodnoty y-súradnicu ($ymax$) a konštantu ($1/m$). Druhý člen 'x-súradnica' sa bude neskôr upravovať v TAH. Na obrázku 5.14 je tabuľka hrán pre mnohouholník z obrázku 5.13. Potom v TAH budeme meniť súradnicu $x = x + 1/m$ pri prechode z jedného riadku na druhý. Pre súradnice $y = 8$ až 10 máme znázornenú situáciu tabuľky TAH na obrázku 5.15.



Obr. 5.15 Tabuľka aktívnych hrán

5.6.3 Upravený algoritmus skanovania

Po vytvorení tabuľky hrán môžeme sformulovať algoritmus riadkového skanovania nasledovne:

Upravený algoritmus Scan_Line

begin

1. Vyberieme pre minimálnu hodnotu y z tabuľky hrán TH s neprázdnym zoznamom.

2. Inicializujeme tabuľku aktívnych hrán TAH = ∅, t.j. prázdny zoznam.

3. Opakujeme krok 3 dovtedy, kým nebude TAH a TH prázdna.

3.1 Pre danú hodnotu y pridáme zodpovedajúci zoznam do tabuľky TAH, pričom musíme zachovať usporiadanie podľa x-súradníc.

3.2 Vyberieme z tabuľky TAH dvojice súradníc x za sebou nasledujúcich v zozname a medzi nimi vypĺňame úseky odpovedajúcich bodov pre dané x.

3.3 Zrušíme tie hrany z TAH, pre ktoré nastala rovnosť $y = ymax$.

3.4 Pre všetky hodnoty x z tabuľky TAH musíme vypočítať nové hodnoty x (upravíme $x := x + 1/m$).

3.5 Urobíme usporiadanie podľa x-súradníc v tabuľke TAH.

3.6 Zvýšime aktuálnu hodnotu y na $y+1$ a vrátime sa k bodu 3.

4. Mnohouholník je vyplnený.

end.

Upravený algoritmus pracuje rýchlo, ale existuje ďalšie jeho zrýchlenie. Jednou z možností je upraviť výpočet x-súradnice do celočíselnej aritmetiky. V algoritme sa stále inkrementuje $y=y+1$. Preto Bresenhamov algoritmus sa dá použiť len pre hrany v 2. oktante. Ak je hrana v 1. oktante, musíme prispôsobiť výpočet súradnice x.

Tento algoritmus sa dá jednoducho upraviť aj pre **šrafovanie mnohouholník**. Stačí v cykle inkrementovať súradnicu $y=y+k$. Týmto spôsobom získame vodorovné šrafovanie posunuté v smere y o k. Algoritmus môžeme upraviť aj pre šikmé šrafovanie o uhol α tak, že najprv mnohouholník otočíme o uhol $-\alpha$. Pri šrafovaní prerušovanými čiarami je dobré určiť vzťažný bod, napr. začiatok súradníc, a určovať prerušovanú čiaru podľa neho.

Algoritmus modifikujeme tiež pre zobrazovanie mnohouholníkov v trojrozmernom priestore pre zobrazenie len viditeľných neprekrývajúcich sa častí.

6

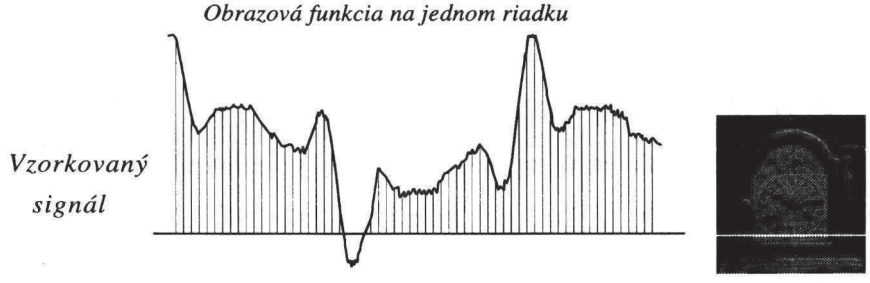
Matematické základy spracovania obrazu

Pojem obrazu používame pre dvojrozmernú funkciu intenzity $f(x, y)$, kde x a y sú priestorové súradnice v rovine a funkčná hodnota odpovedá jasu v tomto bode. Digitálny obraz môže byť diskretizovaný v priestorových i jasových hodnotách. V nasledujúcich častiach budeme reprezentovať **digitálny obraz** pomocou dvojrozmernej matice.

Pri spracovaní obrazu hrá významnú úlohu obrazová funkcia a jej transformácia pomocou harmonických funkcií. V tejto kapitole uvedieme dôležitú **Fourierovu transformáciu** a jej vlastnosti. Najprv uvedieme spojitý prípad transformácie, ktorý prevedieme na diskrétny. Pre diskrétny prípad sú predpoklady existencie Fourierovej transformácie vždy splnené, a preto ju neskôr budeme môcť využívať pri zvýraznení, ostrení, kódovaní a popise obrazu.

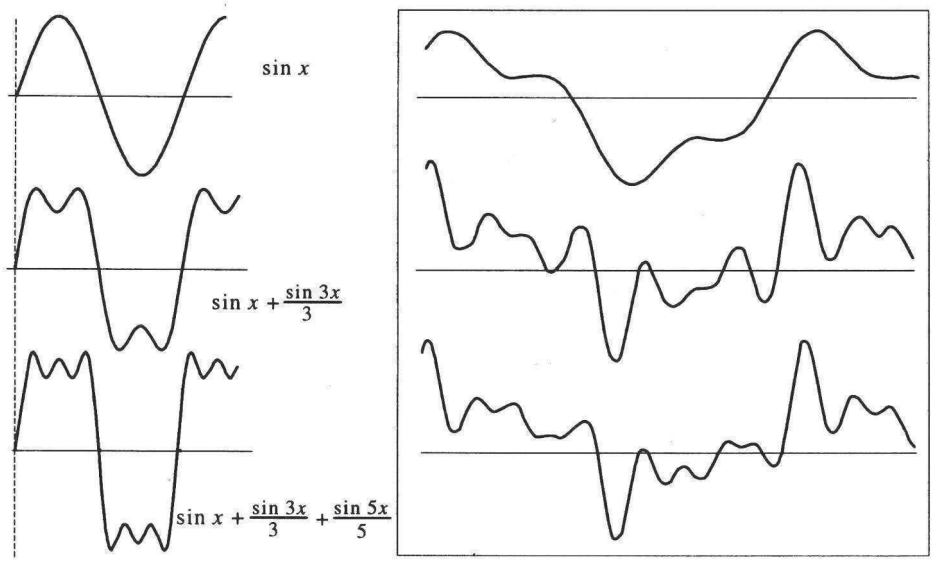
6.1 Úvod do Fourierovej transformácie

Obrazovou funkciou rozumieme obraz zosnímaný do digitálnej formy, ktorú môžeme reprezentovať dvojrozmernou maticou úrovní jasu. **Vzorkovanie** (*sampling*) ukazuje vzájomnú väzbu medzi spojitým a diskrétnym prípadom. Na obr. 6.1 máme znázornený jeden riadok snímaného obrazu. Často sa stretávame so spracovaním jednorozmerného signálu. Tieto metódy môžeme jednoducho aplikovať aj pre dvojrozmerný obraz. Signál môžeme reprezentovať aj vo frekvenčnej oblasti ako súčet periodických funkcií sin a cos. Každá periodická časť funkcie sin (sínusová vlna) reprezentuje v signáli frekvenčnú časť.



Obr. 6.1 Obrazová funkcia získaná vzorkovaním

Na obr. 6.2 a) máme znázornený súčet funkcií $\sin(i.x)/i$ pre $i = 1, 2$ a 3. Čiže pre periodickú funkciu hľadáme rozklad na súčet harmonických funkcií sin. V praktickom prípade obrazovej funkcie nemôžeme očakávať, že vyjadríme signál ako súčet harmonických funkcií. Na obr. 6.2 b) máme príklad ako pomocou súčtu harmonických funkcií sa blížime k danému signálu. Z matematickej hľadiska vieme, že nám nestačí pre spojitú funkciu brať súčet, ale musíme spojite integrovať, vytvárať integrál s harmonickými funkciami. To tvorí základ Fourierovej analýzy, ktorú informatívne uvedieme v tejto časti.



a) b)

Obr. 6.2 Signál vyjadrený frekvenciou harmonických funkcií

Uvedieme **jednorozmerný prípad**, ktorý využívame tiež pri spracovaní signálu. Nech $f(x)$ je spojitá funkcia reálnej premennej x. **Fourierova transformácia** funkcie $f(x)$, označená ako $F\{f(x)\}$, je definovaná rovnicou:

$$F\{f(x)\} = F(u) = \int_{-\infty}^{\infty} f(x) \cdot [\cos 2\pi ux - i \sin 2\pi ux]\, dx \ . \qquad (1)$$

Pre danú funkciu $F(u)$ definujeme **inverznú Fourierovu transformáciu**:

$$F^{-1}\{F(u)\} = f(x) = \int_{-\infty}^{\infty} F(u) \cdot [\cos 2\pi ux + i \sin 2\pi ux]\, du \ . \qquad (2)$$

Premennej u hovoríme **frekvenčná premenná** a x je **priestorová premenná**. Predchádzajúce vzťahy (1) a (2) ukazujú prechod priestorovej premennej na frekvenčnú a naopak. Dá sa dokázať existencia týchto transformácií, ak $f(x)$ je spojitá a

integrovateľná a $F(u)$ je integrovateľná. Pretože Fourierova transformácia je komplexná funkcia, môžeme zapísať:

$$F(u) = R(u) + iI(u),$$

kde $R(u)$ a $I(u)$ sú reálne funkcie reprezentujúce reálnu a imaginárnu časť. Často využívame **exponenciálny tvar**:

$$F(u) = |F(u)|.e^{i\phi(u)},$$

kde

$|F(u)| = (R^2(u) + I^2(u))^{1/2}$ je **Fourierovým spektrom funkcie** $f(x)$ a

$\phi(u) = \text{arctg}\,(I(u)/R(u))$ je **fázový uhol.**

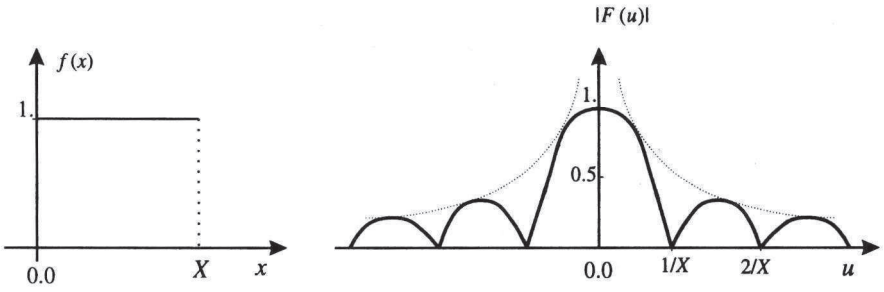
Často tiež využívame funkciu štvorca spektra:

$$P(u) = |F(u)|^2 = R^2(u) + I^2(u).$$

Pre úpravu komplexnej funkcie využívame Eulerovu formulu:

$$\exp(-i2\pi ux) = \cos 2\pi ux - i \cdot \sin 2\pi ux$$

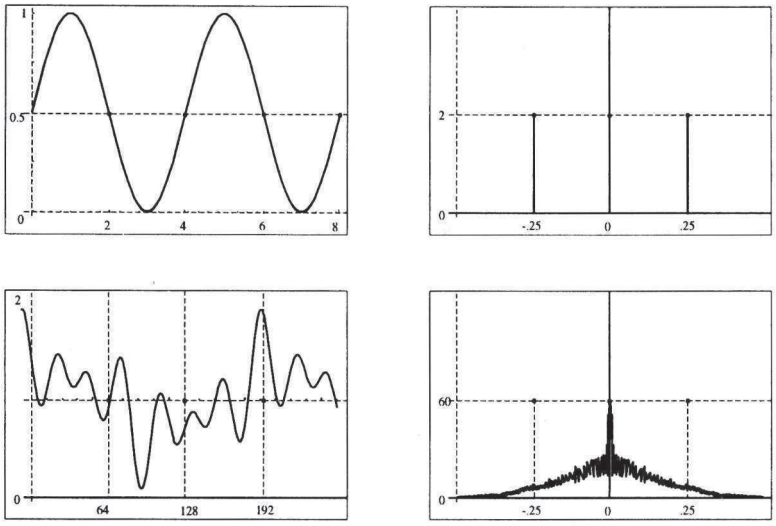
Príklad 1: Jednoducho zo vzťahu (1) môžeme vyjadriť Fourierovu transformáciu konštantnej funkcie znázornenej na obr. 6.1. a). Spektrum funkcie ilustruje obr. 6.3 b).



a) b)

Obr. 6.3 Spektrum konštantnej funkcie

Na nasledujúcom obr. 6.4 vidíme spektrum dvoch funkcií, funkcie sin a zosnímanej obrazovej funkcie na jednom riadku. Obrázok ilustruje poznatok, že informácie o vyšších frekvenciách sa ukladajú bližšie k nule a naopak informácie o nízkych frekvenciách ďalej od nuly.



Obr. 6.4 Spekrum funkcie sin a jednorozmerného signálu

Fourierovu transformáciu (1) a (2) môžeme rozšíriť pre funkciu $f(x, y)$ dvoch premenných. Ak $f(x, y)$ je spojitá a integrovateľná a $F(u, v)$ je integrovateľná, potom nasledujúca Fourierova transformačná dvojica je:

$$F\{f(x,y)\} = F(u, v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y) \cdot \exp[-i2\pi(ux + vy)]\, dxdy$$

a

$$F^{-1}\{f(x,y)\} = f(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v) \cdot \exp[i2\pi(ux + vy)]\, dudv\ ,$$

kde u a v sú frekvenčné premenné.

Podobne ako pre jednorozmerný prípad definujeme spektrum, fázový uhol a štvorec spektra pre dvojrozmerný prípad:

$$|F(u, v)| = (R^2(u, v) + I^2(u, v))^{1/2}, \quad \phi(u, v) = \text{arctg}\,(I(u, v)/R(u, v)),$$

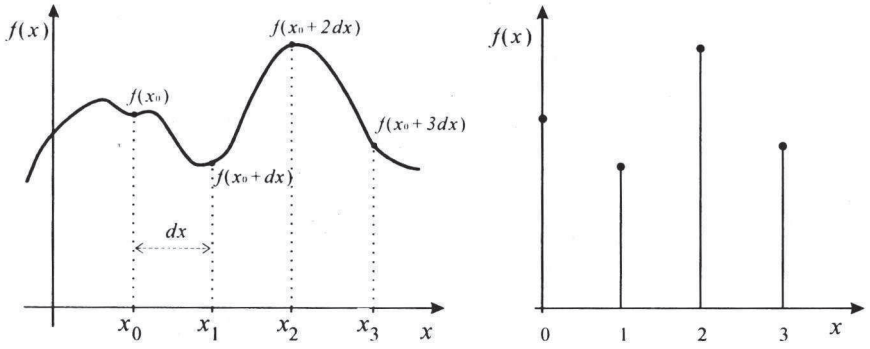
$$P(u, v) = |F(u, v)|^2 = R^2(u, v) + I^2(u, v).$$

6.2 Diskrétna Fourierova transformácia

Predpokladajme, že pre spojitú funkciu $f(x)$ vyberieme postupnosť funkčných hodnôt s pravidelným krokom dx, $\{f(x_0), f(x_0+dx), f(x_0+2dx),..., , f(x_0+Ndx)\}$. Kvôli jednoduchšiemu zápisu vzorkovania budeme písať:

$$f(x) = f(x_0+x.dx), \qquad (3)$$

čo predstavuje pravidelné **vzorkovanie spojitej funkcie pretransformovaným krokom**. Máme postupnosť funkčných hodnôt $\{f(0), f(1), f(2), ..., f(N)\}$ pre celočíselné hodnoty $x = 0, 1, 2, ..., N$, .



Obr. 6.5 Vzorkovanie spojitej funkcie

Diskrétna Fourierova transformácia je definovaná ako dvojica funkcií $f(x)$, $F(u)$:

$$F(u) = \frac{1}{N+1} \sum_{x=0}^{N} f(x) \cdot \exp(-i2\pi ux/(N+1)) \text{ , pre } u = 0, 1, ..., N, \quad (4)$$

$$f(x) = \frac{1}{N+1} \sum_{u=0}^{N} F(u) \cdot \exp(i2\pi ux/(N+1)) \text{ , pre } x = 0, 1, ..., N. \quad (5)$$

Hodnoty $u = 0, 1, ..., N$ pre diskrétnu Fourierovu transformáciu zodpovedajú vzorkám spojitej transformácie v bodoch $0, dx, 2dx, ...$ Označenie $F(u)$ reprezentuje hodnoty $F(u.du)$, podobne ako pre funkciu $f(x)$. Môžeme ukázať, že prírastky dx a du sú vo vzťahu:

$$du = 1/((N+1)dx).$$

Na nasledujúcom obr. 6.6 máme ukážku rekonštrukcie spojitej funkcie po vzorkovaní. Problém určenia vzorkovacieho kroku dx rieši **Shannonova veta**, ktorá hovorí, že vzorkovacia frekvencia musí byť aspoň dvakrát väčšia ako najvyššia frekvencia.

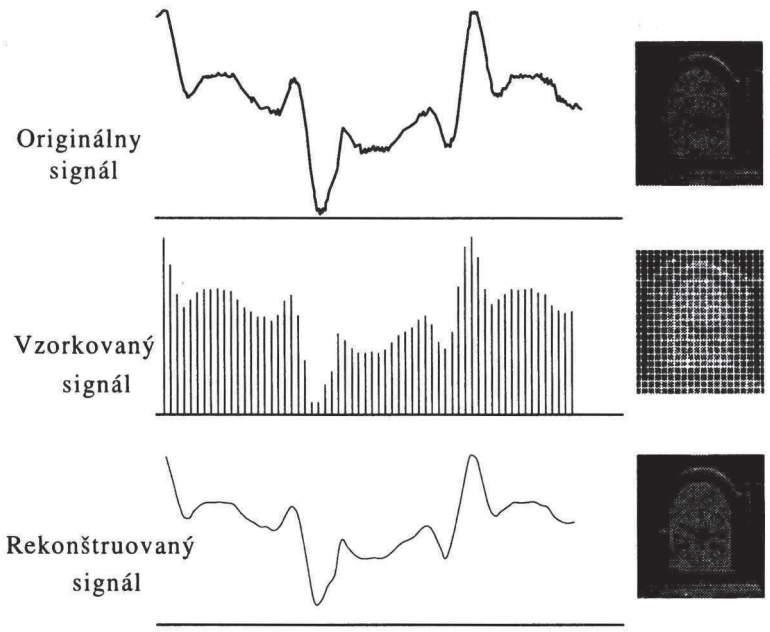
Pre funkciu dvoch premenných podobne definujeme dvojicu Fourierových funkcií $f(x, y)$, $F(u, v)$:

$$F(u, v) = \frac{1}{(M+1)\cdot(N+1)} \sum_{x=0}^{M} \sum_{y=0}^{N} f(x, y) \cdot \exp(-i2\pi(ux/(M+1) + vy/(N+1)))$$

pre $u = 0, 1, ..., M$, pre $v = 0, 1, ..., N$,

$$f(x, y) = \sum_{u=0}^{M} \sum_{v=0}^{N} F(u, v) \cdot \exp(i2\pi(ux/(M+1) + vy/(N+1))$$

pre $x = 0, 1, ..., M$, pre $y = 0, 1, ..., N$.

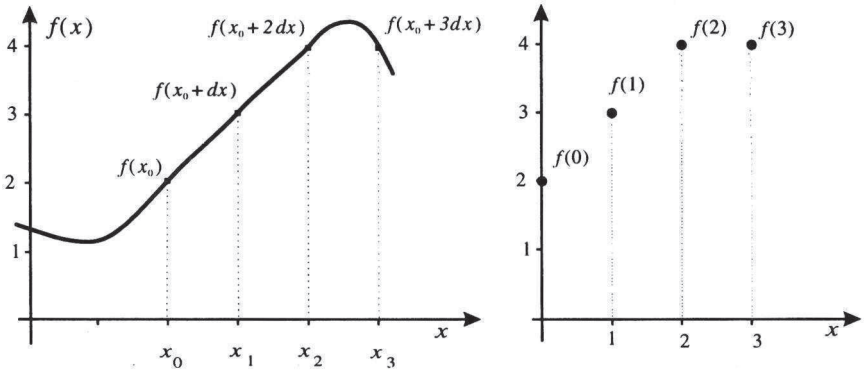


Obr. 6.6 Rekonštruovaný signál po vzorkovaní

Vzorkovanie spojitej dvojrozmernej funkcie uskutočníme osobitne v premennej x a y s krokom dx a dy. Podobne ako v jednorozmernom prípade dostaneme diskrétnu funkciu $f(x+dx, y+dy)$ pre $x = 0, 1, ..., M$ a $y = 0, 1, ..., N$. Vzorkovanie a frekvencia sú vo vzťahu:

$$du = 1/((M+1)dx), \text{ a } dv = 1/((N+1)dy).$$

Príklad 2: Ako ilustráciu rovníc (4) a (5) ukážeme výpočet Fourierovej transformácie funkcie definovanej na obr. 6.7 a). Predpokladajme, že funkcia je navzorkovaná v štyroch bodoch $x_0 = 0.5$, $x_1 = .75$, $x_2 = 1.0$, $x_3 = 1.25$. Po celočíselnej transformácii (3) máme funkčné hodnoty ako sú zobrazené na obr. 6.7 b).



a) b)

Obr. 6.7 Jednoduchá funkcia a jej vzorky

Výslednú Fourierovu transformáciu vypočítame:

$$F(0) = 1/4 \sum_{x=0}^{3} f(x)\exp[0] = 1/4.\ (f(0) + f(1) + f(2) + f(3))$$

$$= 1/4.\ (2 + 3 + 4 + 4) = 3.25$$

$$F(1) = 1/4 \ = 1/4.\ (2e^{0} + 3e^{-i\pi/2} + 4e^{-i\pi} + 4e^{-i3\pi/2}) = 1/4.\ (-2+i),$$

kde posledný vzťah sme upravili pomocou Eulerovej formuly. Podobne:

$$F(2) = 1/4 \sum_{x=0}^{3} f(x)\exp[-i4\pi x/4] \ = 1/4.\ (2e^{0} + 3e^{-i\pi} + 4e^{-i2\pi} + 4e^{-i3\pi}) = 1/4.(-1+i0),$$

$$F(3) = 1/4 \sum_{x=0}^{3} f(x)\exp[-i6\pi x/4] \ = 1/4.\ (2e^{0} + 3e^{-i3\pi/2} + 4e^{-i3\pi} + 4e^{-i9\pi/2}) = -1/4.\ (2+i).$$

Fourierovo spektrum je definované:

$$|F(0)| = 3.25, \qquad |F(1)| = [(2/4)^2 + (1/4)^2]^{1/2} = \sqrt{5}/4,$$

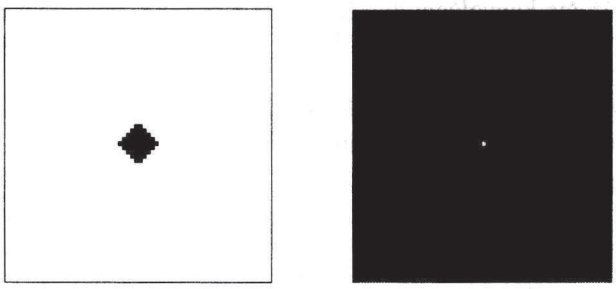
$$|F(2)| = [(1/4)^2 + (0/4)^2]^{1/2} = 1/4, \qquad |F(3)| = [(2/4)^2 + (1/4)^2]^{1/2} = \sqrt{5}/4.$$

Pre výpočet Fourierovej transformácie sme použili hodnoty $f(0), f(1), f(2), f(3)$ pre výpočet každého bodu. Dá sa dokázať, že to nie je nutné. Existuje algoritmus **rýchlej Fourierovej transformacie** (*FFT - Fast Fourier transform*), ktorý zníži počet aritmetických výpočtov na minimum.

Kvôli lepšiemu zobrazovaniu spektra sa často používa funkcia:

$$D(u, v) = \log(1 + |F(u, v)|). \qquad (6)$$

Na nasledujúcom obr. 6.8 máme zobrazenú diskrétnu obrazovú funkciu konštantnú na pootočenom štvorci. V časti b) obrázku vidíme zobrazené spektrum tejto funkcie. Všimnime si, že spektrum funkcie na diagonálach je diskrétnym prípadom jednorozmerného spojitého prípadu z obr. 6.3. Hodnoty funkcie sú odlíšené jasom.



a) b)

Obr. 6.8 Spektrum diskrétnej obrazovej funkcie

6.3 Niektoré vlastnosti Fourierovej transformácie

V tejto časti ukážeme základné vzťahy pre prepojenie medzi priestorovou a frekvenčnou premennou. Tieto vzťahy nazývame **konvolúcia** a **korelácia**. Majú dôležitý význam pri spracovaní obrazu.

6.3.1 Separabilita

Diskrétnu Fourierovu transformáciu môžeme rozložiť po častiach oddelene pre premennú x a y:

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \exp(-i2\pi ux/N) \sum_{y=0}^{N-1} f(x,y) \cdot \exp(-i2\pi vy/N)$$

pre $u, v = 0, 1, ..., N-1$, a

$$f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \exp(i2\pi ux/N) \sum_{v=0}^{N-1} F(u,v) \cdot \exp(i2\pi vy/N)$$

pre $x, y = 0, 1, ..., N-1$.

6.3.2 Konvolúcia

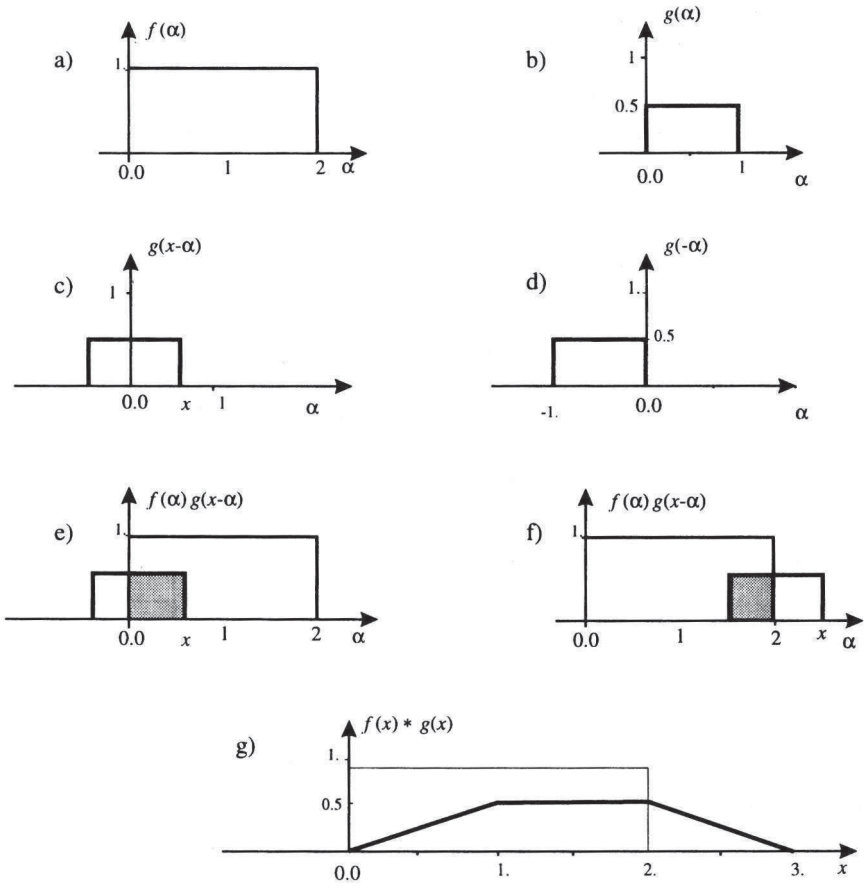
Konvolúciu dvoch funkcií $f(x)$ a $g(x)$ označujeme ako $f(x)*g(x)$ a definujeme pomocou integrálu:

$$f(x)*g(x) = \int_{-\infty}^{\infty} f(\alpha).g(x-\alpha)\, d\alpha\,, \qquad (7)$$

kde α je integrálna premenná. Vzťah (7) sa ťažko interpretuje, a preto si ukážeme jednoduchý prípad.

Príklad 3: Ukážeme konvolúciu dvoch funkcií $f(x)$ a $g(x)$ zobrazených na obr. 6.9. Pred integráciou si predstavíme funkciu $g(x-\alpha)$, ktorá je postupne zobrazená v dvoch krokoch na obr. 6.9 časti c) a d).

Pre každú hodnotu x uskutočníme nasledujúci výpočet, vynásobíme $f(\alpha)$ zodpovedajúcou hodnotou $g(x-\alpha)$ a integrujeme súčin týchto funkcií v intervale od $-\infty$ do ∞. Súčin hodnôt $f(\alpha)$ a $g(x-\alpha)$ je zobrazený na obr. 6.9 e). Platí to pre hodnoty $0 \le x \le 1$. Pretože súčin je rovný 0 pre hodnoty α mimo intervalu $\langle 0, x\rangle$, platí $f(x)*g(x) = x/2$ - plocha narastá s rastom x, čo je šedá časť obr. 6.9 e). Pre x z intervalu $\langle 2, 3\rangle$ máme podobne príslušný integrál zobrazený ako šedú plochu na obr. 6.9 f). V tomto prípade je $f(x)*g(x) = (1-x/2)$.



Obr. 6.9 Grafické zobrazenie konvolúcie dvoch funkcií

Funkcia $f(\alpha).g(x-\alpha)$ je nulová pre hodnoty x mimo intervalu $\langle 0, 3\rangle$, preto:

$$f(x)*g(x) = \begin{cases} x/2 & 0 \le x \le 1 \\ 0.5 & 1 \le x \le 2 \\ 1.5 - x/2 & 2 \le x \le 3 \\ 0 & \text{inak} \end{cases}.$$

Výsledok je zobrazený na obr. 6.9 g). Všimnime si, že po konvolúcii sa funkcia $f(x)$ zmenila tak, že frekvenciu sme zmenšili a ostrú zmenu funkcie sme zhladili.

Z frekvenčnej analýzy vyplýva, že $f(x)*g(x)$ a $F(u).G(u)$ tvoria Fourierovskú transformačnú dvojicu. Inými slovami, ak $f(x)$ má Fourierovu transformáciu $F(u)$ a $g(x)$ má Fourierovu transformáciu $G(u)$, potom $f(x)*g(x)$ má Fourierovu transformáciu $F(u).G(u)$. Tento výsledok formálne zapisujeme:

$$f(x)*g(x) \Leftrightarrow F(u).G(u).$$

Dvojrozmerná konvolúcia je definovaná analogicky, pre dve funkcie $f(x, y)$ a $g(x, y)$ máme:

$$f(x, y)*g(x, y) = \int\int f(\alpha, \beta).g(x-\alpha, y-\beta)\, d\alpha\, d\beta .$$

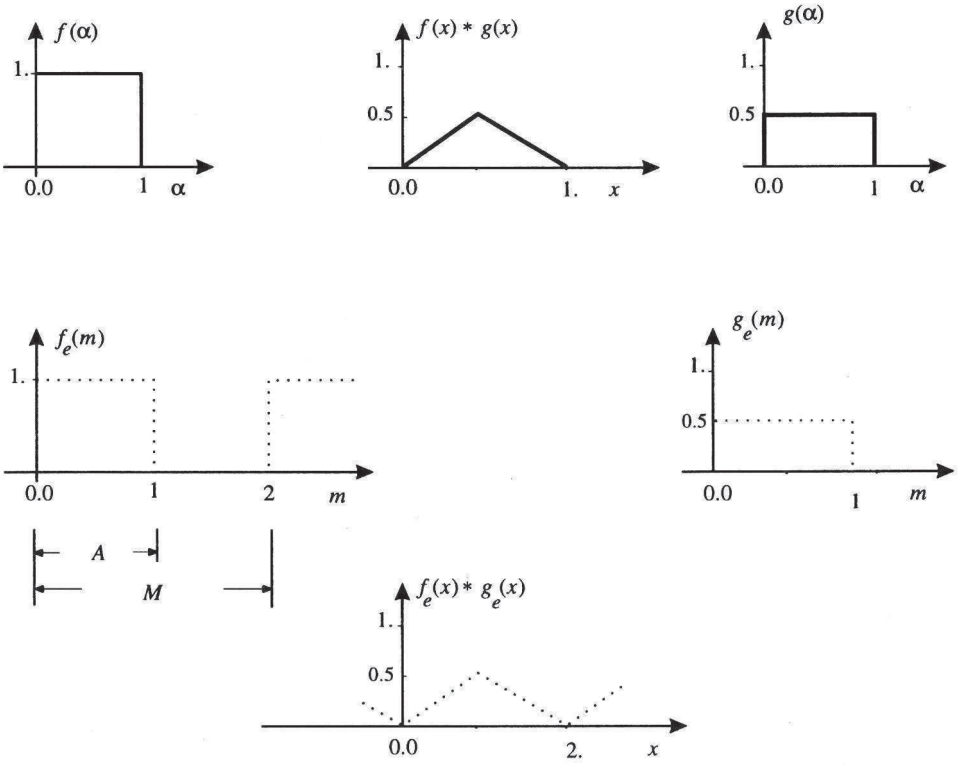
Podobne platí konvolučná veta:

$$f(x, y)*g(x, y) \Leftrightarrow F(u, v).G(u, v).$$

Uvedieme diskrétny prípad konvolúcie. Predpokladajme, že zo spojitých funkcií $f(x)$ a $g(x)$ sme vybrali A a B vzoriek, t.j. $\{f(0), f(1), ..., f(A-1)\}$, a $\{g(0), g(1), ..., g(B-1)\}$. Diskrétne hodnoty $f(x)$ a $g(x)$ môžeme periodicky rozšíriť. Vyberieme hodnotu M:

$$M \geq A + B - 1.$$

Dodefinujeme funkciu $f(x)$ na intervale $\langle A, M-1\rangle$ nulou, t.j. $f_e(x) = 0$, pre $A \leq x \leq M-1$ a funkciu $g(x)$ na intervale $\langle B, M-1\rangle$ nulou, t.j. $g_e(x) = 0$, pre $B \leq x \leq M-1$.



Obr. 6.10 Porovnanie spojitej a diskrétnej konvolúcie

Na základe týchto funkcií definujeme **diskrétnu konvolúciu** $f_e(x)$ a $g_e(x)$ výrazom:

$$f_e(x)*g_e(x) = \sum_{m=0}^{M-1} f_e(m).g_e(x-m),$$

pre $x = 0, 1, ..., M-1$.

Konvolučná funkcia je diskrétna s periódou M, s hodnotami $x = 0, 1, ..., M-1$ pre $f_e(x)*g_e(x)$.

Príklad 4: Diskrétny proces konvolúcie ilustrujeme na obr. 6.10 pre spojitú a diskrétnu konvolúciu. Pre diskrétny prípad vyberáme A vzoriek pre obidve funkcie $f(x)$ a $g(x)$ z intervalu $\langle 0, 1 \rangle$, periódu vyberáme $M = A + B - 1 = 2A - 1$.

Dvojrozmerná diskrétna konvolúcia $f_e(x, y)$ a $g_e(x, y)$ je daná predpisom:

$$f_e(x, y)*g_e(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_e(m, n).g_e(x-m, y-n)$$

pre $x = 0, 1, ..., M-1$ a $y = 0, 1, ..., N-1$. Hodnoty M a N vyberáme podobne:

$$M \geq A + C - 1 \text{ a } N \geq B + D - 1,$$

kde A, B, C, D sú vzorkovacie počty pre premenné x a y funkcií f a g.

Základom diskrétnej frekvenčnej analýzy obrazu je **konvolučná veta**, podobne ako pre spojitý prípad. Ak $g(x, y)$ je obraz vytvorený pomocou konvolúcie obrazu $f(x, y)$ a pozične invariantného operátora $h(x, y)$, potom zapíšeme konvolúciu:

$$g(x, y) = h(x, y)*f(x, y).$$

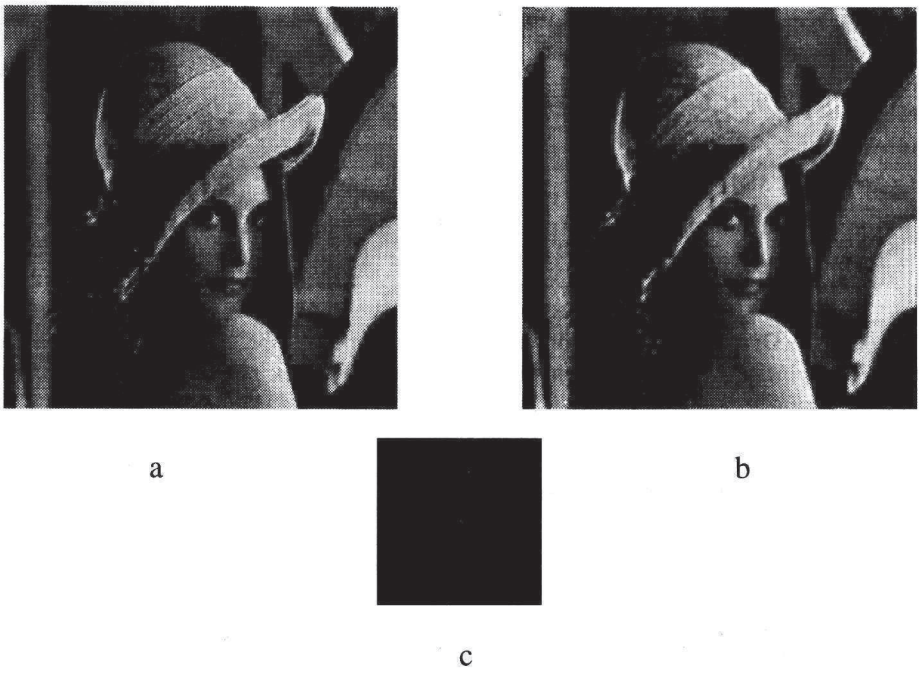
Podľa konvolučnej vety platí nasledujúca frekvenčná závislosť:

$$G(u, v) = H(u, v).F(u, v),$$

kde G, H a F sú Fourierove transformácie pre g, h a f. Transformáciu $H(u, v)$ niekedy nazývame **transformačnou funkciou procesu** a funkciu h **jadrom konvolúcie**. Ako neskôr ukážeme, funkciu obrazu $f(x, y)$ zmeníme výberom funkcie $H(u, v)$ a zmenu vypočítame podľa konvolučnej vety:

$$g(x, y) = F^{-1}\big(H(u, v).F(u, v)\big).$$

Na obr. 6.11 je ukážka známeho obrazu Leny, na ktorý sa aplikovala konvolúcia podobnou konštantnou funkciou, ako v príklade 4, ale pre dvojrozmerný prípad. V časti c) obr. 6.11 je zobrazené logaritmické spektrum obrazu Leny podľa vzťahu (6).



Obr. 6.11 a) obraz Leny, b) jeho zmena po konvolúcii, c) logaritmické spektrum obrazu

7

Jasová korekcia a filtrácia obrazu

7.1 Úvod

Metódy predspracovania obrazu slúžia na zlepšenie kvality obrazu pred jeho ďalším spracovaním. Výber metódy spracovania je často podmienený našou predbežnou znalosťou obrazu, napríklad aký by mal byť obraz. Na druhej strane nemôžeme očakávať, že zo zle nasnímaného obrazu dostaneme obraz bez chýb.

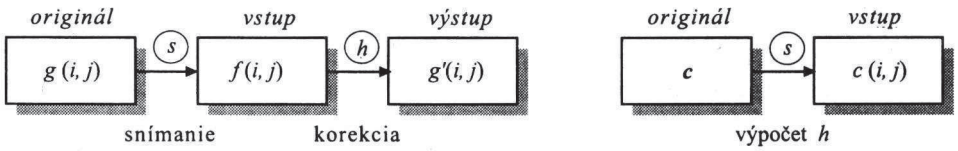
Jasová korekcia a **modifikácia obrazu pomocou histogramu** patria k najčastejším transformáciám obrazu. Uvedieme jednoduché metódy, ktoré vysvetlia obidva prístupy. Jasová korekcia väčšinou závisí len od jasu bodu vo vstupnom obraze, kým modifikácia jasovej stupnice je určená viac globálnymi charakteristikami obrazu.

Predstavu o rozdelení jasovej úrovni v digitálnom obraze získame pomocou histogramu. **Histogram** jasu je funkcia, ktorá priradí každej úrovni jasu zodpovedajúcu početnosť príslušného jasu v obraze. Histogram obrazu dáva globálnu informáciu o obraze. Histogram v spojitom prípade vyjadrujeme **funkciou hustoty rozdelenia**, ktorá je škálovaná na jednotkový integrál.

7.2 Jasová korekcia

Svetlo prechádzajúce v optických zariadeniach ďalej od optickej osi slabne, čo zapríčiňuje nejasný obraz. Ak sa tieto chyby pravidelne opakujú, potom môžeme použiť **jasovú korekciu**. Obyčajne pre každý vstupný obrazový bod $f(i, j)$ nájdeme korekčný koeficient $h(i, j)$ a výslednú opravenú jasovú úroveň zapíšeme vzťahom:

$$g(i, j) = h(i\ \ i)\ .\ f(i, j).$$



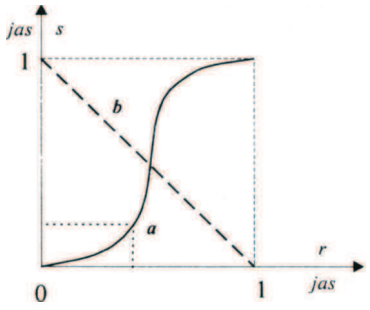
Obr. 7.1 Korekcia skreslenia vstupu

Obr. 7.1 ilustruje postup na výpočet koeficientov $h(i, j)$. Väčšinou vieme pomerne jednoducho určiť **degradačnú funkciu** $h(i, j)$. Stačí získať obraz so známym konštantným jasom c, ktorý označíme ako $c(i, j)$. Potom môžeme systematickú chybu korigovať podľa vzťahu:

$$c = h(i,j).c(i,j), \qquad g'(i,j) = h(i,j).f(i,j) = c \,.\, f(i,j) \,/\, c(i,j).$$

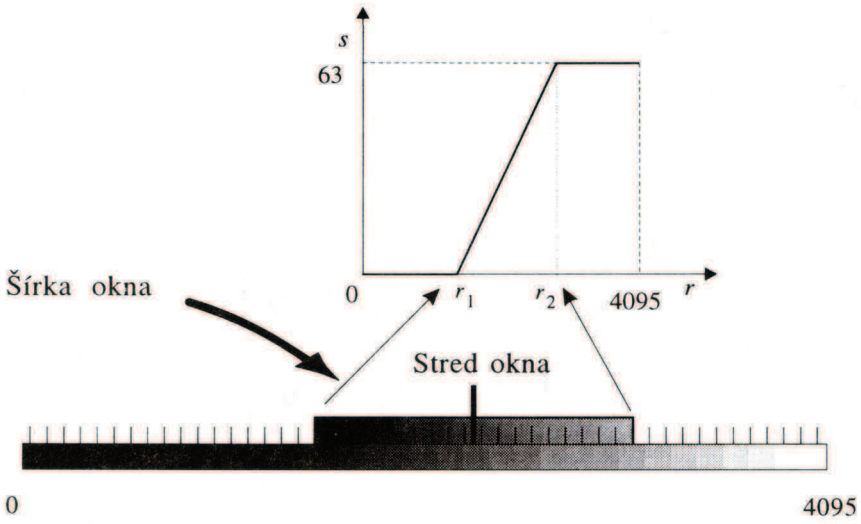
Hodnoty jasu digitálnych obrazov sú ohraničené v určitom intervale. Výpočet korigovaných hodnôt podľa predchádzajúceho vzťahu však môže dať výsledný jas mimo povolený interval. V tom prípade môžeme buď orezávať na krajné hodnoty, alebo posúvať jasovú stupnicu tak, aby sa hodnoty jasu dostali do prípustného intervalu.

Zmenu jasovej stupnice si najlepšie vysvetlíme na obrázku 7.2. Na obidvoch osiach máme vymedzený jednotkový jas. Transformačná funkcia jasu nám mení hodnoty jasu podľa funkcie. Pre lomenú čiaru *a* zvyšujeme kontrast, pretože zvýrazňujeme rozdiely medzi čiernou farbou a bielou. Z obrázku vidíme, že viacerým bodom blízkym čiernej znižujeme jas a naopak bodom blízkym bielej zvyšujeme jas. Úsečka *b* zas mení bielu farbu na čiernu a naopak, t.j. vytvárame negatívny obraz.



Obr. 7.2 Transformácie jasovej úrovne

Napríklad CT *(Computer Tomography)* a MR *(Magnetic Resonance)* obrazy využívajú 12-bitov na kódovanie jasu každého bodu, čo zodpovedá škále od 0 po 4095. Ľudské oko je schopné rozlišovať približne okolo 64 úrovní šedej farby. Je preto prirodzené využívať pre zobrazenie obrazu maximálne 64 úrovní a dať možnosť meniť interaktívne jas a kontrast. Pri zobrazovaní v medicíne sa využíva škálovanie, znázornené na obr. 7.3. Pre zobrazenie sa využíva pojem šírka okna a stred okna intenzít. Stredom a šírkou okna sú dané dve prahové hodnoty, medzi ktorými sa interval rovnomerne rozdelí na príslušný počet úrovní šedej farby. Hodnoty pod danou prahovou hodnotou $r1$ zobrazíme čiernou farbou a nad daným prahom $r2$ zobrazíme bielou.



Obr. 7.3 Zmena zobrazenia jasovej škály

Zmena jasovej úrovne môže byť použitá jednak pre zvýšenie kontrastu v obraze, ale tiež pre rovnomerné zastúpenie úrovní šedej.

7.3 Zlepšenie obrazu pomocou histogramu

Transformačný vzťah sa obvykle hľadá **metódou vyrovnávania histogramu**. Obvykle má histogram niekoľko miním a maxím. Vo výslednom vyrovnanom histograme sú jednotlivé úrovne zastúpené približne rovnako. Zvýši sa kontrast pre úrovne blízke maximu histogramu a zníži sa kontrast blízko minima histogramu.

Ukážeme si základnú myšlienku transformácie úrovní obrazu. Nech máme zadaný normalizovaný obraz svojimi úrovňami t.j. hodnoty šedej farby r ležia v intervale $\langle 0, 1 \rangle$. Čiernu farbu reprezentujeme 0 a bielu 1. Predpokladáme, že máme zadanú transformáciu

$$s = T(r), \qquad (1)$$

ktorá mení hodnoty úrovní šedej farby v obrazových bodoch. Naviac transformácia spĺňa nasledujúce vlastnosti:

1. funkcia $T(r)$ monotónne rastie alebo klesá na celom intervale,

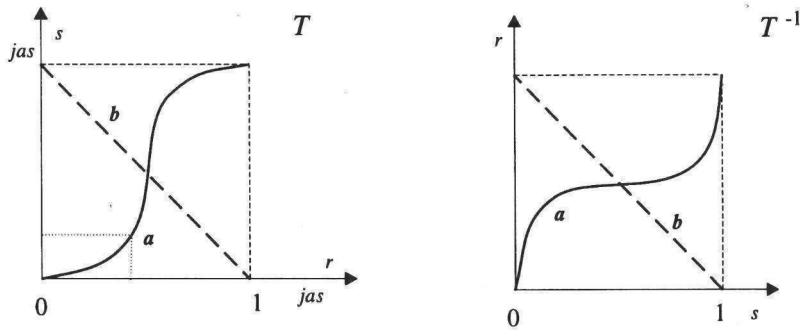
2. $0 \leq T(r) \leq 1$, pre $r \in \langle 0, 1 \rangle$.

Ak je funkcia rastúca, potom zachováva prechod v škále od čiernej po bielu. Ak je klesajúca, mapuje inverzne úrovne šedej farby, t.j. bielu farbu transformuje na čiernu a naopak bielu na čiernu.

Inverznú transformáciu označíme

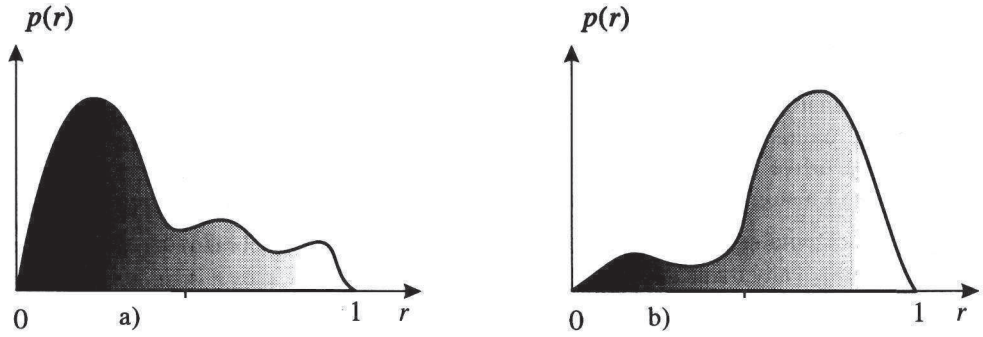
$$r = T^{-1}(s),$$

kde funkcia T^{-1} spĺňa podmienky a) a b) pre premennú s (pozri obr. 7.4).



Obr. 7.4 Transformácie úrovní jasu a ich inverzné transformácie

Predpokladajme, že funkcia hustoty rozdelenia $p_r(r)$ je na intervale $\langle 0, 1 \rangle$ spojitá. Na nasledujúcom obrázku 7.5 máme zodpovedajúce histogramy dvoch obrazov. Časť a) obr. 7.5 zobrazuje histogram pre tmavý obraz a časť b) pre svetlý obraz. Vidíme, že početnosť zastúpenia bodov vystihuje graf znázorňujúci funkciu hustoty rozdelenia.



Obr. 7.5 Funkcia hustoty rozdelenia pre tmavý a svetlý obraz

Z elementárnej teórie pravdepodobnosti platí, že ak poznáme funkciu hustoty $p_r(r)$ a transformáciu $T(r)$ s uvedenými vlastnosťami, potom inverznú funkciu hustoty rozdelenia môžeme vyjadriť vzťahom:

$$p_s(s) = \left[p_r(r) \frac{dr}{ds} \right]_{r=T^{-1}(s)}, \qquad (2)$$

čo môžeme vysvetliť ako súčin pôvodnej hustoty rozdelenia s deriváciou inverznej funkcie $r = T^{-1}(s)$ v konkrétnom bode r.

7.3.1 Vyrovnávanie histogramu

Zostrojíme nasledujúcu funkciu pomocou funkcie rozdelenia

$$s = T(r) = \int_0^r p_r(v)\, dv, \quad 0 \le r \le 1. \qquad (3)$$

Tento integrál sa nazýva **kumulatívny histogram**. Pretože funkcia p_r je kladná, integrál na pravej strane rovnice je monotónna funkcia. Deriváciou integrálu (3) dostaneme:

$$\frac{ds}{dr} = p_r(r).$$

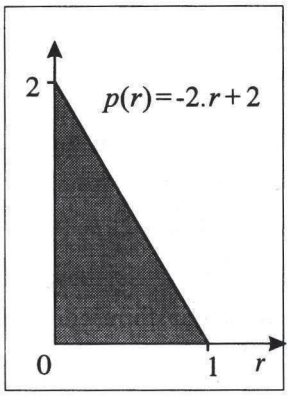
Po substitúciou *dr/ds* do rovnice (2) platí

$$p_s(s) = \left[p_r(r)\frac{1}{p_r(r)}\right]_{r=T^{-1}(s)} = [1]_{r=T^{-1}(s)} = 1, \qquad 0 \le s \le 1,$$

čo predstavuje uniformnú hustotu. Z hľadiska spracovania obrazu sme touto zmenou dosiahli rovnomerné rozloženie úrovní šedej farby v obraze.

Príklad 1: Najprv ukážeme jednoduché použitie spojitého prípadu (2) a (3). Predpokladajme, že máme lineárnu funkciu hustoty:

$$p_r(r) = -2.r + 2 \qquad 0 \le r \le 1.$$



Dosadením p_r do rovnice (3) vypočítame integrál:

$$s = T(r) = \int_0^r (-2v + 2)\, dv = -r^2 + 2.r\,.$$

Hľadáme inverznú funkciu. Zistíme, že rovnici vyhovuje jediné korektné riešenie:

$$r = T^{-1}(s) = 1 - \sqrt{1-s}\,.$$

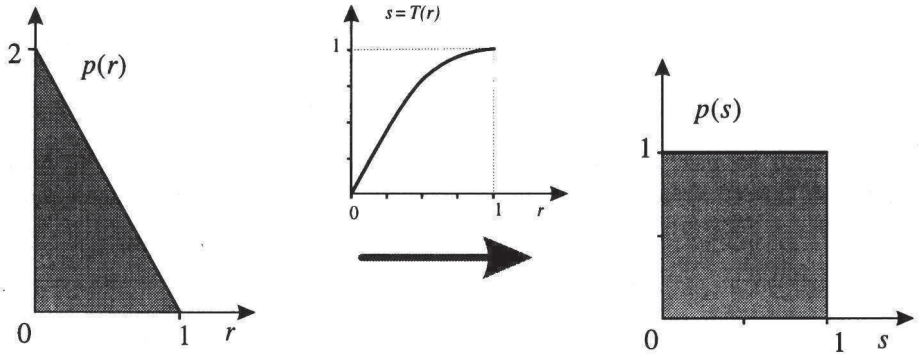
Pre funkciu hustoty závislej od *s* dostaneme úpravou nasledujúci vzťah:

$$p_s(s) = \left[p_r(r)\frac{dr}{ds}\right]_{r=T^{-1}(s)} = \left[(-2r+2)\frac{dr}{ds}\right]_{r=1-\sqrt{1-s}} =$$

$$\left((2\sqrt{1-s})\frac{d}{ds}(1-\sqrt{1-s})\right) = 1,$$

čo je uniformná hustota rozloženia.

Na obr. 7.6 je zobrazená transformačná funkcia $T(r)$.



Obr. 7.6 Ilustrácia uniformnej hustoty

V diskrétnom prípade funkciu hustoty nahradíme pravdepodobnosťou

$$p_r(r_k) = n_k / n, \quad 0 \le r_k \le 1, \quad k = 0, 1, ..., L\text{-}1,$$

kde L je počet úrovní šedej v obraze a $p_r(r_k)$ je pravdepodobnosť k-úrovne, n_k je počet výskytov tejto úrovne.

Pre vyrovnávanie histogramu nahradíme prirodzene integrál súčtom, a preto platí:

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j) = \sum_{j=0}^{k} \frac{n_j}{n}, \quad 0 \le r_k \le 1, \quad k = 0, 1, ..., L\text{-}1. \quad (4)$$

Inverznú transformáciu označíme:

$$r_k = T^{-1}(s_k), \qquad 0 \le s_k \le 1,$$

Všimnime si, že transformačnú funkciu $T(r_k)$ počítame podľa vzorca (4) pre pravdepodobnosť. V prípade, že úrovne jasu nemáme normalizované a vychádzame z početnosti, potom vyrovnávanie histogramom uskutočníme nasledujúcim súčtom:

$$s = T(r) = \frac{q_k - q_0}{n^2} \sum_{i=p_0}^{p} h(i) + q_0,$$

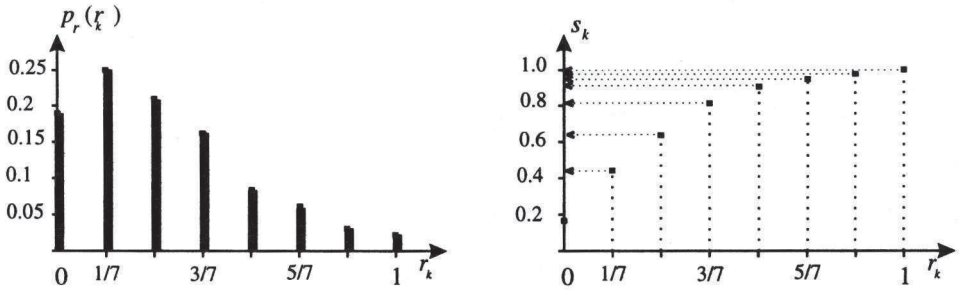
kde interval jasu pre vstupný obraz je $\langle p_0, p_k \rangle$, $h(p)$ je jeho početnosť (histogram) a výstupný interval jasu je $\langle q_0, q_k \rangle$.

Príklad 2: Predpokladajme, že máme obraz s rozmermi 64×64 (4096 pixlov) a 8 úrovní šedej farby. V tabuľke 7.1 máme zistené rozloženie jednotlivých úrovní a na obr. 7.7 zobrazenú zodpovedajúcu pravdepodobnosť. Hľadáme obraz s vyrovnaným histogramom.

Tabuľka 7.1

r_k	n_k	$p_k(r_k)=n_k/n$
$r_0=0$	790	0.19
$r_1=1/7$	1023	0.25
$r_2=2/7$	850	0.21
$r_3=3/7$	656	0.16

r_k	n_k	$p_k(r_k)=n_k/n$
$r_4=4/7$	329	0.08
$r_5=5/7$	245	0.06
$r_6=6/7$	122	0.03
$r_7=1$	81	0.02



a) b)

Obr. 7.7 Vyrovnávanie histogramu a) originál, b) transformačná funkcia

Transformačnú funkciu vypočítame pomocou (4). Pre s_0 platí:

$$s_0 = T(r_0) = \sum_{j=0}^{0} p_r(r_j) = p_r(r_0) = 0.19 .$$

Podobne pre s_1 :

$$s_1 = T(r_1) = \sum_{j=0}^{1} p_r(r_j) = p_r(r_0) + p_r(r_1) = 0.44$$

a $s_2 = 0.65$, $s_3 = 0.81$, $s_4 = 0.89$,
$s_5 = 0.95$, $s_6 = 0.98$, $s_7 = 1.00$.

Pretože máme 8 úrovní, musíme nájsť najbližšie hodnoty tohto rastra pre s_0 až s_7:

$s_0 \cong 1/7$, $s_1 \cong 3/7$, $s_2 \cong 5/7$, $s_3 \cong 6/7$,
$s_4 \cong 6/7$, $s_5 \cong 1$, $s_6 \cong 1$, $s_7 \cong 1$.

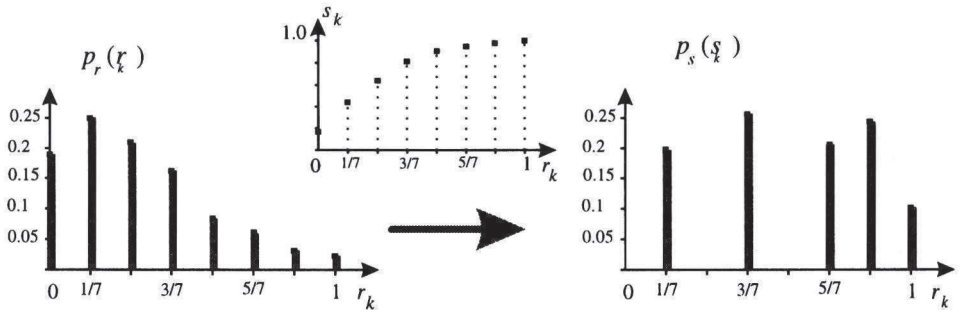
Všimnime si, že máme len 5 rôznych úrovní, ktoré označíme za sebou:

$s_0 = 1/7$, $s_1 = 3/7$, $s_2 = 5/7$, $s_3 = 6/7$, $s_4 = 1$.

Pre novovytvorené hodnoty s zistíme histogram, ktorý je uvedený v nasledujúcej tabuľke 7.2 a zobrazený na obr. 7.8

Tabuľka 7.2

hodn. r	$T(r)$	hodn. s	poč. pixlov	nový počet	histogram
$r_0 = 0$	0.19	$s_0 = 0.14$	790	790	0.19
$r_1 = 1/7$	0.44	$s_1 = 0.43$	1 023	1 023	0.25
$r_2 = 2/7$	0.65	$s_2 = 0.71$	850	850	0.21
$r_3 = 3/7$	0.81	$s_3 = 0.85$	656		
$r_4 = 4/7$	0.89	$s_4 = 0.85$	329	985	0.24
$r_5 = 5/7$	0.95	$s_5 = 1.0$	245		
$r_6 = 6/7$	0.98	$s_6 = 1.0$	122		
$r_7 = 7/7$	1	$s_7 = 1.0$	81	448	0.11



Obr. 7.8 Ilustrácia vyrovnania histogramu

7.3.2 Priama špecifikácia histogramu

Tento spôsob je užitočný hlavne vtedy, keď vieme dopredu, ako máme zmeniť histogram obrazu, aby sme dostali lepší výsledný obraz. Predchádzajúcou technikou vyrovnania histogramu nemáme možnosť ovplyvniť výsledný obraz parametrom, a preto sa zdá táto technika málo interaktívna.

Na jednoduchšie vysvetlenie tohto prístupu uvedieme spojitý prípad. Nech máme zadaný obraz a poznáme pravdepodobnostnú hustotu obrazu $p_r(r)$ a chceme ju zmeniť

na hustotu $p_z(z)$. Potom pre zmenu obrazu použijeme kumulatívny histogram, ktorým budeme vyrovnávať obraz. Preto najprv pre funkcie určíme tieto závislosti:

$$s = T(r) = \int_0^r p_r(v)dv, \qquad (5)$$

$$t = G(z) = \int_0^r p_z(v)dv. \qquad (6)$$

Pretože vyrovnanie histogramu dáva rovnaký výsledný jednotkový kumulatívny histogram, môžeme premennú t brať ako s. Pre inverznú funkciu zo vzťahu (6) platí:

$$z = G^{-1}(s),$$

Dosadením (5) do poslednej rovnice dostaneme vyjadrenie pre požadovanú transformáciu:

$$z = G^{-1}(T(r)),$$

a tým aj špecifikovaný histogram.

7.4 Filtrácia

Metódy spracovania obrazu sú dôležité z hľadiska ďalšieho použitia obrazových dát pre iné aplikácie. Cieľom týchto metód je potlačiť šum vzniknutý pri snímaní a prenose obrazu, odstrániť ostré prechody a podľa potreby potlačiť alebo zvýrazniť určité črty obrazu.

Susedné body v obraze majú často blízku hodnotu jasu. Obraz má ostré zmeny na hranách objektov, ktoré sú obyčajne zastúpené v dvojrozmernom obraze len čiarovo, a preto rádovo zriedkavejšie. Pokiaľ teda vieme nájsť pixel poškodený náhodným šumom, potom môžeme jeho hodnotu opraviť napríklad jednoduchým priemerovaním hodnôt bodov z jeho okolia.

Pre modifikáciu obrazu sa využívajú hlavne dva všeobecné prístupy, ktoré sa delia na **priestorové** a **frekvenčné**, preto v prvej časti uvedieme tieto prístupy. Podľa spracovávania frekvencie obrazovej funkcie môžeme rozdeliť takéto prístupy na dve skupiny: vyhladenie a ostrenie.

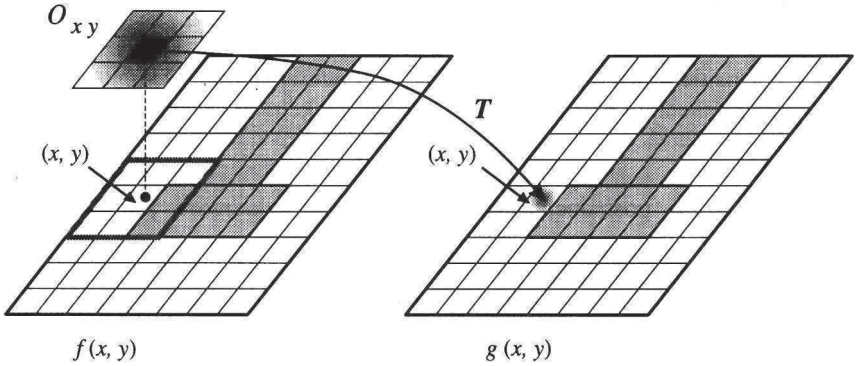
Vyhladenie obrazu vedie k potlačeniu vyšších frekvencií a tým je potlačený náhodný šum. Súčasne však dochádza k potlačeniu náhlych zmien jasovej funkcie a tým k rozostreniu hrán. Naopak **ostrenie obrazu** alebo **gradientné operácie** vedú k zdôrazneniu vyšších frekvencií a tým zvýrazňujeme náhle zmeny pri hranách, ale žiaľ aj pri šumoch. Z uvedeného popisu vyplýva, že vyhladenie a ostrenie obrazu sú protichodné operácie.

7.4.1 Priestorové prístupy

Tieto prístupy zmeny obrazu pracujú priamo v časti digitálnej roviny tak, že sa transformuje obraz pomocou operátora v nasledujúcej forme:

$$g(x,y) = T[f(x,y)],$$

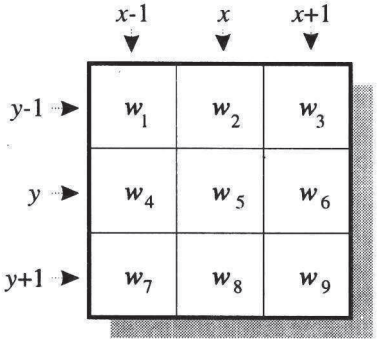
kde $f(x, y)$ je vstupný obraz a $g(x, y)$ je výstupný obraz. Operátor T najčastejšie vyjadrujeme pomocou funkcie hodnôt okolia bodu. Ukážeme si jednoduchý príklad vyjadrenia pomocou **masky okolia**. Na obrázku 7.9 je zobrazené okolie O rozmerov 3×3. Pre výpočet hodnoty g v bode (x, y) využijeme funkčné hodnoty posunutého okolia O_{xy}. Stred okolia presúvame do každého bodu a pomocou tejto masky vyjadríme operátor v ľubovoľnom bode.



Obr. 7.9 Posun masky okolia O do bodu (x, y) pre určenie g(x, y)

Všeobecný prípad lineárnej závislosti operátora T sa dá vyjadriť na príklade masky 3×3. Ako ukazuje obr. 7.10, pre **koeficienty masky** w_1, w_2,..., w_9 8-okolia bodu (x, y), môžeme definovať nasledujúci lineárny operátor:

$$T[f(x, y)] = w_1.f(x-1, y-1) + w_2.f(x, y-1) + w_3.f(x+1, y-1) +$$
$$w_4.f(x-1, y) + w_5.f(x, y) + w_6.f(x+1, y) +$$
$$w_7.f(x-1, y+1) + w_8.f(x, y+1) + w_9.f(x+1, y+1). \quad (7)$$



Obr. 7.10 Všeobecný prípad masky na okolí 3x3

V prípade, že si zvolíme koeficienty $w_i = 1/9$, $i = 1, 2, ..., 9$, a $g(x, y) = T[f(x, y)]$, potom novú hodnotu bodu $g(x, y)$ určíme priemerovaním hodnôt 8-okolia bodu $f(x, y)$. Neskôr sa k tomuto príkladu operátora vrátime.

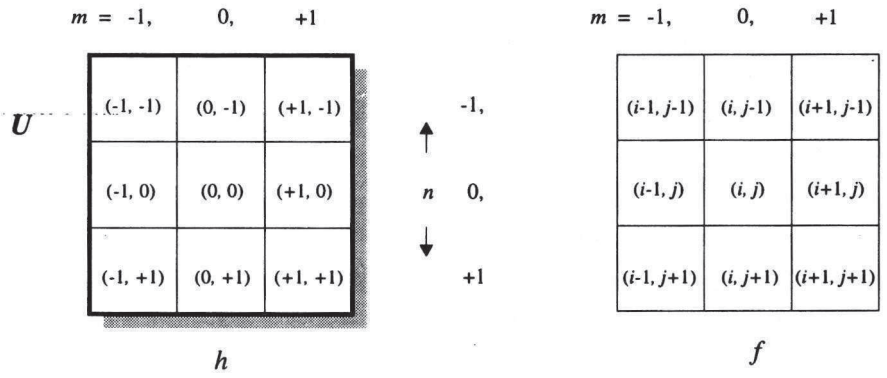
7.4.2 Lokálne spracovanie ako konvolúcia

V tejto časti sa budeme zaoberať takými transformáciami obrazu, ktoré určujú zmenu vo výstupnom obraze na základe lokálneho okolia spracovávaného bodu. Podľa fukčného vzťahu pre výpočet jasu výsledného obrazu delíme metódy na **lineárne** a **nelineárne**.

Definujeme masky ľubovoľných rozmerov. Zväčšíme okolie pre operátor T na $n \times m$ bodov. Pre lineárnu závislosť si zavedieme masky definované na tomto okolí, ktoré určujú výsledný jas ako lineárnu kombináciu jasu v okolí vstupného obrazu f s váhovými koeficientami h:

$$g(i,j) = \sum_{(m,n)\in O} \sum h(i-m, j-n) \cdot f(m,n) = \sum_{(m,n)\in U} \sum h(m,n) \cdot f(i-m, j-n). \qquad (8)$$

Tento vzťah môžeme interpretovať ako diskrétnu konvolúciu s **jadrom** h. Voľba veľkosti lokálneho okolia spracovávaného bodu závisí od veľkosti objektu v obraze. Častokrát kvôli pravidelnosti uvažujeme, že okolie O funkcie f je štvorcové okolie so stredom v bode (i, j), alebo ho vyjadríme pomocou okolia U funkcie h, kde za krajné hodnoty m a n sa berú obvykle hodnoty od -1 do 1 alebo od -2 do 2, ako je to znázornené na obr. 7.11. Porovnaním vzťahov (7) a (8) vidíme, že lineárne operátory sa dajú vyjadriť pomocou konvolučných jadier.



Obr.7.11 Okolie U pre vyjadrenie lineárnej kombinácie masky h a funkcie f

7.4.3 Frekvenčné prístupy

Základom frekvenčnej techniky je konvolučná veta, ktorú sme uviedli v 6. kapitole. Pripomenieme, že ak $g(x, y)$ je obraz vytvorený pomocou konvolúcie obrazu $f(x, y)$ a invariatného jadra $h(x, y)$, potom podľa konvolučnej vety platí nasledujúca frekvenčná závislosť:

$$G(u, v) = H(u, v).F(u, v), \qquad (9)$$

kde G, H a F sú Fourierove transformácie funkcií g, h, a f. Ako neskôr ukážeme, funkciu obrazu $f(x, y)$ zmeníme výberom funkcie $H(u, v)$ a zmenu vypočítame podľa konvolučnej vety s využitím inverznej Fourierovej transformácie:

$$g(x, y) = F^{-1}\big(H(u, v).F(u, v)\big).$$

7.5 Vyhladenie obrazu

Často požadujeme vo vnútri oblasti potlačenie rozdielu jasu, ktorý zapríčiňuje šum. Ak objekty nášho záujmu sú v porovnaní s oblasťami šumu neporovnateľne väčšie, potom je možné šum v obraze odstrániť metódami založenými na priemerovaní hodnôt z okolia alebo výberom mediánu strednej hodnoty. Ak rozdiely vo veľkosti objektu a šumu sú pomerne malé, potom je dobré nájsť transformáciu, ktorá v obraze nájde a odstráni relatívne veľké diferencie jasu.

7.5.1 Priemerovanie

Pokiaľ máme m nezávislých obrazov rovnakej predlohy, potom je výhodné filtrovať šum bez rozmazávania hrán počítaním priemeru jasu cez rovnaké body:

$$g(i,j) = \frac{1}{m} \sum_{l=1}^{m} f_l(i,j).$$

Ak je k dispozícii len jeden obraz, potom filtrujeme lokálnym priemerovaním cez okolie O bodu (x, y):

$$g(x,y) = \frac{1}{M} \sum_{(i,j)\in O} f(i,j),$$

kde M je počet bodov okolia O.

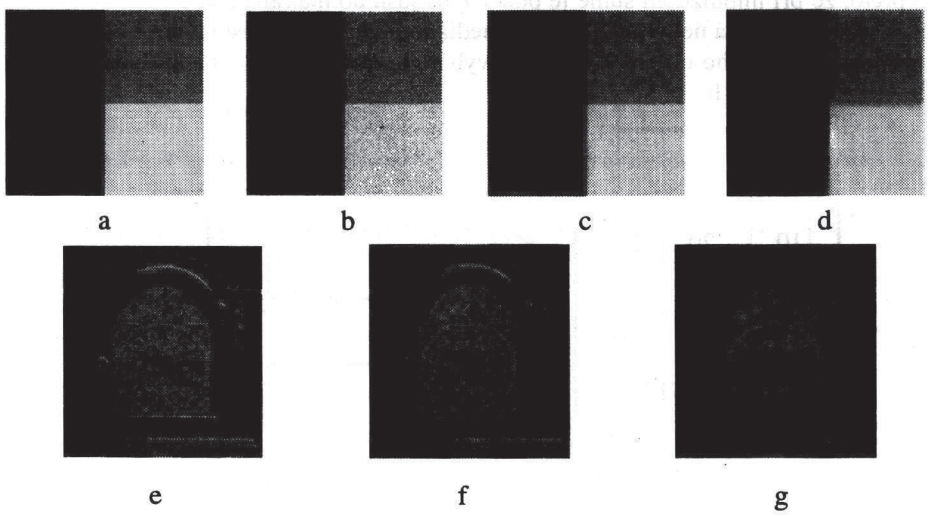
Filtrácia obyčajným priemerovaním je špeciálnym prípadom diskrétnej konvolúcie. Pre okolie 3×3 je **konvolučná maska** definovaná:

$$h = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Niekedy zväčšujeme váhu stredového bodu masky (príp. aj jeho štyroch susedov), aby sme sa viac blížili k vlastnostiam **Gaussovského rozdelenia**:

$$h = \frac{1}{10}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \qquad h = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}.$$

Na obrázku 7.12 je ukážka vlastností filtračných operátorov priemerovaním. Časti a) a e) tohto obrázka sú originálne obrázy, k časti b) obrázka sme pridali šum a obrázky c) a f) (resp. d) a g) sme filtrovali okolím 5×5 (resp. 9×9).



Obr. 7.12 Ukážka aplikácie filtrovania priemerovaním

Základnou nevýhodou využitia obyčajného priemerovania je rozmazávanie hrán v obraze. Preto sa táto metóda používa ako pomocná metóda pre výpočet strednej hodnoty jasu alebo rozptylu v danom bode.

Zníženie rozmazania hrán sa dá zabezpečiť nasledujúcim vzťahom:

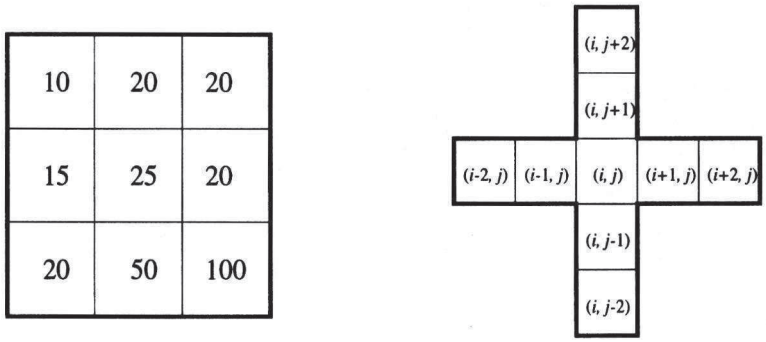
$$g(x,y) = \begin{cases} \frac{1}{M} \sum_{(m,n)\in O} f(m,n), & \text{ak } \left| f(x,y) - \frac{1}{M} \sum_{(m,n)\in O} f(m,n) \right| < T \\ f(x,y), & \text{inak,} \end{cases}$$

kde T je špecifikovaná prahová hodnota a O je okolie bodu (x, y). V prípade, že rozdiel priemerovanej hodnoty od pôvodnej presiahne prahovú hodnotu, potom sa pôvodná hodnota nezmení.

7.5.2 Medián

Často sa používa **filtrácia metódou mediánu**. Medián je veličina používaná v teórii náhodnej premennej. Výpočet mediánu je pre diskrétnu obrazovú funkciu jednoduchý. Stačí usporiadať hodnoty jasu v lokálnom okolí a medián je prvok, ktorý sa nachádza uprostred.

Metodu si ukážeme na jednoduchom príklade okolia 3×3. Napríklad máme okolie z obrázku 7.13 a). Po usporiadaní deviatich hodnôt z okolia dostaneme množinu (10, 15, 20, 20, **20**, 20, 25, 50, 100) a v nej je **mediánový prvok** prostredný, 5. v poradí. Výhoda tejto filtrácie je, že redukuje stupeň rozmazania hrán a dobre potláča **impulzný šum**. To preto, že pri impulznom šume je buď chyba jasu do maxima alebo minima niektorej šedej úrovne. Hlavná nevýhoda filtrácie mediánom je tá, že porušuje tenké čiary, a preto miesto obdĺžnikového okolia sa niekedy vyberá upravené okolie, napríklad ako je znázornené na obr.7.13 b).



a) b)

Obr.7.13 Upravené okolie pre filtráciu metódou mediánu

7.5.3 Nízkofrekvenčné filtrovanie (low-pass filtering)

Podľa konvolučnej vety (3) platí nasledujúca frekvenčná závislosť:

$$G(u, v) = H(u, v).F(u, v),$$

kde G, H a F sú Fourierové transformácie pre g, h, a f. Funkciu $f(x, y)$ budeme filtrovať **transformačnou funkciou procesu** $H(u, v)$ a filtráciu počítať podľa konvolučnej vety:

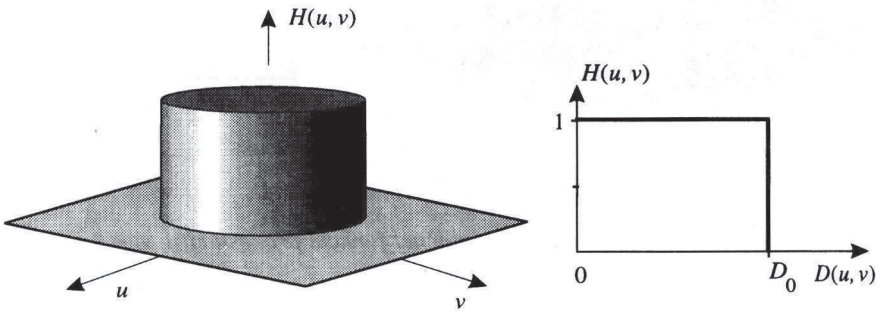
$$g(x, y) = F^{-1}\big(H(u, v).F(u, v)\big).$$

Ideálny filter

V tomto prípade volíme funkciu H nasledujúcim predpisom:

$$H(u,v) = \begin{cases} 1, & \text{ak } D(u,v) \le D_0 \\ 0, & \text{inak} \end{cases}, \qquad (10)$$

kde $D(u, v) = \sqrt{u^2 + v^2}$ je euklidovská vzdialenosť a D_0 je prahová hodnota pre orezanie vyšších frekvencií. Na obr. 7.14 máme znázornenú funkciu $H(u, v)$ a jej rez v rovine daný len jednou premennou.



Obr. 7.14 Zobrazenie transformačnej funkcie procesu H(u, v) a jej rez

Nasledujúci obr. 7.15 ukazuje použitie ideálneho nízkofrekvenčného filtrovania s rôznym polomerom D_0 (D_0 je 30% , 20% a 10% s dĺžky strany obrazu).

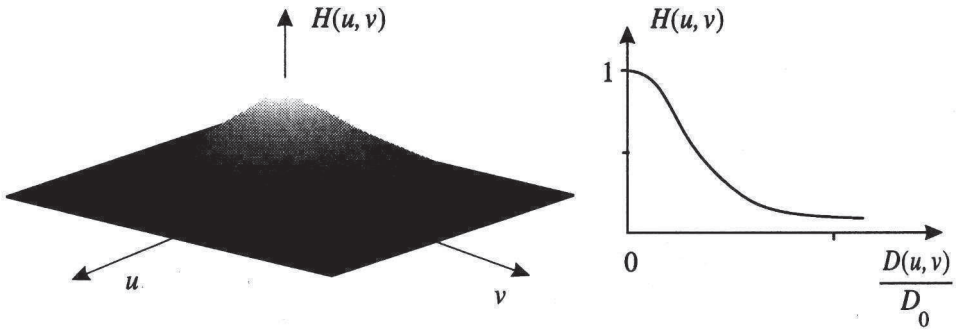
Obr. 7.15 Nízkofrekvenčné filtrovanie na obraz Leny

Filter Butterworth

Aby bola funkcia H spojitá, definujeme ju nasledujúcim predpisom:

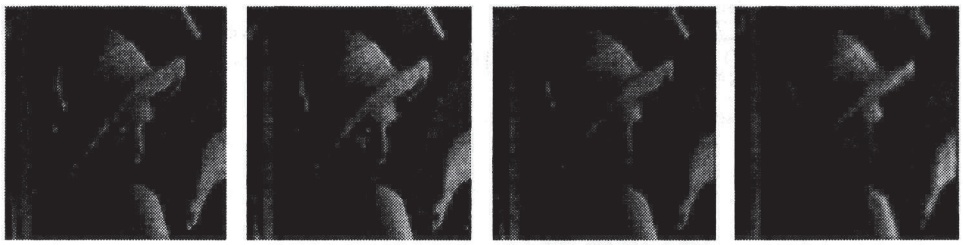
$$H(u, v) = \frac{1}{1+(D(u,v)/D_0)^{2n}},$$

kde parametre $D(u, v)$ a D_0 majú rovnaký význam ako pri ideálnom filtri. Obr. 7.16 znázorňuje funkciu $H(u, v)$ a jej rez v rovine daný len jednou premennou.



Obr. 7.16 Zobrazenie transformačnej funkcie procesu H(u, v) a jej rez

Nasledujúci obr. 7.17 ukazuje použitie Butterworth nízkofrekvenčného filtrovania s rôznym polomerom D_0. Exponent n sme zvolili rovný 1.



Obr. 7.17 Nízkofrekvenčné filtrovanie na obraz Leny

7.6 Ostrenie obrazu

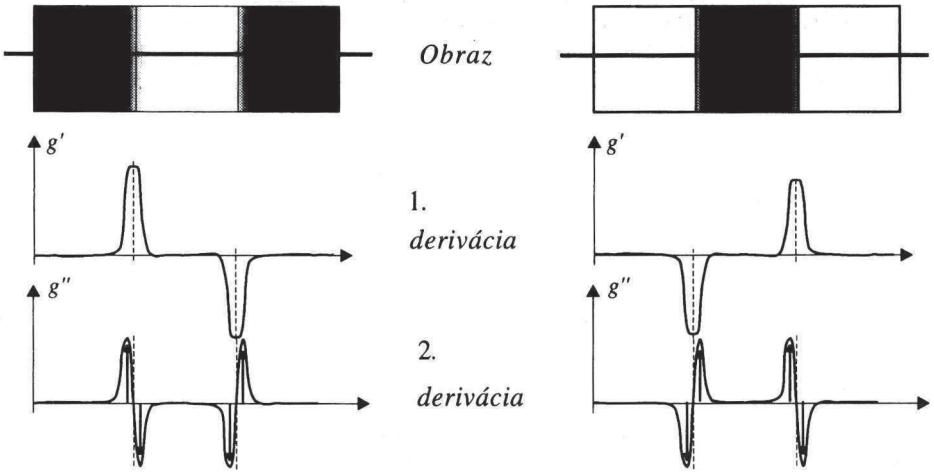
Hranu v obraze chápeme ako množinu bodov obrazu, v ktorých susedstve sa veľmi mení jas. Hrana je vlastnosť obrazového bodu a jeho okolia. Naším cieľom bude jednak určiť množinu bodov s vlastnosťou hrany v obraze a taktiež pre daný vstupný obraz vytvoriť výstupný obraz, v ktorom sú hrany zvýraznené. Cieľom ostrenia obrazu je zvýrazniť hrany. To zodpovedá zvýrazneniu vysokých frekvencií vo Fourierovom spektre. Hranové operátory môžeme rozdeliť do dvoch skupín:

1. Operátory, ktoré odhadujú prvú deriváciu. Smer gradientu sa odhaduje hľadaním masky, ktorá prislúcha najväčšej veľkosti gradientu. Prípadne realizácia diskrétnej konvolúcie.

2. Druhá skupina operátorov je založená na hľadaní hrán v miestach, kde druhá derivácia obrazovej funkcie prechádza nulou.

Na hranu sa môžeme pozerať aj ako na vektorovú veličinu, pretože je určená veľkosťou a smerom. Zložitejší problém je určiť obrys ako usporiadanú množinu susedných hranových bodov.

Nasledujúci obrázok 7.18 schématicky znázorňuje situáciu pre spojitý prípad.



Obr. 7.18 Ilustrácia prvej a druhej derivácie spojitej funkcie na obraze

7.6.1 Ostrenie pomocou gradientu

Body hrany sa dajú určovať pomocou **gradientu**, ktorý je definovaný pre spojité funkcie a dá sa jednoducho upraviť aj pre digitálny obraz. V spojitom prípade definujeme gradient funkcie $f(x, y)$ pomocou parciálnych derivácií ako vektor:

$$\mathbf{G}(f(x, y)) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right).$$

Z gradientu môžeme vyčítať dôležité vlastnosti obrazu:

1. smer, v ktorom dosahuje funkcia $f(x, y)$ maximálne zväčšenie,

2. veľkosť maximálnej miery zväčšenia funkcie $f(x, y)$ na jednotkovú dĺžku.

Veľkosť gradientu počítame ako dĺžku vektora:

$$G[f(x, y)] = \text{mag}[\mathbf{G}] = \sqrt{(\partial f/\partial x)^2 + (\partial f/\partial y)^2} .$$

Upozorňujeme na rozdiel označenia vektora $\mathbf{G}(f(x, y))$ a dĺžky vektora $G[f(x, y)]$. Druhou zložkou vyjadrenia gradientu v polárnych súradniciach je **smer** φ, ktorý vypočítame pomocou inverznej funkcie tangens:

$$\varphi = \tan^{-1}\left(\frac{\partial f/\partial y}{\partial f/\partial x} \right).$$

V digitálnom obraze aproximujeme parciálne derivácie diferenciami d_x a d_y:

$d_x f(i,j) = f(i,j) - f(i-1,j)$ alebo $d_x f(i,j) = f(i+1,j) - f(i,j)$,

$d_y f(i,j) = f(i,j) - f(i,j-1)$ alebo $d_y f(i,j) = f(i,j+1) - f(i,j)$.

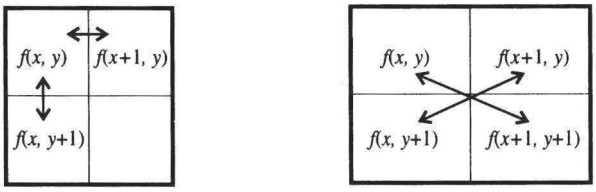
Preto pre digitálny obraz vyjadríme veľkosť v **Euklidovskej metrike**:

$$G[f(x,y)] = \sqrt{(f(x+1,y) - f(x,y))^2 + (f(x,y+1) - f(x,y))^2} ,$$

alebo jednoduchšie pomocou absolútnej hodnoty (v **manhattanskej metrike**):

$$G[f(x,y)] = |f(x+1,y) - f(x,y)| + |f(x,y+1) - f(x,y)|.$$

Pomocou veľkosti gradientu definujeme nový obraz tak, že každému bodu priradíme hodnotu gradientu. Vzťahy pri vyjadrení veľkosti gradientu sú znázornené symbolicky na obr. 7.19 a).



a) b)

Obr. 7.19 Rozdiely bodov pre diskrétny gradient a Robertsov operátor

Pretože z okolia 2×2 využívame len tri body, často sa pre ostrenie obrazu využíva iná aproximácia, znázornená symbolicky na obr. 7.19 b) a využívajúca derivácie aj v diagonálnom smere. Takýto operátor sa nazýva **Robertsov** a jeho aproximáciu vyjadríme nasledovne:

$$G[f(x,y)] = \sqrt{(f(x,y) - f(x+1,y+1))^2 + (f(x,y+1) - f(x+1,y))^2}$$

alebo výpočtovo menej náročnejšou aproximáciou (absolútnou hodnotou)

$$G[f(x,y)] = |f(x,y) - f(x+1,y+1)| + |f(x,y+1) - f(x+1,y)|. \qquad (12)$$

7.6.2 Ostrenie pomocou konvolúcie

Prvá skupina metód ostrenia (t.j. metódy využívajúce 1. deriváciu) sú založené na diskrétnej konvolúcii, a preto budeme uvádzať len konvolučné jadrá. Jedna z najstarších metód je vyššie spomenutý Robertsov operátor, ktorý pracuje na najmenšom okolí 2×2. Jeho konvolučné masky sú

$$h_1 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad h_2 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

a veľkosť gradientu sa počíta podľa vzorca (5). Výpočet využíva len veľmi malé okolie a preto je dosť citlivý na šum.

Príkladom lepšej aproximácie gradientu je **Sobelov operátor**, ktorý aproximuje prvú parciálnu deriváciu. Sobelov operátor je možné vytvoriť pre rôzne veľkosti a smery. Napríklad pre rozmer 3x3 existuje osem konvolučných masiek, ktoré dostávame rotáciou prvej:

$$h_1 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}, h_2 = \begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}, h_3 = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \ldots$$

Presnejšie týchto osem konvolučných jadier definuje nasledujúci operátor:

$$S[f(x, y)] = h_1.f + h_2.f + \ldots + h_8.f = |f(x\text{-}1, y\text{+}1) + 2.f(x, y\text{+}1) + f(x\text{+}1, y\text{+}1) - f(x\text{-}1, y\text{-}1) - 2.f(x, y\text{-}1) - f(x\text{+}1, y\text{-}1)| + \ldots.$$

Pričom masky h_1 a h_3 majú dôležitý význam, pretože sa využívajú pre parciálne derivácie:

$h_1 = D_x$ približuje parciálnu deriváciu v smere x, (11)

$h_3 = D_y$ približuje parciálnu deriváciu v smere y.

Laplaceov gradientný operátor (*Laplacián*) aproximuje druhú deriváciu. Udáva veľkosť hrany a nie jej smer, a preto nie je citlivý na otočenie. V digitálnom obraze je tiež aproximovaný diskrétnou konvolúciou. Podľa typu susednosti (4-susednosť alebo 8-susednosť) rozoznávame dve konvolučné jadrá:

$$h^4 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \quad h^8 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

Tieto masky generujú nasledujúce operátory:

$$L^4[f(x, y)] = h^4.f = f(x, y\text{-}1) + f(x, y\text{+}1) + f(x\text{+}1, y) + f(x\text{-}1, y) - 4.f(x, y),$$

$$L[f(x, y)] = h^8.f = L^4[f(x,y)] + f(x\text{-}1, y\text{-}1) + f(x\text{-}1, y\text{+}1) + f(x\text{+}1, y\text{-}1) + f(x\text{+}1, y\text{+}1) - 4f(x, y).$$

7.6.3 Vysokofrekvenčné filtrovanie (high-pass filtering)

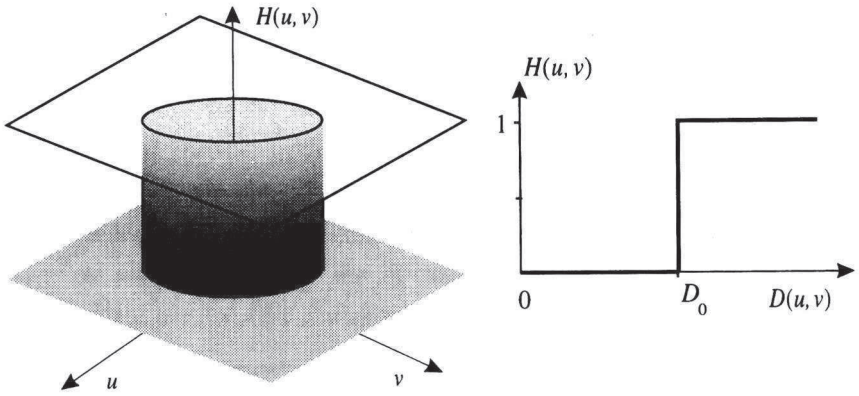
Podobne ako pre vyhladenie obrazu aj pre ostrenie môžeme využiť konvolučnú vetu. V tom prípade volíme pre transformačnú funkciu procesu $H(u, v)$ orezanie nižších frekvencií, čo zabezpečíme jednoducho voľbou:

$$H(u, v) = \begin{cases} 0, & \text{ak } D(u, v) \le D_0 \\ 1, & \text{inak} \end{cases},$$

kde parametre D a D_0 majú rovnaký význam ako v (4). Podobne môžeme zvoliť spojitú funkciu H nasledujúcim predpisom:

$$H(u,v) = \frac{1}{1 + (D_0/D(u,v))^{2n}}.$$

Na nasledujúcom obr.7.20 máme znázornenú funkciu pre ideálny filter.



Obr. 7.20 Zobrazenie transformačnej funkcii procesu H(u, v) a jej rez

Na nasledujúcom obrázku 7.21 je použitý ideálny vysokofrekvenčný filter na ostrenie obrazu.

Obr. 7.21 Ostrenie obrazu pomocou vysokofrekvenčného filtru

8

Segmentácia a hranica obrazu

8.1 Úvod

Segmentácia je metóda, ktorá rozdelí obraz do častí podľa súvislostí objektov či predmetov v obraze. Pri automatickom vyhľadávaní objektov máme často v obraze pozadie definované konštantným jasom a objekty sú výrazne odlíšené od pozadia (napríklad pri spracovávaní textu alebo manipulácii robota na linke, prípadne spracovávanie mikroskopických obrazov). Pri spracovávaní zložitejších obrazov ako sú obrazy z družíc, sond alebo snímanie reálneho sveta kamerou, nemusí byť také jednoduché rozdeliť obraz na objekty a pozadie.

Pri segmentácii je rozdelenie objektov ovplyvnené nejednoznačnosťou obrazových údajov (napr. šumom alebo nepodstatnými objektami). Vtedy je vhodné získať aspoň čiastočnú segmentáciu. Podľa typu použitej techniky môžeme rozdeliť segmentáciu na:

- **globálnu** (využívajú sa globálne vlastnosti obrazu)

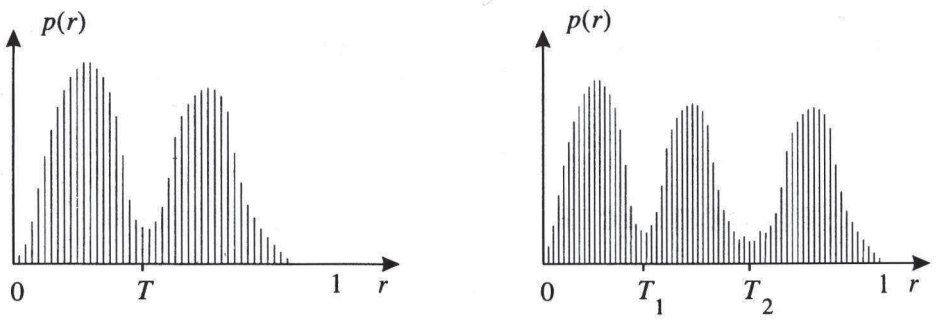
- **lokálnu** (využívajú sa lokálne vlastnosti oblasti)

V časti 8.3 uvedieme vyhľadávanie hranice a obrysu pre binárne obrazy, ktoré môžeme dostať po segmentácii. V poslednej časti ukážeme postup vytvorenia segmentácie pomocou obrysu.

8.2 Segmentácia prahovaním

Prahovanie je jeden z dôležitých prístupov k členeniu obrazu. V tejto časti uvedieme niekoľko spôsobov prahovania a poukážeme na podstatu problému.

Predpokladajme, že histogram obrazu $f(x, y)$, ktorý je zobrazený na obr. 8.1 a), je zložený zo svetlého objektu a tmavého pozadia. Jednoduchá cesta odčlenenia objektu od pozadia je výber prahovej hodnoty T, ktorá oddelí dve maximá histogramu. Potom bod (x, y), pre ktorý funkcia $f(x, y) > T$, je **bod objektu**. Ostatné body nazývame **bodmi pozadia**. O trochu všeobecnejší prístup približuje aj riešenie pre histogram zobrazený na obr. 8.1 b). V tomto prípade obraz histogramu je charakterizovaný troma prevládajúcimi modmi (napr. dva typy svetlých predmetov na tmavom pozadí). Môžeme použiť ten istý prístup. Bod (x, y) bude patriť prvému objektu, keď $T1 < f(x, y) \leq T2$, k druhému objektu, keď $f(x, y) > T2$. Nakoniec pozadie je určené pre $f(x, y) \leq T1$.



a) b)

Obr. 8.1 Bimodálny histogram a trimodálny histogram pre rôzne obrazy

Nájsť pravidlo či je histogram bimodálny, nie je také jednoduché, pretože nevieme vždy jednoznačne rozhodnúť o význame lokálnych miním a maxím. Jeden možný spôsob je založený na určení významnej vzdialenosti. Určíme vzdialenosť d pre jasové úrovne.

Nájdeme dve najväčšie lokálne maximá v histograme, ktoré sú vzdialené od seba viac ako *d*. Medzi týmito dvoma úrovňami nájdeme minimum a zodpovedajúcu úroveň jasu vezmeme za prahovú hodnotu.

Modifikáciu prahovania pre viac prahových hodnôt získame, ak zadáme intervaly alebo prípustné množiny úrovní jasu I_1, ..., I_n. Výsledkom nebude binárny obraz, ale obraz s *n* jasovými úrovňami. Klasifikáciu objektov uskutočníme pravidlom:

$$g(x, y) = 1, \quad \text{ak } f(x, y) \in I_1, \ldots,$$
$$n, \quad \text{ak } f(x, y) \in I_n,$$
$$0, \quad \text{inak.}$$

Môžeme opäť označiť funkciu prahovania *T* v nasledujúcej forme $T = T[x, y, p(x, y), f(x, y)]$, kde $f(x, y)$ je úroveň bodu (x, y), a $p(x, y)$ označuje lokálnu vlastnosť tohto bodu (napríklad, priemernú úroveň jasu okolia bodu (x, y)).

Vytvoríme prahovaný obraz $g(x, y)$ definovaný nasledovne:

$$g(x, y) = 1, \quad \text{ak } f(x, y) > T$$
$$0, \quad \text{ak } f(x, y) \leq T.$$

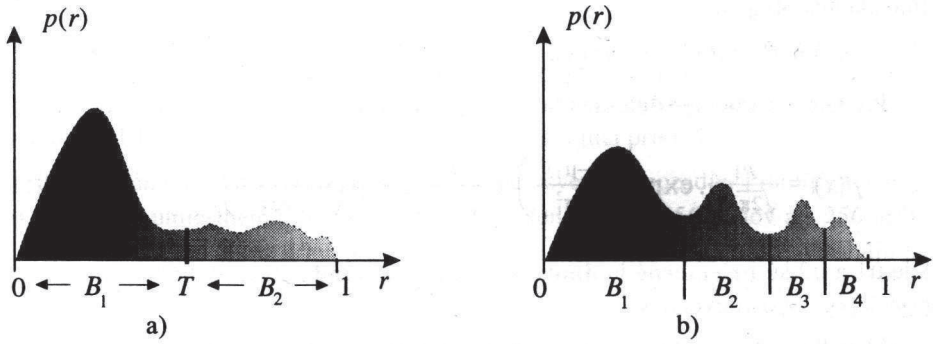
Takto vytvorený obraz $g(x, y)$ má označené pozadie nulou a objekty jednotkou. Keď prah *T* závisí len od $f(x, y)$, vtedy prahovanie nazývame **globálne**. Ak hodnota *T* závisí od $f(x, y)$ aj $p(x, y)$, potom prahovanie voláme **lokálne**. V prípade, že *T* závisí aj od priestorových súradníc *x* a *y*, potom prahovanie nazývame **dynamické**.

8.2.1 Globálne prahovanie

Jednoduché priblíženie k segmentácii obrazu je rozdelenie úrovní šedej do hraníc a použitie prahových hodnôt na určenie oblastí alebo na získanie hraničných bodov.

Predpokladajme, že šedý obraz $f(x, y)$ má histogram z obr. 8.2 a). Z histogramu vyvodzujeme, že veľký počet pixlov v obraze $f(x, y)$ je tmavých a zostávajúce pixle sú rovnomerne rozdelené z hľadiska jasu. Tento histogram je charakteristický pre obrazy zložené zo šedých objektov na tmavom pozadí.

Kvôli určeniu hranice medzi objektami a pozadím, rozdelíme tento histogram na dva oddelené intervaly určené prahom T, ako je to ukázané na obr. 8.2 a). K tomuto cieľu vyberieme prah T tak, aby interval B_1 určoval body pozadia a B_2 určoval body objektu. Ak obrazom prechádzame, zmena v šedých úrovniach od jedného intervalu k druhému nám označí hranicu.



Obr. 8.2 Histogram globálneho prahovania

Z toho dôvodu určíme hranice v horizontálnom aj vertikálnom smere dvoma prechodmi obrazom $f(x, y)$. Vo všeobecnosti môžu byť určené intervaly hodnôt $B1$, $B2$, ..., Bn.

Algoritmus vyhľadania hranice

1. Pre každý riadok v obraze $f(x, y)$ (t.j. $y = 0, 1, ..., N-1$), vytvoríme príslušný riadok v pomocnom obraze $g_1(x, y)$ použitím nasledovného vzťahu pre $x = 1, 2, ..., N-1$:

$$g_1(x, y) = \begin{cases} L_E, & \text{ak } f(x, y) \text{ a } f(x-1, y) \text{ sú v rôznych intervaloch,} \\ L_B, & \text{inak,} \end{cases} \quad (1)$$

kde L_E a L_B sú špecifikované úrovne **hranice** (*edge*) a **pozadia** (*background*).

2. Pre každý stĺpec v obraze $f(x, y)$ (t.j. $x = 0, 1, ..., N-1$), vytvoríme príslušný stĺpec v pomocnom obraze $g_2(x, y)$ použitím nasledovného vzťahu pre $y = 1, 2, ..., N-1$:

$$g_2(x, y) = \begin{cases} L_E, & \text{ak } f(x, y) \text{ a } f(x, y-1) \text{ sú v rôznych intervaloch,} \\ L_B, & \text{inak.} \end{cases} \quad (2)$$

3. Požadovaný obraz, pozostávajúci z hranice objektov odlíšených od pozadia je získaný použitím nasledovného vzťahu pre $x, y = 1, 2, ..., N-1$:

$$g(x, y) = \begin{cases} L_E, & \text{ak } g_1(x, y) \text{ alebo } g_2(x, y) \text{ je rovné } L_E, \\ L_B, & \text{inak.} \end{cases}$$

Výsledný obraz je **binárny**, t.j. máme dve špecifikované úrovne L_E a L_B hranicu a pozadia. Tento postup môžeme zmeniť tak, že kódujeme určitým spôsobom body hranice s odlišnými šedými úrovňami, a to podľa intervalov, v ktorých sa body v podmienke (1) a (2) nachádzajú.

8.2.2 Optimálne prahovanie

Predpokladajme, že obraz obsahuje dve rôzne odlíšené úrovne jasu (pozadie a objekt). Histogram tohto obrazu môžeme zostrojiť, keď odhadneme jas funkciou hustoty $p(x)$. Funkcia hustoty môže byť súčtom dvoch unimodálnych hustôt, pre pozadie a objekt v obraze.

Keby tvar hustôt rozdelenia bol známy, potom je možné určovať najvhodnejší prah (vzhľadom na minimálnu chybu) pre segmentáciu obrazu pre dve modality. Predpokladajme, že histogram obrazu je zložený aditívne z dvoch Gaussovských rozdelení. Táto funkcia hustoty je daná:

$$p(x) = P_1 . p_1(x) + P_2 . p_2(x),$$

Pre **Gaussovské rozdelenie** platí:

$$p(x) = \frac{P1}{\sqrt{2\pi}\,\sigma_1} . \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) + \frac{P2}{\sqrt{2\pi}\,\sigma_2} . \exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right), \qquad (3)$$

kde $\mu 1$ a $\mu 2$ sú **priemerné hodnoty** a $\sigma 1$ a $\sigma 2$ sú **štandardné odchýlky** a $P1$ a $P2$ sú pravdepodobnosti dvoch úrovní.

Platí $P1 + P2 = 1$. Ak všetky parametre sú známe, potom najvhodnejší prah T sa určí ľahko. Predpokladáme, že tmavá oblasť zodpovedá pozadiu. V prípade $\mu 1 < \mu 2$ môžeme určiť prah T tak, že všetky pixle s úrovňami jasu menšími ako T tvoria pozadie. Označíme pravdepodobnosti klasifikácie chyby určenia bodu objektu resp. pozadia $E_1(T)$ resp. $E_2(T)$.

Preto celková pravdepodobnosť chyby je daná:

$$E(T) = P_2 . E_1(T) + P_1 . E_2(T) .$$

Prahovú hodnotu T s minimálnou chybou určíme derivovaním $E(T)$ s použitím Leibnitzovho pravidla:

$$P_1 . p_1(T) = P_2 . p_2(T).$$

Po logaritmovaní Gaussovského rozdelenia hustoty vychádza pre T kvadratická rovnica:

$$A . T^2 + B . T + C = 0, \qquad (4)$$

kde

$$A = \sigma_1^2 - \sigma_2^2 , \; B = 2(\mu_1\sigma_2^2 - \mu_2\sigma_1^2), \; C = \sigma_1^2 . \mu_2^2 - \sigma_2^2 . \mu_1^2 + 2\sigma_1^2 . \sigma_2^2 \ln(\sigma_2 . P_1 / \sigma_1 . P_2).$$

V prípade, že odchýlky sú rovnaké $\sigma = \sigma 1 = \sigma 2$, prah je určený:

$$T = \frac{\mu_1 + \mu_2}{2} + \frac{\sigma^2}{\mu_1 - \mu_2} \ln\left(\frac{P_2}{P_1}\right).$$

Ak naviac $P1 = P2$, potom optimálny prah je určený priemerom z hodnôt $\mu 1$ a $\mu 2$.

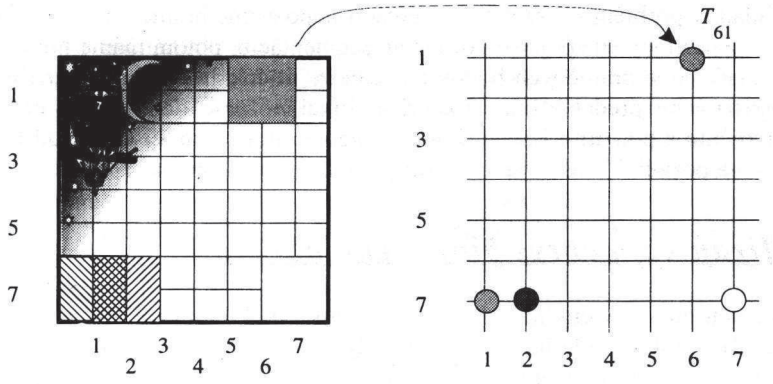
Na určenie parametrov histogramu obrazu môžeme použiť metódu priblíženia k minimálnej chybe. Ak predpokladáme N bodov histogramu, tak napr. metóda najmenších štvorcov dáva možnosť minimalizovať chybu medzi hustotou $p(x)$ a získaným histogramom obrazu $h(x_i)$:

$$M = \frac{1}{N} \sum_{i=1}^{N} [p(x_i) - h(x_i)]^2, \qquad (5)$$

8.2.3 Lokálne prahovanie

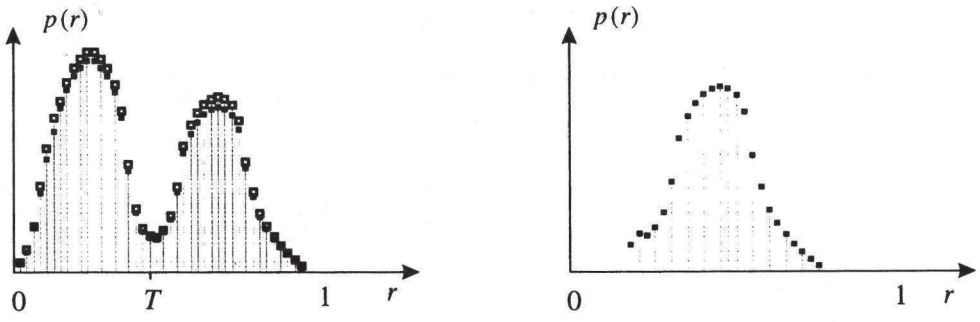
Lokálne prahovanie je funkciou polohy. Jednou z možností segmentácie je rozdeliť obraz na podobrazy a určiť prah nezávisle v každom z nich. Pokiaľ v niektorej z podoblastí nevieme definovať prahovú hodnotu, potom môžeme použiť interpoláciu susedných prahov. Ukážeme si na lokálne prahovanie nasledujúci príklad.

Príklad: Máme obraz, ktorý má nerovnomerne osvetlené pozadie. Preto prahovanie upravíme podľa umiestnenia. Nech je obraz s rozlíšením 256×256 bodov a s 256 úrovňami jasu. Rozdelíme obraz pravidelne na 7×7 podobrazov (oblastí), ktoré obsahujú 64 ×64 obrazových bodov (pozri obr. 8.3). Susedné podokná sa prekrývajú v spoločnej 50% časti.



Obr. 8.3 Rozdelenie obrazu na podobrazy

Pre každú oblasť zostrojíme histogram a testujeme, či môžeme použiť optimálne prahovanie pre bimodálny histogram. Na obr. 8.4 máme zobrazené dva prípady. Pre prvý sme optimálne prahovanie našli a pre druhý nenašli.



a) b)

Obr. 8.4 Určenie optimálneho prahu T pre bimodálny histogram

Pre ľubovoľný bod (x, y) určíme určíme prahovú hodnotu T_{xy} interpoláciou určených hodnôt z definovaných oblastí. Nakoniec môžeme vytvoriť binárny obraz nasledujúcim prahovaním:

$g(x, y) = 1$, ak $f(x, y) > T_{xy}$

$= 0$, inak.

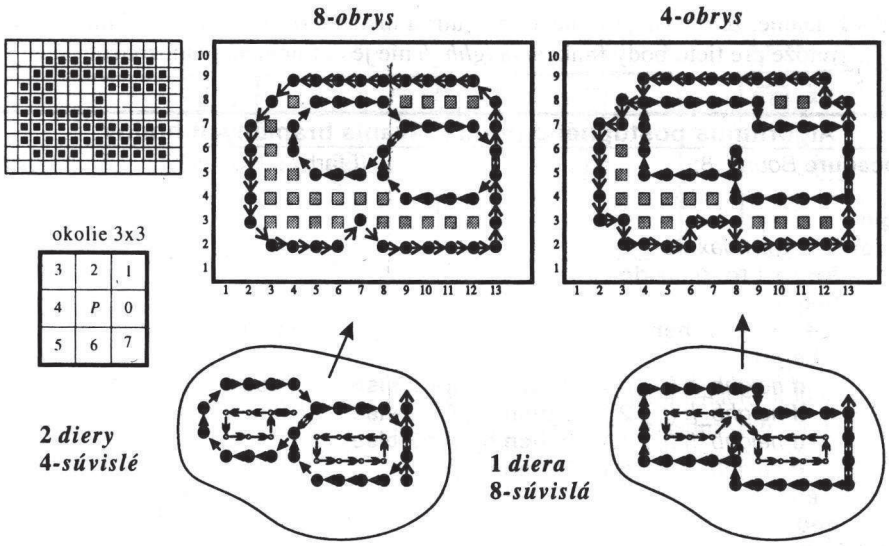
Jedným z prístupov ako zlepšiť vyšetrovanie histogramom je skúmať iba pixle blízko hranice. Zlepšenie je v tom, že histogramy budú menej závislé od rozmerov objektov a pozadia. Okrem toho pravdepodobnosť, že daný bod leží na pozadí alebo v objekte je skoro rovnaká. Preto dosiahneme symetriu vrcholov maxím histogramu.

Základný problém je, že pri segmentácii nepoznáme hranicu medzi objektom a pozadím. Tieto úlohy sú spojené. Ak máme segmentáciu, potom máme hranicu a naopak. Avšak zisťovanie hraničných bodov môžeme robiť lokálne pomocou gradientných metód popísaných v predchádzajúcej kapitole. Pixel môžeme identifikovať na hrane pomocou gradientu a použitím Laplaciánu dostaneme informáciu, či daný bod leží v objekte alebo je na pozadí. V nasledujúcej časti predpokladáme binárny obraz.

8.3 Hranica a obrys binárneho obrazu

Hranicu oblasti v euklidovskej rovine chápeme ako množinu tých bodov, pre ktoré sa v ich ľubovoľnom okolí nachádzajú body z oblasti aj jej doplnku (pozadia). Pretože nás zaujíma diskrétna reprezentácia obrazov (oblastí), budeme tomuto analogickému pojmu v diskrétnej rovine hovoriť hranica alebo obrys. V tejto časti uvedieme algoritmy pre vyhľadávanie hranice oblasti a obrysu.

Podobne ako sme definovali 4-súvislú (resp. 8-súvislú) oblasť, môžeme definovať 4-obrys (resp. 8-obrys). Bod P bude 8-obrysovým (resp. 4-obrysovým) bodom oblasti, ak v jeho najmenšom (3×3) okolí medzi susednými bodmi existuje aspoň jeden 4-sused (resp. 8-sused) nepatriaci oblasti. Na obrázku 8.5 vidíme rozdiel medzi 8-obrysom a 4-obrysom oblasti. Napríklad bod P = (3, 3) je 4-obrysovým, ale nie je 8-obrysovým.



Obr. 8.5 Rozdiel medzi 8-obrysom a 4-obrysom tej istej oblasti

Uvažujme zadanú 8-súvislú oblasť a hranicu hľadáme ako 8-obrys. Tú istú množinu môžeme vyšetrovať ako 4-súvislú a hľadáme 4-obrys. Doplnok 8-súvislej množiny sa rozpadá na 4-súvislé komponenty, a naopak 4-súvislej na 8-súvislé komponenty, tak ako je to vidieť aj na obr. 8.5.

V euklidovskej rovine vieme zostrojiť hranicu oblasti z orientovaných kriviek tak, aby sa daná oblasť nachádzala vľavo od krivky. V rastrovej rovine môžeme obdobne hranicu oblasti usporiadať do postupnosti hraničných bodov. Pri postupnom prechode hraničných bodov sa body oblasti budú nachádzať vľavo. Takúto postupnosť budeme nazývať kladne orientovaným obrysom (pozri obr.8.5). Ak sa body oblasti nachádzajú vpravo od obrysu, hovoríme o záporne orientovanom obryse. Hranicou budeme rozumieť (neusporiadanú) množinu hraničných bodov a obrysom (usporiadanú) postupnosť hraničných bodov. Na obrázku 8.5 je znázornené najmenšie okolie bodu P ako cyklická postupnosť bodov (0, 1, ..., 7). Zavedieme pre suseda bodu P označenie *neighb*(P, *i*), kde *i* nadobúda hodnotu 0,..,7. V algoritme prirodzene chápeme číslo *i* modulo 8.

8.3.1 Jednoduchý algoritmus hranice

Vyhľadať množinu hraničných bodov môžeme nasledovne. Pre každý bod oblasti testujeme, či je hraničným bodom. Stačí nám overiť, či jeden jeho 4-sused je z doplnku oblasti. Obrazová mapa je matica reprezentujúca obraz. Predpokladajme, že oblasť je zadaná svojimi vnútornými bodmi v obrazovej mape h[*i*, *j*] pre $i = 1,..., Xmax$ a $j = 1,..., Ymax$ hodnotou 1 a doplnok hodnotou 0.

Nasledujúca procedúra *Bound_8* v matici h označí hraničné body hodnotou 2. Funkcia *neighb_h* pre daný bod (*i*, *j*) a zadaný kód susednosti vráti hodnotu z matice h.

Predpokladáme, že oblasť je zadaná vo vnútri a nie na okraji obdĺžnika obrazovej mapy h[*i*, *j*], pretože pre tieto body funkcia *neighb_h* nie je definovaná jednoznačne.

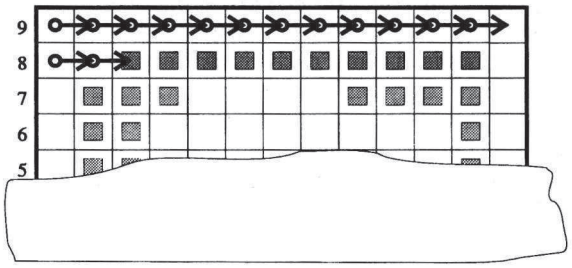
Algoritmus postupného prehľadávania hraničných bodov

```
Procedure Bound_8;                          { farby sú: 0 - doplnok, 1 - množina }

begin
   for i:= 1 to Xmax do
      for j:= 1 to Ymax do
      begin
        if h[i, j] = 1 then                                  { bod z množiny }
        begin
          if neighb_h (i, j, 0 ) = 0 then h[i, j]:= 2 else   { bod z doplnku množiny }
          if neighb_h (i, j, 2 ) = 0 then h[i, j]:= 2 else
          if neighb_h (i, j, 4 ) = 0 then h[i, j]:= 2 else
          if neighb_h (i, j, 6 ) = 0 then h[i, j]:= 2
        end
      end
end;
```

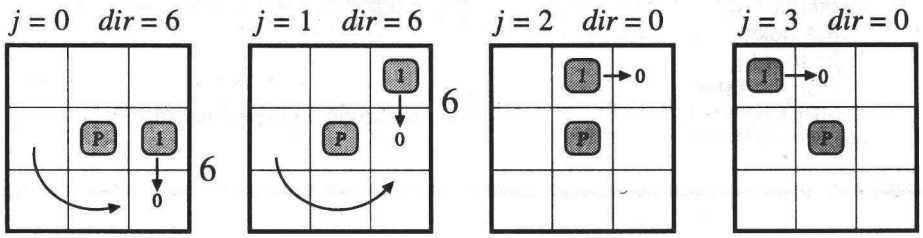
8.3.2 Jednoduchý algoritmus obrysu

Predpokladajme ako v predchádzajúcej časti, že oblasť máme zadanú v matici h[]. Určíme prvý hraničný bod, ktorý bude prvým bodom postupnosti obrysu. Tento bod vyhľadáme napríklad tak, že postupne prehľadávame body v riadkoch, až kým nenájdeme prvý bod s hodnotou 1 (pozri obr. 8.6). Túto procedúru vyhľadania prvého bodu označme *first*. Aktuálny bod obrysu označme *P*.



Obr. 8.6 Vyhľadanie prvého bodu v algoritme určenia obrysu

Algoritmus môžeme jednoducho popísať ako sledovanie cesty susedných hraničných bodov. Postupným prehľadávaním najmenšieho 3×3 okolia bodu P (postupne od doplnkového bodu) nájdeme prvý bod oblasti. Všimnime si, že tento bod bude zároveň aj bodom obrysu. Označme si *neighb*(P, *j*-1) predchádzajúci bod z doplnku oblasti a *neighb*(P, *j*) bod z oblasti. Pre ďalší krok algoritmu potrebujeme zistiť kód susednosti bodu *neighb*(P, *j*-1) v okolí bodu *neighb*(P, *j*). Z obrázku 8.7 vidíme, že každému bodu P zodpovedá v tabuľke nová hodnota susednosti.



Obr. 8.7 Niektoré prípady dvojíc bodov z množiny a doplnku

Takže pre index *j* zistíme príslušný vektor, ktorý určuje jednoznačne kód susednosti bodu *neighb*(P, *j*-1). Napríklad z obrázku 8.7 pre *j* = 0 je kód susednosti 6. Takto vytvoríme tabuľku kódov nových susedností, ktorú označíme ***tran***(*j*). V tejto tabuľke máme zapísané indexy, ktoré transformujú (*j*-1) suseda bodu P do kódu suseda bodu *neighb*(P, *j*).

j	0	1	2	3	4	5	6	7
tran	6	6	0	0	2	2	4	4

Tab. 8.1 Tabuľka zmeny kódov susednosti pri vytváraní obrysu

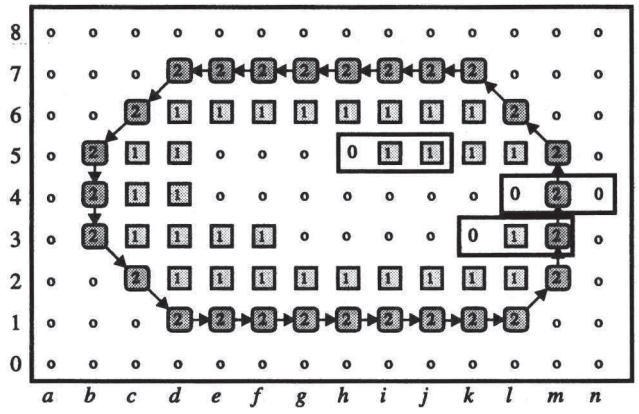
Našou úlohou je hľadať obrys oblasti dovtedy, pokiaľ nenarazíme opäť na prvý obrysový bod určený na začiatku. Nasledujúca procedúra '*Tracer*' nám v matici h označí hraničné body hodnotou 2 (3 alebo 4) v 8.kroku algoritmu. Funkcia '*neighb*' nám pre daný bod P a kód suseda vráti susedný bod. Pôvodná matica h charakterizuje body oblasti nenulovou hodnotou. Počas sledovania obrysu sa súradnice alebo smerníky bodov hranice ukladajú do zásobníka v 3. kroku algoritmu.

Algoritmus vyhľadania obrysu zadanej oblasti

```
procedure Tracer();
begin
0. first; { vyhľadá prvý bod obrysu                                                    }
{ ------------------------------------------------------------------------------------ }
{ od 1. kroku - modifikovaný algoritmus pre prípad vyhľadania vnútorného obrysu       }
1.  P:= first_P;                                  { vezme aktuálny bod P }
    j:= 4;                                        { 4-sused (P) je z doplnku }
    first_fl:= true;                              { prvý raz prejdi cyklus }
2.  while ( (P ≠ first_P) or (first_fl = true) )
    begin
3.      repeat                                    { v cykle hľadaj bod oblasti }
            j:= j+1;
4.          next:= neighb ( P, j );
5.      until (h[next] = 1);
```

```
6.      c := j;
        push (P, c);                          { odlož bod P a kód do zásobníka }
7.      P:= next;
        j:= tran (j);
        first_fl:= false;
8.      h[P]:= h[P] + 1;                      { poznač v obrazovej mape h }
    end;
end;
```

Takto formulovaný algoritmus *Tracer* vyhľadá vonkajší obrys oblasti. Ak chceme určiť vnútorný obrys oblasti, musíme vyhľadať vnútorný hraničný bod. Vyšetrujme úseky riadkov, ktorých koncové body máme uložené v zásobníku ako vonkajšie obrysové body. Z obrázku 8.8 vidíme, že tri susedné body v riadku, ak majú hodnoty (0, 1, 1), (0, 1, 2) alebo (0, 2, 0) charakterizujú opäť hraničný bod (prostredný v trojici), ktorý indikuje, že v oblasti je diera. Pre takýto bod môžeme opäť použiť procedúru *Tracer* od kroku 1.



Obr.8.8 Indikátory diery oblasti pre trojice bodov za sebou v riadku

V predchádzajúcej procedúre *Tracer* ukladáme do zásobníka nielen súradnice bodov, ale tiež kód susednosti nasledovníka obrysu. Všimnime si, že obrys je jednoznačne zadaný prvým bodom a postupnosťou kódov. Napríklad pre vonkajší obrys oblasti z obrázku 8.8 budeme mať v zásobníku od prvého bodu (*d*, 7) nasledujúce kódy :

(*d*, 7) → (5, 5, 6, 6, 7, 7, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 3, 3, 4, 4, 4, 4, 4, 4, 4).

Ak sa obrys ukončí, potom do zásobníka uložíme hodnotu kódu 8, čím indikujeme ukončenie. Funkcia *neighb* nám vráti susedný bod. V kroku 5 vyberáme trojice bodov za sebou v riadku a v kroku 6 testujeme pre hľadanú postupnosť trojíc bodov (0,2,0) a (0,1,x). V kroku 7 vieme zistiť, že či ide o vonkajší obrys, a preto môžeme testovanie ukončiť.

Vyhľadanie obrysu aj vnútorných dier

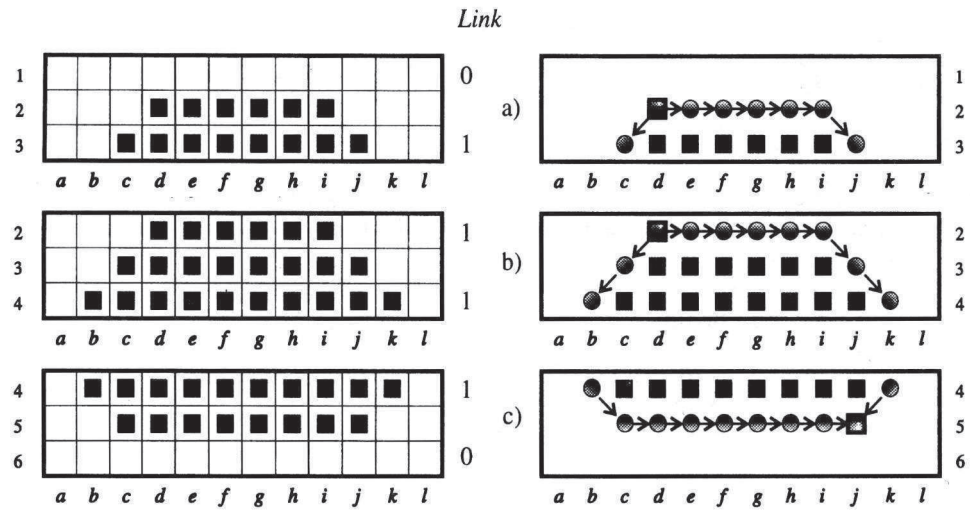
```
procedure Trac_contour;

begin
0.  first(first_p);                                  { vyhľadá prvý bod obrysu }
1.  Tracer (first_p);                                { vezme prvý bod pre obrys }
2.  while ( queue(P) ≠ 0 );
    begin
3.      (p, c):= pop(P);                             { vyber bod a kód smeru }
4.      if c = 8 then begin
                        c0:=c;                       { ak je koniec obrysu }
                        (p, c):= pop(P);             { vyber ďalší bod }
                      end
        if ( (4 ≤ c0 and c0 ≤ 7) or (5 ≤ c and c ≤ 7) then
              begin
5.              ap:= neighb (p, 0);
                bp:= neighb (ap, 0);
                cp:= neighb (bp, 0);
                ex:= false;
                repeat
6.                if (h[ap]=0 and h[bp]=1) or                    { prípad (0, 1, x) }
                    (h[ap]=0 and h[bp]=2 and h[cp]=0) then      { prípad (0, 2, 0) }
                  begin                                  { bod vnútorného obrysu }
                    Tracer (bp); ex:= true;
                  end
7.                else if (h[ap]=0 and h[bp]>2 and h[cp]=0)
                       or (h[ap]=1 and h[bp]=2 and h[cp]=0) then ex:= true
                  else                              { ďalší bod riadku, posuň ap, bp, cp }
                  until ( ( cp.x = Xmax ) or (ex) );
              end;
8.      c0:=c;                                                 { ak je koniec obrysu }
    end;
end;
```

8.3.3 Skanovací algoritmus obrysu

Doteraz sme predpokladali, že obrazová mapa oblasti je uloženú v pamäti. V niektorých prípadoch to nie je možné, a preto uvedieme algoritmus, ktorý minimalizuje pamäťové nároky. Je celkom prirodzené vyšetrovať jednotlivé riadky obrazu a udržiavať v pamäti, len dva až tri aktuálne riadky. Takýto postup riešenia problému zaraďujeme do skupiny **skanovacích algoritmov** (*scan-line algorithm*).

Na obrázku 8.9 vidíme tri typické prípady. Vľavo je vždy znázornená obrazová mapa obrazu na troch riadkoch a vpravo je znázornená situácia pre vytvárajúce sa smerníky obrysu (kódy). Hodnoty získané z uvedenej procedúry *Link* (z predchádzajúcej kapitoly) informujú o situácii nad a pod aktuálnym riadkom. Na obr. 8.9 je znázornená situácia, keď horný riadok neobsahuje body z oblasti (prípad a) alebo dolný riadok neobsahuje body z oblasti (prípad c).



Obr. 8.9 Určenie obrysu oblasti pomocou riadkového prístupu

V prípade a) sa začína vytvárať obrys v bode (*d*, 2) a informácia o obryse sa ukladá ako postupnosť nasledujúcich bodov:

(*d*, 2) → (*e*, 2) → (*f*, 2) → (*g*, 2) → (*h*, 2) → (*i*, 2),

čo môžeme prepísať do postupnosti reťazových kódov od začiatku takto:

(*d*, 2) → (0, 0, 0, 0, 0).

V prípade b) vidíme, že sa postupnosť kódov pre pravý obrys predĺži a naviac sa začne vytvárať aj postupnosť kódov pre ľavý obrys. Podrobnejšie si všimnime situáciu na obrázku 8.9 c), kde uzatvárame postupnosti kódov do koncového bodu (*j*, 5). Pre postupnosť kódov dostaneme dva obrysy:

(*d*, 2) → (0, 0, 0, 0, 0, 7, 7, 5); (pravý obrys),

(*d*, 2) → (5, 5, 7, 0, 0, 0, 0, 0, 0, 0); (ľavý obrys).

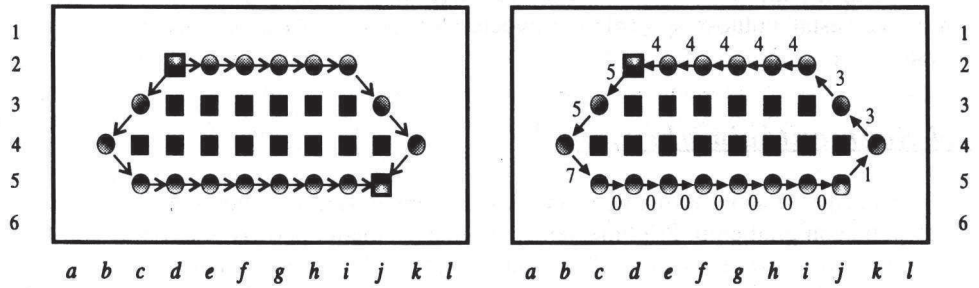
Ak chceme vytvoriť vonkajší obrys orientovaný kladne, potom musíme pre pravý obrys postupnosti kódov obrátiť smery podľa nasledujúcej tabuľky na opačné smery a postupovať od posledného kódu až do prvého.

kód	0	1	2	3	4	5	6	7
invers	4	5	6	7	0	1	2	3

Tab. 8.2 Inverzný kód susednosti pre 8 susednosť

Pre vonkajší obrys danej oblasti znázornenej na obr. 8.10 dostaneme výsledný reťazový kód :

$(d, 2) \rightarrow (5, 5, 7, 0, 0, 0, 0, 0, 0, 0, 1, 3, 3, 4, 4, 4, 4, 4)$.



Obr. 8.10 Výsledný reťazový kód vonkajšieho obrysu oblasti

V prípade obr. 8.11 je zobrazená situácia, kedy musíme začať aj s vnútornou časťou obrysu, preto máme dve rôzne začiatky $(d, 2)$ a $(f, 3)$:

$(d, 2) \rightarrow (0, 0, 0, 0, 0, 0, 7)$; (pravý obrys),

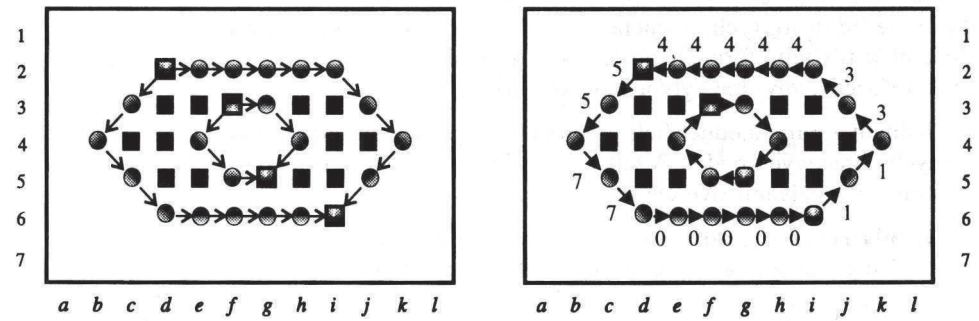
$(d, 2) \rightarrow (5)$; (ľavý obrys),

$(f, 3) \rightarrow (0, 7)$; (pravý obrys),

$(f, 3) \rightarrow (5)$; (ľavý obrys).

Sú to klasické príklady rozvetvenia a spájania smerníkov (kódov). Pre vnútornú dieru oblasti na obrázku 8.11 dostaneme postupnosti kódov:

$(f, 3) \rightarrow (5, 7, 0)$; $(f, 3) \rightarrow (0, 7, 5)$; (ľavý a pravý obrys).



Obr. 8.11 Vetvenie a spájanie smerníkov (kódov) obrysov oblasti

Pre spojenie týchto smerníkov do reťazového kódu musíme obrátiť v tomto prípade smery ľavého obrysu tak, aby sme dostali kladne orientovaný obrys diery oblasti. Dostaneme túto postupnosť výsledného reťazca:

$(f, 3) \rightarrow (0, 7, 5, 4, 3, 1)$.

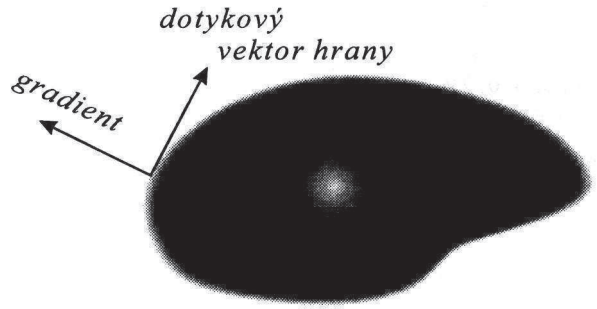
Pre vonkajší obrys danej oblasti znázornenej na obr.8.11 dostaneme výsledný reťazový kód:

$(d, 2) \rightarrow (5, 5, 7, 7, 0, 0, 0, 0, 0, 1, 1, 3, 3, 4, 4, 4, 4, 4)$.

Treba upozorniť, že vo všeobecnosti môže byť situácia spájania ešte zložitejšia, pretože môže nastať nutnosť spojenia nie dvoch, ale viacerých častí na vytvorenie celého obrysu.

8.4 Segmentácia obrysom

V tejto časti opäť vyšetrujeme viacúrovňové obrazy. Hraničný bod môžeme identifikovať pomocou gradientu. Pripomenieme, že smer gradientu je daný smerom najväčšieho rastu obrazovej funkcie. Obr. 8.12 znázorňuje vzťah hrany a gradientu. Ak objekt zodpovedá oblasti zhruba konštantného jasu, potom jeho hranica je určená v miestach, kde nastáva výrazná zmena jasu a vidíme, že hranica je kolmá na smer gradientu.



Obr. 8.12 Hrana a smer gradientu oblasti

Táto metóda vychádza z hraníc oblastí vytvorených hranovými operátormi, ktorými sme sa zaoberali v časti pre ostrenie obrazu. Hrany označujú miesta, v ktorých dochádza k istej nespojitosti, rýchlej zmene jasu. Takto identifikované body sú však pre identifikáciu oblastí v tejto forme nepoužiteľné. A preto musí nasledovať spájanie bodov hranice do reťazcov, aby sme vytvorili obrys oblasti.

Jedným z najjednoduchších prístupov na spájanie hraničných bodov je analýza susedných obrazových bodov. Všetky body, ktoré sú **podobné**, sa pospájajú. Na definíciu podobnosti využívame dve vlastnosti.

1. Sila reakcie gradientu na vytvorenie hraničného bodu. Táto vlastnosť je daná veľkosťou gradientu napr. pomocou Sobelovho operátora:

$$G[f(x, y)] = |D_x| + |D_y| \qquad (6)$$

2. Smer gradientu je definovaný hodnotou:

$$\varphi = \tan^{-1}(D_y/D_x), \text{ kde } D_x \neq 0. \qquad (7)$$

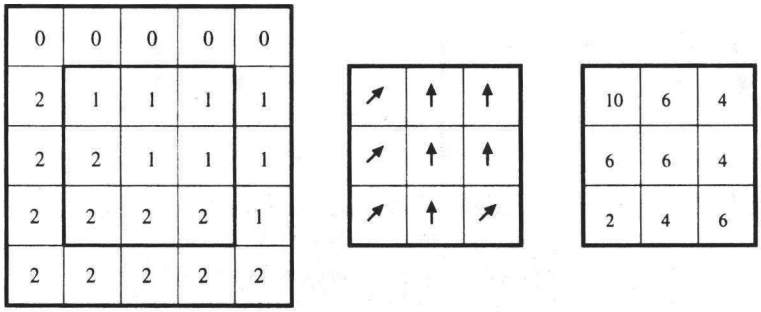
Pre podobnosť sú zadané dve prahové hodnoty T a A. Spájame body (x', y') v okolí bodu (x, y), ak sú splnené nerovnosti:

$|G[f(x, y)] - G[f(x', y')]| \leq T$ - body sú podobné vo veľkosti gradientu,

$|\varphi(x, y) - \varphi(x', y')| \leq A$ - a podobné v smere gradientu.

Popíšeme stratégiu určenia obrysu oblasti vychádzajúc z gradientu. Predpokladajme, že máme zadaný obraz. Určíme hodnoty Sobelovho operátora D_x a D_y z 3×3 okolia bodu a pomocou nich smer gradientu (7), zaokruchlením do násobku 45°. Na veľkosť gradientu použijeme vyjadrenie, ktoré zaviedli Frei a Chen pomocou konvolučných masiek uvedených v predchádzajúcej kapitole [GONZ87]. Tieto vzniknú rotáciou základnej masky o 45°, čím zohľadníme derivácie vo viacerých smeroch. Na obr. 8.13 máme znázornenú časť digitálneho obrazu, veľkosť gradientu a smer gradientu.

$$h_1 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}, h_2 = \begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}, h_3 = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, h_3 = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

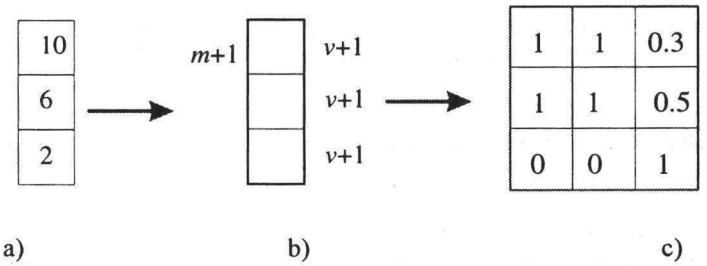


Obr. 8.13 Výrez obrazu, smer a veľkosť gradientu

Pravidlo **LBE** (*Likelihood Being an Edge* - pravdepodobne byť hranou) sformulujeme nasledovným spôsobom:

Hodnota LBE pre každý bod P je z intervalu $\langle 0, 1 \rangle$. Pre výpočet zavedieme dve lokálne hodnoty pre každý pixel: m - určuje počítadlo, koľkokrát je bod lokálnym maximom, a v je počítadlo, koľkokrát bol pixel navštívený. Na každý pixel umiestnime okno 3x1 v smere gradientu podľa obr. 8.14. Pre každý pixel okna zvýšime počítadlo v a pre ten pixel, v ktorom je maximálna veľkosť gradientu, zvýšime počítadlo m.

Po ukončení skanovania obrazu, vypočítame hodnotu v pixli P LBE(P) = m/v.



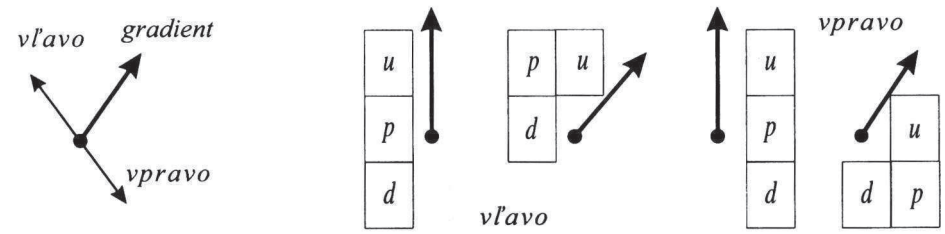
Obr. 8.14 Výpočet LBE a) veľkosť gradientu v okne b) zvýšenie hodnôt m, v c) výsledné hodnoty LBE

Nakoniec postupujeme **vyhľadávaním obrysu**, ktorý bude zodpovedať hraničným bodom. Obraz opäť skanujeme po všetkých bodoch. V tých, v ktorých má hodnotu LBE = 1 a ešte nie je označený pixel ako spracovaný, začneme vyhľadávať ľavý a pravý obrys vzhľadom na gradient. Pre každý obrysový bod pokračujeme podľa pravidla (pozri obr. 8.15):

1. Označme si tri nasledujúce body *u* (*up* - **horný**), *p* (*perpendicular* - **kolmý**), *d* (*down* - **dolný**) rešpektovaním gradientu.

2. Označme *max* maximálnu hodnotu ich LBE hodnôt.

3. Predpokladajme, že $max \neq 0$ a body *u*, *p*, *d* nie sú označené, preto môžeme pokračovať v obryse.



Obr. 8.15 Ľavý a pravý sused od bodu určeného gradientom

Popíšeme algoritmus určenia obrysových bodov:

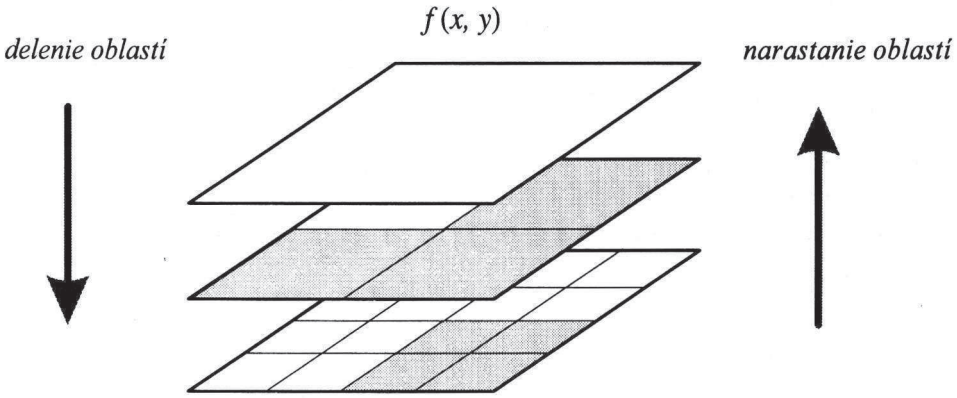
Algoritmus obrysu

```
if LBE(p) = max then
    begin
    next(p) - p je nasledujúci bod obrysu;      { dávame prednosť kolmému smeru }
    if LBE(u) = 1 then modify ( LBE(u))         { zmenši hodnotu LBE }
    if LBE(d) = 1 then modify ( LBE(d))
    end
else if len 1 hodnota LBE(u) alebo LBE(d) = max then pokračuj, kde je max;
else if LBE(u)= max and LBE(d) = max then nextjunction (P);     { označ spojový bod }
```

Nakoniec tie obrysy, ktoré majú dĺžku menšiu ako je predpísaná prahová hodnota *T*, môžeme zo skúmania vylúčiť. Ak obraz má veľký šum, odporúčame najprv obraz vyhladiť.

8.5 Segmentácia narastaním oblasti

Z hľadiska segmentácie je dôležitou vlastnosťou **homogenita oblastí**. Základná myšlienka segmentácie narastaním a delením oblastí je rozdeliť obraz do maximálnych súvislých oblastí. Homogenitu oblasti môžeme definovať pomocou jasových úrovní, ale tiež pomocou popisu ako je to pre textury. Najjednoduchšie kritérium homogenity je stredná hodnota jasu oblasti. Prístupy narastania resp. delenia oblastí sú prístupom riešenia zdola resp. zhora z hľadiska pyramidálnej štruktúry, znázornenej na obr. 8.16.



Obr. 8.16 Delenie a narastanie oblastí v pyramidálnej dátovej štruktúre

Narastanie oblastí môžeme začať na úrovni pixlov. Definujeme prvotné rozdelenie do veľkého množstva oblastí. Pokiaľ dve susedné oblasti môžeme spojiť podľa kritéria, vytvoríme novú spojenú oblasť. Tento proces opakujeme dovtedy, kým nemôžeme už spojiť žiadne dve susedné oblasti. Delenie oblastí je opačný proces segmentácie. Najprv definujeme spravidla jednu veľkú oblasť, ktorú postupne delíme na menšie oblasti. Tento postup opakujeme dovtedy, kým nemôžeme rozdeliť žiadnu oblasť.

Obidva prístupy neprinášajú rovnaké výsledky. Na jednoduchom príklade šachovnice dostaneme pri kritériu strednej hodnote jasu rôzne výsledky. Narastaním oblasti sa postup ukončí, keď sa vytvoria štvorce šachovnice. Naopak pri delení sa algoritmus ukončí hneď v prvom kroku, pretože pri rozdelení sú stredné hodnoty jasu rovnaké.

Nasledujúci algoritmus spája predchádzajúce prístupy:

1. Definujeme prvotné rozdelenie do oblastí, kritéerium homogenity a vytvoríme pyramidálnu štruktúru.

2. Ak platí na i-tej úrovni, že oblasť nie je homogénna, rozdelíme ju na štyri samostatné oblasti. Naopak, ak pre niektoré štyri oblasti môžeme spojiť na vyššej úrovni do jednej oblasti, uskutočníme spojenie.

3. V prípade, že už nemôžeme ani rozdeliť ani spojiť, potom najprv hľadáme dve oblasti na rôznych úrovniach, či sa nedajú spojiť. Malé oblasti pripojíme k najpodobnejšej väčšej susednej oblasti.

9

Morfologické transformácie a skelet

9.1 Úvod do morfológie

Jednoduché segmentovanie vždy neposkytuje dostatok informácií, a preto sa pracuje často s morfologickými transformáciami v rovine resp. priestore, kde za tretí rozmer môžeme považovať úroveň šedej farby. Obrazy budeme spracovávať hlavne dvojúrovňové (binárne obrazy), ale niektoré operácie môžeme upraviť aj pre viacúrovňové. Preto morfologické transformácie väčšinou aplikujeme až po segmentácii. **Morfológiu** môžeme považovať za obdobu klasických konvolučných filtrácií. Hlavný dôvod tranformácie obrazu je v jeho zjednodušení. Obraz sa snažíme správne segmentovať a potom ho ďalej spracovávať, ako príklad uvedieme **skelet** (*kostru*, t.j. minimálne popisujúcu množinu).

Matematická morfológia predstavuje prostriedok na geometrickú analýzu obrazu. Používa sa predovšetkým na predspracovanie obrazu, zdôraznenie štruktúry objektov (skelet, stenčovanie a zosilňovanie) a popis objektov číselnými charakteristikami (plocha, obvod a iné). Konečným cieľom je kvantitatívna číselná charakteristika. Napríklad v rovine sa používajú tri na posunutí nezávislé miery, a to: plocha, obvod a Eulerova charakteristika. V morfológii sa prejavuje lokálna znalosť obrazu na rozdiel od globálnej znalosti použitej v iných prístupoch. Princípy matematickej morfológie sú postavené na presných definíciách, čo má výhodu v tom, že tvrdenia o vlastnostiach obrazu sa dajú matematicky dokazovať, [SERR82].

Každá informácia o obraze je výsledkom interakcie s testovacou množinou (*bázou*) nazývanou **štrukturálny prvok**. *Štrukturálny prvok B* je množina, ktorá spoločne so študovanou reláciou (napr. stenčovania alebo zosilňovania) definuje určitú transformáciu obrazu. Často budeme využívať spôsob definovania pomocou posunutia. Pre danú množinu X definujeme posunutý obraz X_b predpisom:

$$X_b = \{ y \mid y = x + b, \text{ pre všetky } x \in X \text{ a pre zvolený prvok } b \in B \}.$$

V ďalšom budeme predpokladať, že máme zadaný systém podmnožín. Základnú množinu E si môžeme predstaviť v spojitom prípade euklidovsku rovinu a v diskrétnom prípade vydláždenú rovinu.

9.2 Konštrukcia elementárnych transformácií

V tejto časti definujeme niektoré základné transformácie pre binárne obrazy. Z nich operácia **erózie** nám množinu stenčuje a **dilatácia** zosilňuje. Operácia **otvorenia** a **uzatvorenia** je podobná ako v euklidovskej rovine množinové operácie otvorenia a uzáveru množiny. Nakoniec zavedieme **stenčovanie** a **zosilňovanie**, ktoré majú najväčší význam pri spracovaní obrazu.

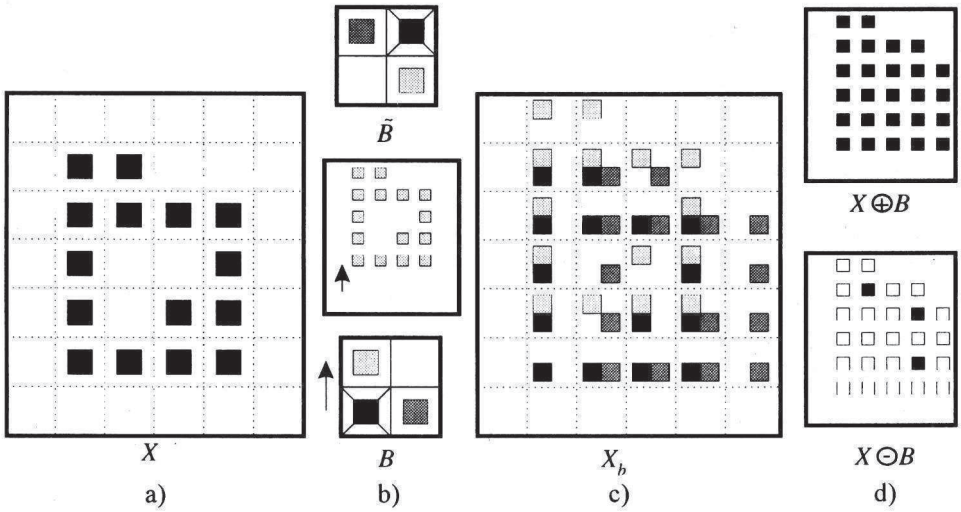
9.2.1 Dilatácia a erózia

Označme množinu B_x ako obraz posunutia množiny B do bodu x a $\widetilde{B}$ symetrickú množinu podľa začiatku súradníc. Dilatáciu a eróziu definujeme nasledujúcim predpisom:

$$X \oplus B = \cup X_b = \{x |\ B_x \cap X \neq \emptyset\} \qquad (dilatácia),$$

$$X \ominus B = \cap X_b = \{x |\ \widetilde{B}_x \subseteq X\} \qquad (erózia).$$

Ukážeme jednoduchý príklad pre dilatáciu a eróziu. Na obr. 9.1 vidíme zadanú množinu X a štrukturálny prvok B, kde diagonálnym krížikom označíme začiatok (*origin*) B, ktorý prikladáme na prvok množiny X. V časti b) je znázornená množina X_b pre najsvetlejší prvok $b \in B$, ktorý sa posúva vertikálne vzhľadom k začiatku B a v časti c) situácia obrazových bodov pre tri posunuté množiny X_b, ktoré sa zobrazujú obrazovými bodmi odlíšenými šedými farbami pre jednotlivé body množiny B. Dilatácia $X \oplus B$ je zjednotenie všetkých množín X_b (na obrázku sú zobrazené všetky body) t.j. všade, kde zasiahla niektorá z množín X_b. V časti d) vidíme výsledok operácie dilatácie. Erózia $X \ominus B$ je prienik všetkých množín X_b, preto vo výsledku sú zobrazené na obrázku len tie body, ktoré sa nachádzajú vo všetkých množinách X_b. Zároveň vidíme, že sú to práve tie body, pre ktoré zároveň platí $\widetilde{B}_x \subseteq X$.



Obr. 9.1 Ukážka dilatácie a erózie na množinu X

Z obrázku je vidieť, že dilatácia sa používa na zaplnenie malých dier a úzkych zálivov. Dilatácia objekty zväčšuje, a preto pokiaľ chceme po dilatácii objekt dostať do pôvodných rozmerov, používame po dilatácii eróziu, ktorá objekt zmenší.

Pre náročnejšieho čitateľa uvádzame zaujímavé a dôležité vlastnosti, ktoré platia pre dilatáciu a eróziu:

1. komutatívnosť $X \oplus B = B \oplus X, \qquad X \ominus B \neq B \ominus X$

2. asociatívnosť $X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z, \qquad X \ominus (Y \ominus Z) = (X \ominus Y) \ominus Z$

3. invariantnosť k posunutiu $X_h \oplus B = (X \oplus B)_h, \qquad X_h \ominus B = (X \ominus B)_h$

4. monotónnosť $X \subseteq Y \Rightarrow X \oplus B \subseteq Y \oplus B, \qquad X \subseteq Y \Rightarrow X \ominus B \subseteq Y \ominus B$

5. duálnosť erózie a dilatácie $X^c \oplus B = (X \ominus B)^c, \qquad X \oplus B = (X^c \ominus B)^c$

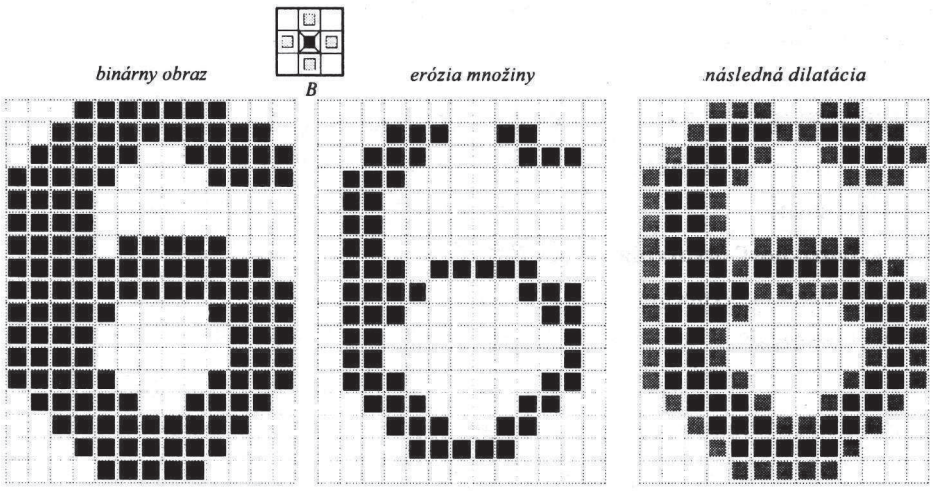
6. inklúzia erózie a dilatácie $0 \in B \Rightarrow X \ominus B \subseteq X \subseteq X \oplus B$

9.2.2 Otvorenie a uzatvorenie

Z predchádzajúcich úvah vyplýva, že je vhodné použiť operácie dilatácie a erózie za sebou. Ak použijeme najprv dilatáciu a potom eróziu, to nie je to isté, ak vymeníme poradie, t.j. operácie dilatácie a erózie nie sú navzájom komutatívne. Z toho dôvodu definujeme ďalšie dve transformácie:

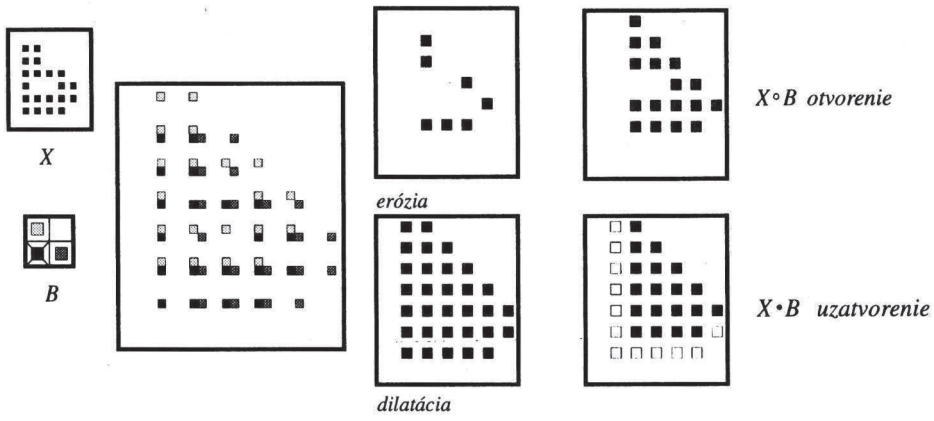
$X \circ B = (X \ominus B) \oplus B$ (*otvorenie*),

$X \bullet B = (X \oplus B) \ominus B$ (*uzatvorenie*).



Obr. 9.2 Príklad otvorenia a nasledného uzatvorenia množiny

Ukážeme si tiež jednoduchý príklad pre otvorenie a uzatvorenie. Na obrázku 9.2 je znázornený výsledok otvorenia, t.j.erózie a následnej dilatácie množiny. Z obrázku vidíme, že erózia rozdelí tenké čiary, ktoré následne dilatácia môže ale nemusí spojiť. Vo všeobecnosti otvorenie môže rozdeliť čiary spojené úzkou líniou, a naopak uzatvorenie spojí objekty, prípadne zaplní malé diery. Na nasledujúcom obr. 9.3 máme znázornenú postupnosť krokov pre vytvorenie dilatácie a erózie, na ktorom sa pre otvorenie rozdelila čiara a pre uzatvorenie zaplnila diera.

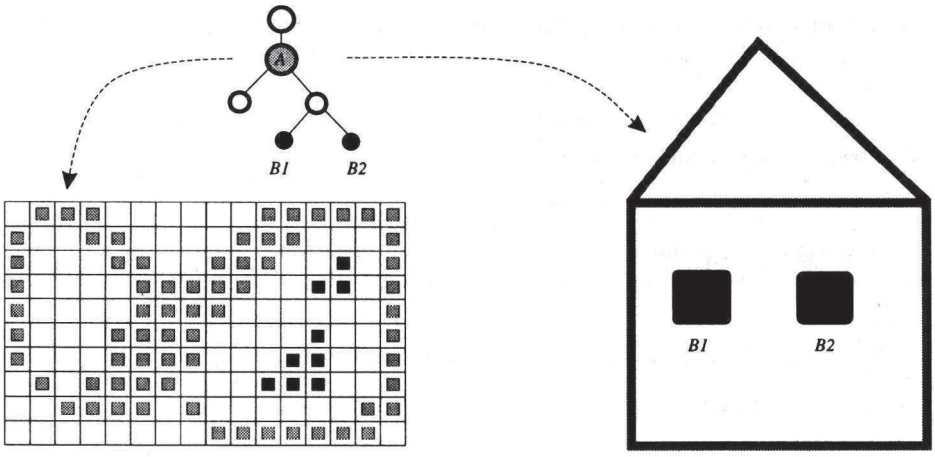


Obr. 9.3 Postupne vytvorenie otvorenia a uzatvorenia

Opäť pre náročnejšieho čitateľa uvedieme bez dôkazu niekoľko zaujímavých vlastností, ktoré platia pre tieto dve transformácie:

1. pre otvorenie platí	$X \circ B = \cup B_x$, pre $B_x \subseteq X$
2. invariantnosť k posunutiu	$X \circ B_h = X \circ B \qquad X \bullet B_h = X \bullet B$
3. monotónnosť	$X \subseteq Y \Rightarrow X \circ B \subseteq Y \circ B$ a $X \bullet B \subseteq Y \bullet B$
4. antiextenzívnosť otvorenia	$X \circ B \subseteq X$
5. extenzívnosť uzatvorenia	$X \subseteq X \bullet B$
6. idempotentnosť	$(X \circ B) \circ B = X \circ B$ a $(X \bullet B) \bullet B = X \bullet B$

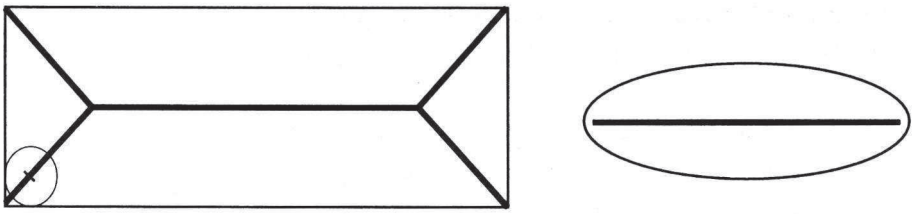
Homotopické transformácie zachovávajú spojitosť (súvislosť) objektov. Túto reláciu môžeme vyjadriť pomocou homotopického stromu. Jeho koreň reprezentuje pozadie obrazu a každá ďalšia úroveň reprezentuje buď objekty (komponenty) alebo diery. Na obrázku obr. 9.4 vidíme dva obrazy, ktoré majú rovnaký homotopický strom. Otvorenie a uzatvorenie nezachováva homotopický strom, pretože uzatvorenie môže uzavrieť dieru a otvorenie roztrhnúť množinu. To znamená, že nie sú to homotopické transformácie.



Obr. 9.4 Homotopický strom objektov v obraze

9.2.3 Stenčovanie a zosilňovanie

Častokrát je výhodné popísať objekty pomocou tenkých čiar tak, aby ostal zachovaný ich homotopický strom. Takému popisu sa hovorí skelet alebo kostra objektu. Pre spojitý prípad sa dá definovať skelet množiny ako zjednotenie stredov kružníc, ktoré sú obsiahnuté v objekte a dotýkajú sa hranice množiny aspoň v dvoch bodoch. Na obr. 9.5 je znázornený skelet dvoch objektov hrubšou čiarou.



Obr. 9.5 Skelet množín v euklidovskej rovine

Pomocou erózie môžeme stenčovať množinu a tak vytvárať skelet (kostru) množiny. Takto získaný skelet nie je však homotopicky ekvivalentný so zadanou množinou. Preto nahradzujeme eróziu sekvenčným homotopickým stenčovaním. Najprv si zavedieme nasledujúcu definíciu.

Serrova transformácia (*Hit or Miss*) je definovaná pre zložený štrukturálny prvok $\{B_1, B_2\}$, kde B_1 je podmnožina množiny X a B_2 je podmnožina doplnku X^c:

$$X \otimes B = \{x | (B_1)_x \subseteq X \text{ a } (B_2)_x \subseteq X^C\} .$$

Serrovu transformáciu vyjadríme pomocou dilatácie a erózie nasledujúcou rovnosťou:

$$X \otimes B = (X \ominus B_1) \cap (X^C \ominus B_2) = (X \ominus B_1) - (X \oplus B_2)$$

Pomocou Serrovej transformácie teraz môžeme vyjadriť stenčovanie a zosilňovanie

$X \nabla B = X - (X \otimes B)$ - stenčovanie

$X \Delta B = X \cup (X \otimes B)$ - zosilňovanie

Transformácie stenčovania a zosilňovania sú duálne:

$(X \Delta B)^C = X^C \nabla B^*$, kde $B^* = (B_2, B_1)$.

Stenčovanie a zosilňovanie sa často používajú opakovane, pretože nie sú idempotentné. Z toho dôvodu si zavedieme pre postupnosť zložených štrukturálnych prvkov $\{B_1, B_2, \ldots\}$

$X \nabla \{B_i\} = (((X \nabla B_1) \nabla B_2) \ldots)$ - sekvenčné stenčovanie,

$X \Delta \{B_i\} = (((X \Delta B_1) \Delta B_2) \ldots)$ - sekvenčné zosilňovanie.

Sekvenčné stenčovanie konverguje do konečného stavu, keď sa po sebe idúce iterácie nelíšia. Počet iterácií závisí od objektu a veľkosti štrukturálnych prvkov. Za určitých predpokladov sekvenčné stenčovanie a zosilňovanie zachovávajú homotopické stromy. V praxi existuje niekoľko používaných prvkov, ktoré tu uvedieme. Pre jednoduchosť budeme štrukturálny prvok uvádzať jedinou maticou, kde jednotka označuje bod množiny B_1 a nula označuje bod množiny B_2. Hviezdička v štrukturálnom prvku v matici označuje nezávislý bod, ktorý na porovnávanie nemá vplyv.

Štrukturálne prvky $\boldsymbol{L}$ pre 4-susednosť môžeme popísať ôsmimi maticami, kde

$$L_1 = \begin{pmatrix} 0 & 0 & 0 \\ * & 1 & * \\ 1 & 1 & 1 \end{pmatrix}, L_2 = \begin{pmatrix} * & 0 & * \\ 1 & 1 & 0 \\ 1 & 1 & * \end{pmatrix}, \ldots,$$

a nasledujúce matice získame pootočením o 90 stupňov.

Štrukturálne prvky $\boldsymbol{L}$ pre 8-susednosť vytvoríme matice z L_1 a L_2 rotáciou o 90°.

$$L_1 = \begin{pmatrix} 0 & 0 & 0 \\ * & 1 & * \\ 1 & 1 & 1 \end{pmatrix}, L_2 = \begin{pmatrix} * & 0 & 0 \\ 1 & 1 & 0 \\ * & 1 & * \end{pmatrix}, \ldots,$$

Podobne definujeme štrukturálny prvok $\boldsymbol{E}$ pre 4-susednosť

$$E_1 = \begin{pmatrix} * & * & * \\ 0 & 1 & 0 \\ * & 0 & * \end{pmatrix}, E_2 = \begin{pmatrix} * & 0 & * \\ 0 & 1 & * \\ * & 0 & * \end{pmatrix}, \ldots$$

a pre 8-susednosť

$$E_1 = \begin{pmatrix} * & 1 & * \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, E_2 = \begin{pmatrix} 0 & * & * \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \ldots$$

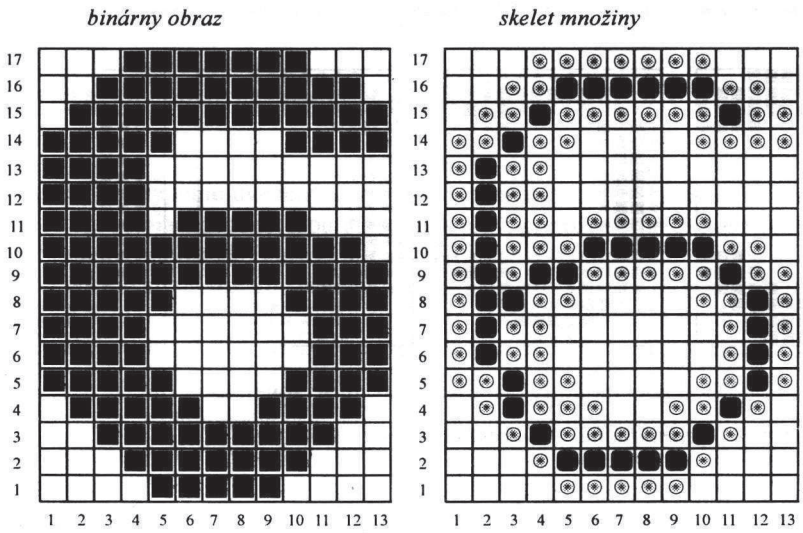
Na záver môžeme spomenúť, že morfologické transformácie sa využívajú aj pri náročnejších spracovaniach obrazu. Napríklad zavedie sa pojem podmienenej dilatácie [HLAV92]. Tento prístup sa využíva k oddeleniu objektov dotýkajúcich sa okna obrazu.

Druhým užitočným použitím podmienenej dilatácie je vyhľadanie dier objektov. Podobne sa zavedie konečná erózia, ktorá sa využíva na oddelenie prekrývajúcich sa konvexných častí.

9.3 Kostra (skelet) množiny

V tejto časti sa budeme zaoberať algoritmami, ktoré vytvárajú **kostru** (*skelet*) danej množiny zadanej v rastri (v digitálnej rovine). Skeletom množiny rozumieme jednorozmerné čiary, topologicky ekvivalentné so zadanou množinou a zachovávajúce jej geometrickú podstatu. Všeobecný postup týchto algoritmov je podobný ako pri morfologických transformáciach a spočíva v **stenčovaní zadanej množiny**. Na obrázku 9.6 vidíme ako môže vyzerať obraz zosnímaného čísla '6' a jeho skelet v rastri. Všimnime si, že sa číslica '6' stenčila, pričom sa zachovala súvislosť množiny i počet jej dier. Algoritmov skeletovania je známych niekoľko desiatok. Treba si uvedomiť, že ich výstupy sa líšia. Skelet v spojitom prípade sa definuje ako geometrické miesto stredov vpísaných kružníc s maximálnymi polomermi. Niet všeobecne prijatej definície skeletu v rastrovej rovine. Skelet je to, čo vyrobí skeletovací algoritmus.

Praktické využitie skeletu nachádzame hlavne pri rozpoznávaní obrazov (napríklad pri rozpoznávaní znakov) a pri kódovaní a prenose veľkého objemu grafických údajov. Po transformácii obrazu sa robí aj opačný postup - skelet sa **zosilňuje** (*expanduje*), aby užívateľ získal pôvodný obraz.



Obr. 9.6 Skelet binárneho obrazu - oblasti zosnímanej scannerom

9.4 Základné pojmy

Aby sme zachovali charakteristické geometrické črty danej množiny, všimnime si tri základné požiadavky na skelet:

1. Zachovanie súvislosti množiny. Ak je množina súvislá, potom aj skelet musí ostať súvislý. To znamená, že komponenty súvislosti musia ostať zachované pre danú množinu a tiež pre jej doplnok.

2. Zachovanie koncových bodov. V množine treba čo najskôr detekovať koncové body tak, aby sa neskrátili charakteristické línie.

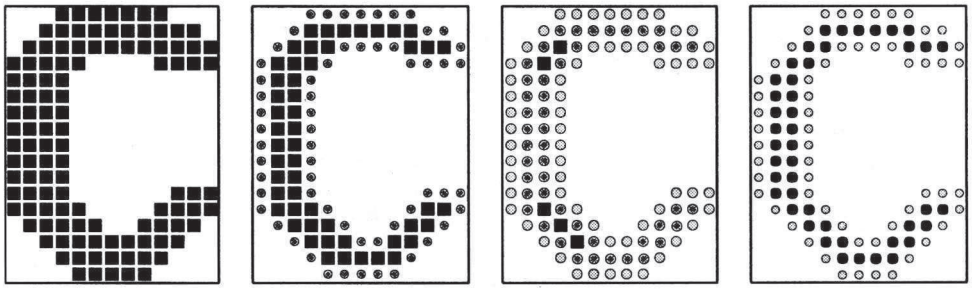
3. Zachovanie strednej línie. Pri stenčovaní hrubšej množiny je treba zachovávať približne stred t.j. stenčovať symetricky. Pretože pri spätnej rekonštrukcii množiny vytvoríme okolie skeletu, čo by mala byť množina podobná pôvodnej množine.

Len prvá uvedená požiadavka je topologického charakteru, ostatné dve sú geometrického. Prirodzene chceme, aby skelet minimalizoval počet bodov zadanej množiny. To znamená, jednoducho popísať tvar pomocou súvislých čiar. Pri konkrétnych aplikáciách vyžadujeme, aby výsledný skelet bol stabilný voči malým zašumeniam zosnímaneho objektu t.j. pri malej lokálnej zmene množiny by mal skelet zostať nezmenený alebo len málo zmenený. Väčšina algoritmov skeletovania vychádza z obrysu danej množiny.

Všimnime si nasledujúci jednoduchý postup stenčovania :

Jednoduchý algoritmus skeletu

1. Pre danú množinu vyhľadáme obrys oblasti.
2. Všetky obrysové body vylúčime, pokiaľ nenarušíme súvislosť.
3. Ak už sú všetky body obrysové, ukončíme postup.
   Inak opakujeme kroky 1 a 2.



a) b) c) d)

Obr. 9.7 Jednoduché skeletovanie pomocou obrysu

Na obrázku 9.7 b) vidíme, že pri prvej iterácii vyhľadania obrysu písmena 'C' je možné všetky obrysové body vylúčiť, pretože nenarušíme súvislosť množiny. Na obrázku 9.7 c) už nie je možné obrysové body vylúčiť, pretože sa jednak naruší súvislosť a tiež stratíme charakteristické črty písmena 'C'. Na obrázku 9.7 d) sme teda zachovali obrysové body, avšak na úkor hrúbky. Neskôr ukážeme, ako sa tomuto môžeme problému vyhnúť.

Aby sme zachovali predtým uvedené základné požiadavky skeletu, musíme zaviesť pojem skeletový bod (niekedy tiež nazývaný viacnásobný obrysový bod). Najprv si ešte

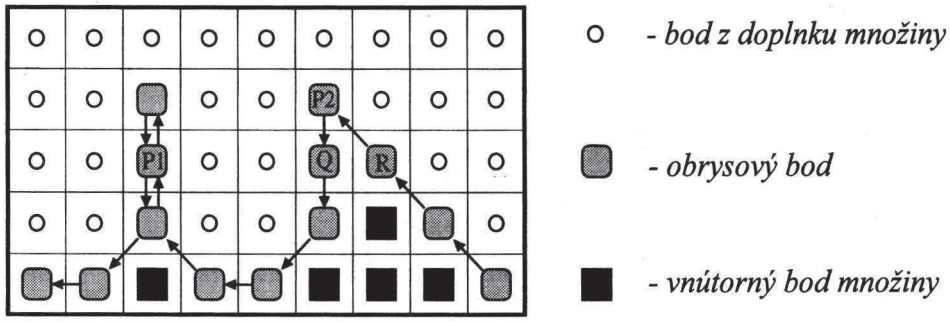
uvedieme definíciu obrysovej susednosti. Dva body nazveme obrysovo susedné, ak nasledujú bezprostredne za sebou v postupnosti obrysových bodov množiny (postupnosť obrysu chápeme cyklicky). Pod vnútrom množiny rozumieme všetky body množiny, ktoré nie sú obrysové.

Hovoríme, že bod A je skeletový, ak platí jedna z nasledujúcich podmienok:

1. Bod A sa vyskytuje v postupnosti obrysu dva razy, čiže je viacnásobný

2. Bod A nemá žiadnych susedov z vnútra množiny t.j. má len obrysových susedov a susedov z doplnku

3. Má aspoň jedného priameho suseda, ktorý je obrysovým bodom, ale nie je jeho obrysovým susedom.



Obr. 9.8 Ukážka bodov s uvedenými vlastňosťami 1, 2 a 3

Na lepšie pochopenie uvedených pojmov si všimnime obr. 9.8. Bod P1 spĺňa vlastnosť 1, pretože je viacnásobný. Bod P2 je skeletový, pretože spĺňa 2. podmienku, t.j. nemá suseda z vnútra oblasti. Body Q a R spĺňajú zase 3. podmienku, t.j. nie sú obrysovo susedné, a pritom sú priamymi susednými bodmi. Je zrejmé, že ak by sme vylúčili obidva body Q a R, potom by sme narušili súvislosť množiny s bodom P2. Pritom bod P2 je vhodné ponechať pre skelet, pretože sa môže chápať ako koncový bod množiny.

Overenie týchto podmienok je možné zistiť pri druhom prechode po obrysových bodoch množiny. Výhodnejšie však bude preformulovať uvedené podmienky na lokálne vlastnosti skúmaných obrysových bodov. Vyjadríme podmienky 1, 2 a 3 v okolí bodu pomocou masky 3×3.

Zavedieme nasledujúce označenie :

P vyšetrovaný bod, **0** bod z doplnku množiny (0), **A, B** bod má ľubovoľnú hodnotu, avšak jeden z rovnako označených bodov má nenulovú hodnotu, **1** vnútorný bod množiny (1), **2** obrysový bod, cez ktorý prejdeme len raz, **2+** obrysový bod, cez ktorý prejdeme aspoň 2 razy, **1'** bod s hodnotou rôznou od **1.**

A	A	A
0	P	0
B	B	B

A	A	A
A	P	0
A	0	B

Obr. 9.9 Vyjadrenie podmienky 1 ako lokálnej vlastnosti bodu P

Z 1. podmienky vyplýva, že cez obrysový bod prejdeme aspoň dvakrát, a preto aspoň dvaja priami susedia vyšetrovaného bodu P musia mať hodnotu nula. Na obrázku 9.9 sú znázornené takéto situácie. Ak pripustíme pre tieto konfigurácie otočenie o násobky 90 stupňov, tak dostávame pre 1. podmienku všetky možné lokálne konfigurácie.

1'	1'	1'
1'	P	1'
1'	1'	1'

Obr. 9.10 Okolie nevnútorných bodov - podmienka 2

Z 2. podmienky vyplýva, že obrysový bod má okolie ako je znázornené na obrázku 9.10, čiže v okolí nie sú vnútorné body. Táto konfigurácia sa overí ľahko.

Podrobnejším skúmaním môžeme zistiť, že 3. podmienka sa dá charakterizovať konfiguráciou znázornenou na obrázku 9.11, pričom pripúšťame otočenie o násobky 90 stupňov. Pritom na body sú kladené tieto požiadavky. Aspoň jeden z bodov C musí byť nenulový. Ak sú oba nulové, potom A aj B môžu byť ľubovoľné. Inak aspoň jeden z dvojice bodov A (rovnako aj B) musí byť nenulový.

A	A	C
0	P	2+
B	B	C

Obr. 9.11 Okolie nesusedných bodov hranice - podmienka 3

Preformulovaním podmienok 1, 2 a 3 do masiek 3×3 sme získali spôsob, ako vyšetriť skeletovosť bodu priamo bez dvojitého obchádzania obrysu. Naviac takýto prístup dáva možnosť paralelného spracovania na počítači.

9.5 Algoritmy skeletovania

Algoritmy skeletovania môžeme rozdeliť do dvoch skupín podľa toho, či využívajú obrys množiny alebo nie.

1. V algoritmoch využívajúcich obrys môžeme sledovať dve etapy: v prvej etape sa zostrojí obrys a v druhej etape sa hľadajú vhodné podmienky na obrysové body, ktoré je možné vylúčiť. Takým je klasický algoritmus skeletovania a jeho rôzne modifikácie, napr. Kwockov algoritmus.

2. Do druhej skupiny algoritmov patria tie, ktoré sa snažia priamo vyhľadávať skeletové body, pričom sa ostatné body postupne vylučujú. Reprezentantom tejto skupiny je algoritmus Zhanga-Suena.

9.5.1 Klasický algoritmus skeletovania

Tento algoritmus patrí do 1. skupiny algoritmov, ktorý využíva vyššie uvedené lokálne charakteristiky skeletových bodov. Na obrázku 9.7.c) vidíme, že skoro všetky obrysové body spĺňajú 2. podmienku skeletových bodov t.j. nemajú vnútorných susedov, a preto ostávajú v skelete. To však má za následok to, že môže vzniknúť skelet s hrúbkou dvoch obrazových bodov ako na obrázku 9.7 d). Preto je algoritmus navrhnutý tak, aby vždy v jednom cykle (pozri kroky 5-12 uvedené v algoritme) sa uvažovali len hraničné obrysové body z jedného smeru vzhľadom na doplnok množiny (t.j. postupne sprava, zľava, zhora a zdola). Algoritmus sa preto musí opakovať pre všetky štyri smery, pokiaľ sa ešte obrysové body odstraňujú. To zhoršuje časovú náročnosť algoritmu. Jediná výhoda algoritmu spočíva vo využití lokálnych charakteristík skeletových bodov. Preto sa tento algoritmus dá ľahko previesť na vhodný typ paralelného počítača.

Masky znázornené na obrázkoch 9.9, 9.10 a 9.11 označíme ako *Loc_Sk*. Flag *remain* indikuje, že pri spracovávaní v cykle (kroky 5-12) sme niektoré obrysové body ešte vylúčili. Flag *skel* indikuje nájdenie skeletového bodu.

Algoritmus Skelet;

```
Procedure Skelet_1;

begin
1.  remain:= true;
2.  while remain = true do                    { boli odstránené obrysové body ? }
    begin
3.      remain:= false;
4.      for j:= 0, 2, 4, 6 do                 { pre jednotlivé smery obrysu }
        begin
5.          for pre všetky body P do
            begin
```

```
6.              if h[P] = 1  and  neighb_h (P, j) = 0  then      { hraničný bod zo smeru j }
                begin
7.                  skel:= false;
8.                  for pre všetky locS z Loc_Sk do
9.                      if loc(P) = locS  then  skel:= true;
10.             if skel = true    then h[P]:= 2                      { skeletový bod }
                                  else  h[P]:= 3                   { bod na vylúčenie }
                end
            end
11.     for pre všetky body P do
12.         if h[P] = 3 then begin
                                  h[P]:=0;                         { vylúčime obrysové body }
                                  remain:= true;
                              end;
        end { koniec 4. for ... }
13. end { koniec while ... }
end.
```

9.5.2 Algoritmus Zhanga-Suena

Tento algoritmus vyhľadáva obrys množiny prehľadaním všetkých bodov oblasti, v ktorej je množina definovaná. Namiesto skeletových bodov obrysu vyhľadáva prebytočné obrysové body, a preto patrí do druhej skupiny algoritmov. Najmenšie okolie vyšetrovaného bodu cyklicky označíme 0,..., 7.

Algoritmus sa dá rozdeliť na dve časti. V prvej časti sa obrysový bod P stáva prebytočný, ak spĺňa nasledovné štyri vlastnosti :

V.1 $2 \le neighb_h(P, 0) + ... + neighb_h(P, 7) \le 6$, (9.1)

t.j. aby počet jeho susedov z danej množiny bol od dvoch do šiestich. Ak má iba jedného suseda, ide o koncový bod, a ak má sedem, potom je to skôr vnútorný bod a nebude sa zatiaľ vylučovať.

Ďalej musí platiť, že v najmenšom okolí bodu je od 0-suseda po 7-suseda cyklicky počet 0-1 prechodov rovný jednej. Označíme tento počet $numb01(P)$. Táto podmienka napríklad nie je splnená pre líniu s jednotkovou hrúbkou, čo vlastne bude zaručovať súvislosť množiny,

V.2 $numb01(P) = 1$. (9.2)

Tretia a štvrtá podmienka skúmajú absenciu aspoň jedného suseda z nasledujúcich množín susedov { 0, 2, 6 } a { 0, 4, 6 }, t.j.

V.3 $neighb_h(P, 0) \,.\, neighb_h(P, 2) \,.\, neighb_h(P, 6) = 0$

a

V.4 $neighb_h(P, 0) \,.\, neighb_h(P, 4) \,.\, neighb_h(P, 6) = 0.$

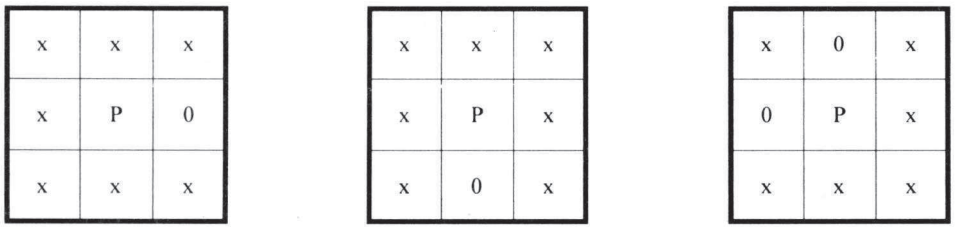
Tieto podmienky sa dajú previesť na ekvivalentné tak, aby platila aspoň jedna podmienka :

$neighb_h(P, 0) = 0$ alebo $neighb_h(P, 6) = 0$ (9.3a)

alebo

$$neighb_h(P, 2) + neighb_h(P, 4) = 0. \qquad (9.4a)$$

Vidíme, že v prvom prechode sa vyšetrujú pravé alebo dolné (resp. ľavo-horné) obrysové body, ako je to vidieť na obr. 9.12.



Obr. 9.12 Okolie vyšetrovania hraničných bodov

V druhej časti sa podmienky (9.1) a (9.2) nezmenia a posledné dve podmienky sa nahradia podobnými podmienkami :

V.3' $neighb_h(P, 0) \,.\, neighb_h(P, 2) \,.\, neighb_h(P, 4) = 0$

a

V.4' $neighb_h(P, 2) \,.\, neighb_h(P, 4) \,.\, neighb_h(P, 6) = 0.$

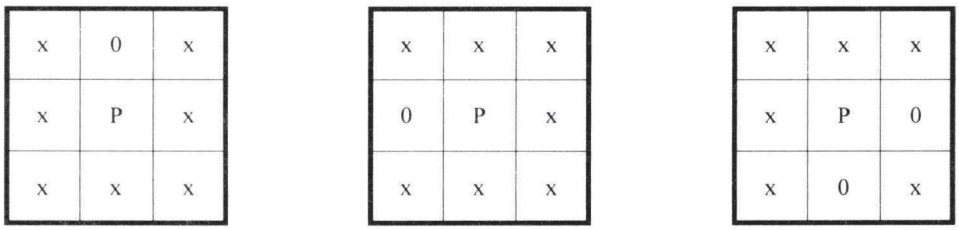
Môžeme modifikovať podmienky **V.3'** a **V.4'** na podobne ako (9.3a) a (9.4a) tak, aby platila aspoň jedna podmienka:

$neighb_h(P, 2) = 0$ alebo $neighb_h(P, 4) = 0$

alebo

$$neighb_h(P, 0) + neighb_h(P, 6) = 0.$$

Vidíme, že v druhom prechode sa vyšetrujú horné alebo ľavé (resp. pravo-dolné) obrysové body, ako je to vidieť na obr. 9.13.



Obr. 9.13 Okolie vyšetrovania hraničných bodov

Popíšeme algoritmus Zhan-Suena. Flag *remain* indikuje, že pri spracovávaní v cykle (kroky 3-11) mohli ešte prípadne zostať neskeletové body pre vylúčenie.

Algoritmus Zhang-Suen;

```
Procedure Zhang-Suen;
begin
1.  remain:= true;
2.  while remain = true do                          { boli odstránené hraničné body ? }
    begin
3.      remain:= false;
4.      for pre všetky body P do                    { dolné pravé body hranice }
        begin
5.          if h[P] = 1 and
                ( 2 < neighb_h(P, 0) + ... + neighb_h(P, 7) < 6 )   and      {V.1}
                numb01(P) = 1                                       and      {V.2}
                neighb_h(P, 0) * neighb_h(P, 2) * neighb_h(P, 6) = 0  and    {V.3}
                neighb_h(P, 0) * neighb_h(P, 4) * neighb_h(P, 6) = 0  then   {V.4}
                h[P]:= 3;
        end
6.      for pre všetky body P do
7.          if h[P] = 3 then begin
                                h[P]:=0;
                                remain:= true;
                             end
8.      for pre všetky body P do                    { horné ľavé body obrysu }
        begin
9.          if h[P] = 1 and
                ( 2 < neighb_h(P, 0) + ... + neighb_h(P, 7) < 6 )   and      {V.1}
                numb01(P) = 1                                       and      {V.2}
                neighb_h(P, 0) * neighb_h(P, 2) * neighb_h(P, 4) = 0  and    {V.3'}
                neighb_h(P, 2) * neighb_h(P, 4) * neighb_h(P, 6) = 0  then   {V.4'}
                h[P]:= 3;
        end
10. for pre všetky body P do
11.     if h[P] = 3 then begin
                            h[P]:=0;                { vylúčime hraničný bod množiny }
                            remain:= true;
                         end
12. end                         { koniec while ... }
end.
```

9.5.3 Algoritmus Kwocka

Tento algoritmus vychádza z generovania obrysu a je modifikáciou klasického algoritmu. Základná myšlienka spočíva v tom, ako vytvoriť z daného obrysu nový obrys a pritom zachovať niektoré skeletové body.

Predpokladajme, že sme našli reťazec obrysových bodov, popísaných prvým bodom a postupnosťou kódov (t.j. smermi pohybu). Napríklad na obr. 9.8 obrys danej množiny je daný bodom R a postupnosťou kódov:

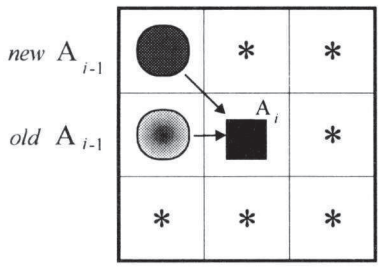
3, 6, 6, 5, 4, 3, 2, 2,...

Budeme uvažovať o rozdiele kódov, ktorý označíme ako $dir_i = c_i - c_{i-1}$.

Ak máme na obryse určené skeletové body, môžu nastať tieto prípady:

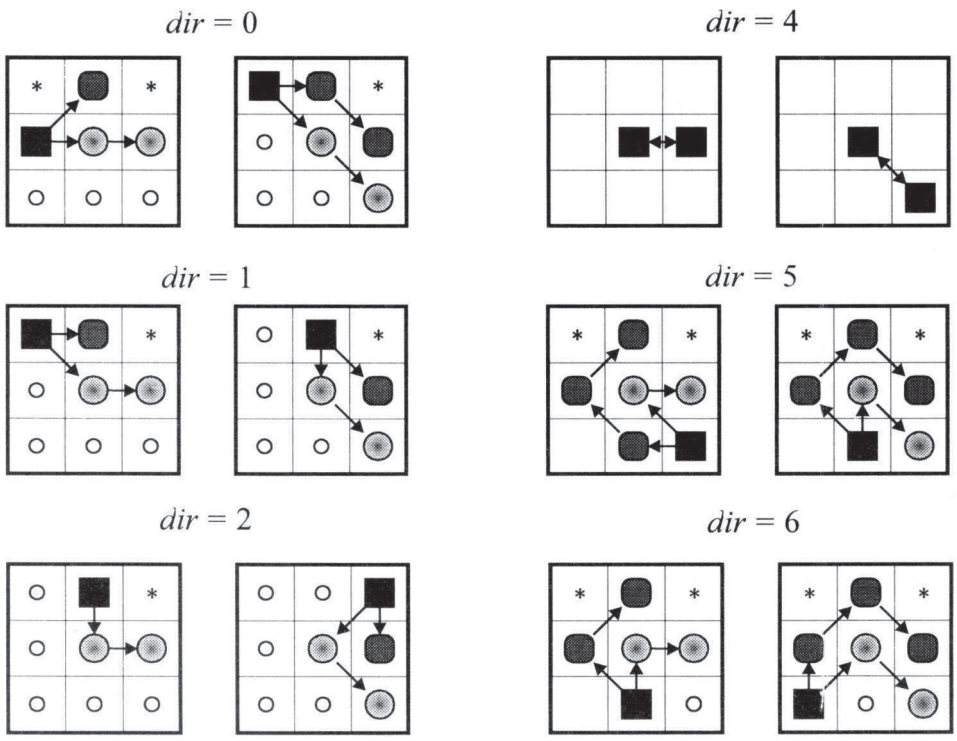
1) Ak máme v postupnosti za sebou dva skeletové body A_{i-1} a A_i , potom v ďalšej iterácii ostanú zachované tieto body aj smer dir_i .

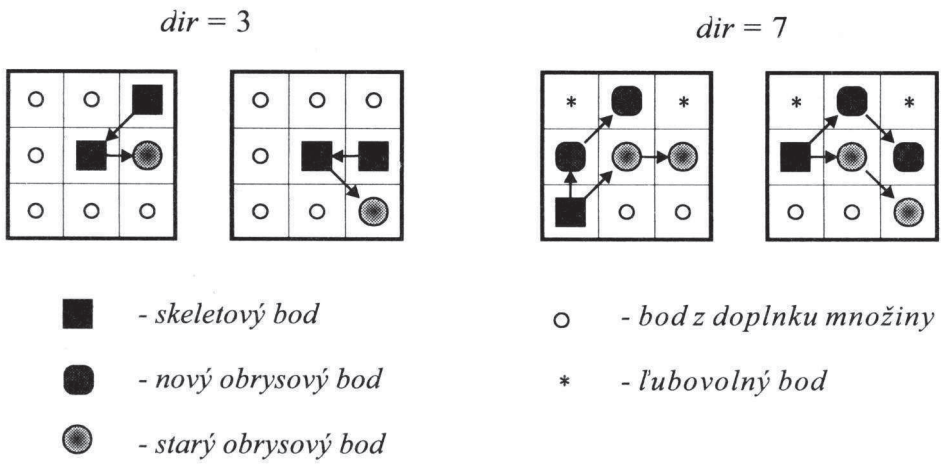
2) Bod A_i je skeletový a A_{i-1} nie je, potom sa určí nový obrysový bod A_{i-1}, ako je vidieť z obr. 9.14 a smer $dir = dir + 7$ (modulo 8).



Obr. 9.14 Zadanie nového obrysu

3) Bod A_{i-1} je skeletový a A_i nie je, potom sa určia nové body obrysu a tiež ich zodpovedajúce smery podľa nasledujúcej tabuľky. Smer *dir* sa berie (modulo 8).





Obr. 9.15 Zadanie nového obrysu

Z tabuľky znázornenej na obrázku 9.15 vidíme, že začiatočný bod bude vychádzať zo skeletového bodu. Sú tu uvedené pre úplnosť aj prípady, ktoré nemôžu nastať.

4) Obidva body A_{i-1} a A_i nie sú skeletové, potom sa určia nové body obrysu a tiež ich odpovedajúce smery podobnou tabuľkou ako je to na obr.9.15. Pri vytváraní tabuľky pre tento prípad si všimnime, že začiatočný bod bude podobne závislý od smeru.

V algoritme si musíme všímať uzavreté reťazce obrysu. Nový reťazec sa začína v začiatočnom bode reťazca, v ktorom bolo ukončené generovanie z minulého reťazca. Obrys pokračuje v začiatočnom bode a prechádza proti smeru hodinových ručičiek po priamych susedoch. Zjednodušene môžeme zapísať Kwockov algoritmus nasledovne :

Algoritmus Kwocka;

```
begin
1. tracer;                                  { vytvor obrys niektorým algoritmom }
2. queue;                                   { a zaraď do fronty }
3. while fronta je neprázdna do             { boli odstránené zbytočné obrysové body ? }
   begin
4.     { detekuj prvý reťazec a inicializuj 1. bod }
5.     while obrys nie je spracovaný do
           begin
6.         { vygeneruj podľa tabuľky nový reťazec obrysu a zaraď ho do fronty obrysov }
           end
   end
end.
```

10

Metódy modelovania a zobrazovania trojrozmerných objektov

10.1 Úvod

V mnohých aplikáciách počítačovej grafiky treba pracovať v trojrozmernom priestore. Od druhej polovice 60. rokov sa skúmali možnosti zobrazovania na počítači. Hlavný smer výskumu grafiky sa orientoval k automatizovaným systémom projektovania (CAD - Computer Aided Design), ktoré umožnovali navrhovať rôzne súčiastky aj riadiť výrobu pomocou počítača (CAM - Computer Aided Manufacturing). Mnohé systémy umožnovali výstup návrhu aj zobrazení trojrozmerných scén. Avšak aj v súčasnosti užívateľ návrhových systémov musí čakať na reálne zobrazenie zložitej scény aj niekoľko hodín. Pri vytváraní obrazu je niekedy dôležité zobrazovať nielen realisticky, ale tiež dostatočne rýchle. Napríklad na počítačovom trenažéri je treba vytvoriť až 30 snímkov za sekundu, aby vznikol dojem skutočného pohybu (napr. pre letový simulátor) .

Vhodný spôsob modelovania objektov v počítači je dôležitý pre výsledné zobrazenie. Často aj kvalita zobrazenia súvisí s kvalitou modelu (reprezentácie) objektov. Pri spracovávaní telesa nás zaujíma popis telesa a potom jeho reprezentácia v počítači. Pri modelovaní telies používame tri základné spôsoby: hraničnú reprezentáciu, pomocou CSG stromu (Constructive Solid Geometry) a objemovú (voxlovú) reprezentáciu.

Úplne reálne zobrazenie s najjemnejšími podrobnosťami, aké vidíme v skutočnosti okolo seba, možno realizovať na počítači veľmi ťažko. Cieľom je však zobraziť aspoň toľko informácií, aby operátor model pochopil a mohol s ním pracovať. Najväčšie ťažkosti pri zobrazovaní priestorových objektov sú v tom, že všetky prakticky dostupné zariadenia zobrazujú dvojrozmerne. V ďalšom rozoberieme možnosti znázornenia trojrozmernej informácie.

10.2 Premietanie

V tejto časti popíšeme spôsoby zobrazovania trojrozmerných objektov. Pojem premietania sa v počítačovej grafike používa práve na transformáciu trojrozmerného priestoru (3D) do dvojrozmerného (2D). **Priemetňa** je rovina v priestore, do ktorej

transformujeme (premietame) objekty. Pri zobrazovaní 3D objektov musíme zadať spôsob premietania do priemetne tak, aby sme ich mohli zobraziť na výstupnom zariadení. Uvedieme dva spôsoby premietania: rovnobežné a stredové. Pri vyjadrovaní súradníc bodov v rovine alebo v priestore niekedy stotožníme homogénne súradnice s euklidovskými súradnicami.

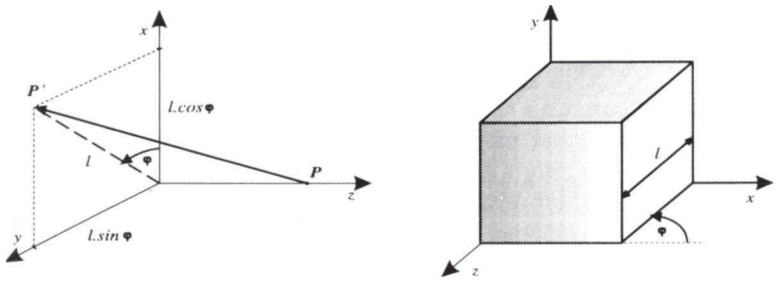
10.2.1 Rovnobežné premietanie

Rovnobežné premietanie je zadané priemetňou a smerom premietania (vektorom), ktorý nesmie byť rovnobežný s priemetňou. Priemet daného bodu do priemetne zostrojíme tak, že vedieme priamku týmto bodom a smerom premietania (t.j. preložíme premietací lúč cez daný bod), a potom priesečník premietacieho lúča s priemetňou definuje priemet daného bodu. Podľa smeru premietania delíme rovnobežné premietanie na kolmé premietanie (t.j. smer premietania je kolmý na priemetňu) a šikmé premietanie (všeobecné rovnobežné premietanie). Neskôr ukážeme, že jednoduchým spôsobom môžeme transformovať priestor tak, aby priemetňa bola zhodná s rovinou xy. Kolmé premietanie do roviny xy sa zadáva jednoduchými rovnosťami. Bodu $P = (x, y, z)$ zodpovedá v priemete bod $P' = (x', y')$, kde $x' = x$ a $y' = y$.

Zodpovedajúca matica kolmého **rovnobežného premietania** do roviny xy je:

$$M_{ort} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Pre odvodenie rovníc **šikmého premietania** predpokladajme, že smer premietania je zadaný bodom $P = (0, 0, 1)$ a jeho priemetom v rovine xy bodom $P' = (l.\cos\varphi, l.\sin\varphi, 0)$, kde l a φ sú polárne súradnice bodu P' v rovine xy. Na obr. 10.1 a) máme znázornený vektor PP' a uhol φ. Parameter l nám určuje dĺžku natiahnutia pre os z v priemetni xy a uhol φ je uhol od osi x. Ak parameter $l = 0$, ide o kolmé premietanie.



a) b)

Obr.10.1 Šikmé (kosouhlé) premietanie zadané dvoma bodmi P a P'

Na vyjadrenie priemetu ľubovoľného bodu (x, y, z) do roviny xy zapíšeme parametrické vyjadrenie priamky určenej daným bodom a smerovým vektorom P'- P. Priesečník tejto priamky s rovinou $z = 0$ vhodne vyjadruje šikmé premietanie:

$$x' = x + z.(l.\cos\varphi),$$

$$y' = y + z.(l.\sin\varphi).$$

Šikmé premietanie do roviny xy môžeme vyjadriť nasledujúcou maticou:

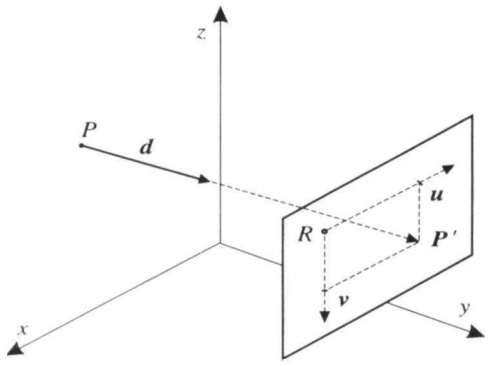
$$M_{kos} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ l\cdot\cos\varphi & l\cdot\sin\varphi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Matica **kosouhlého priemetu** je podobná matici skosenia v priestore. Skutočne môžeme najprv realizovať skosenie v smere roviny xy, a potom kolmé premietanie do roviny xy. Pretože skosenie zachová vzdialenosť v rovinách rovnobežných s rovinou xy, zachová sa vzdialenosť aj pri výslednom šikmom premietaní. Na obr. 10.1 b) vidíme, že steny kocky sa premietajú tak, že predná a zadná stena sú zhodné so svojimi pôvodnými stenami a hrany spájajúce tieto stieny zvierajú s osou x uhol φ.

Pri **vojenskej perspektíve** je uhol medzi smerom premietania a priemetňou 45 stupňov a preto $l = 1$. Pri šikmom premietaní, keď $l = 1/2$ je uhol medzi smerom premietania a priemetňou arctg(2), čo je približne 63.4 stupňa. Takéto premietanie nazývame **kabinetné**.

Veta 10.1. Nech pre rovnobežné premietanie je priemetňa daná referenčným bodom R a vektormi $\boldsymbol{u}$ a $\boldsymbol{v}$ a smer premietania je zadaný vektorom $\boldsymbol{d}$. Nech ľubovoľný bod $P = (x, y, z)$ v sústave súradníc xyz má priemet bod $P' = (x', y')$ v sústave súradníc priemetne R, $\boldsymbol{u}$, $\boldsymbol{v}$, potom pre súradnice priemetu platia nasledujúce rovnosti

$$x' = \frac{(P-R)\cdot(v\times d)}{d\cdot(u\times v)}, \qquad y' = -\frac{(P-R)\cdot(u\times d)}{d\cdot(u\times v)}.$$



Obr. 10.2. Priemet bodu pri šikmom rovnobežnom premietaní

Dôkaz. Na obrázku 10.2 sa body P a P' sa dajú vyjadriť vzhľadom k dvom sústavám súradníc a platí medzi nimi nasledujúci vzťah:

$$P' = R + x'.\boldsymbol{u} + y'.\boldsymbol{v} = P - z'.\boldsymbol{d} .$$

Poslednú rovnosť môžeme prepísať na rovnosť vektorov:

$$P - R = x'.\boldsymbol{u} + y'.\boldsymbol{v} + z'.\boldsymbol{d} .$$

Ak rovnosť vynásobíme skalárne vektorom $(\boldsymbol{v} \times \boldsymbol{d})$, potom dostaneme:

$$(P - R).(\boldsymbol{v} \times \boldsymbol{d}) = x'.\ \boldsymbol{u}.(\boldsymbol{v} \times \boldsymbol{d}),$$

odkiaľ jednoduchou úpravou vyjadríme požadovanú súradnicu x'. Ak vynásobíme skalárne vektorom $(\boldsymbol{u} \times \boldsymbol{d})$, potom dostaneme pre hľadanú súradnicu y':

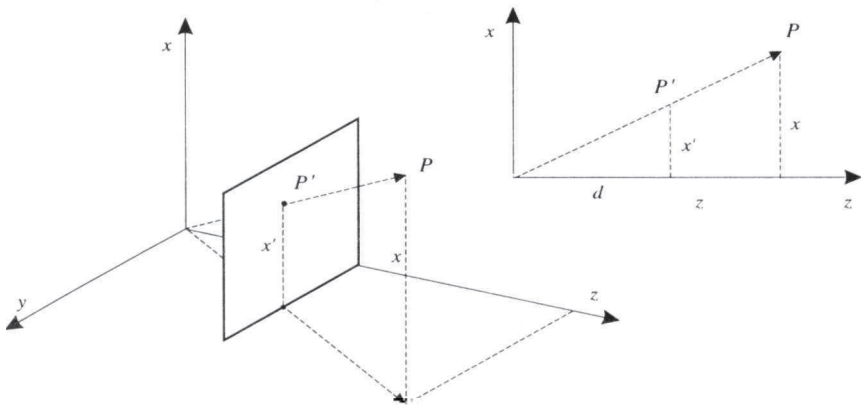
$$(P - R).(\boldsymbol{u} \times \boldsymbol{d}) = y'.\ \boldsymbol{v}.(\boldsymbol{u} \times \boldsymbol{d}).$$

10.2.2 Stredové premietanie

Stredové premietanie je zadané priemetňou a stredom premietania. Pre každý bod, mimo bodov roviny obsahujúcej stred premietania a zároveň rovnobežnej s priemetňou, vieme zostrojiť jeho priemet do priemetne takto: daným bodom a stredom premietania preložíme priamku a jej priesečník s priemetňou je priemetom daného bodu. Pre analytické vyjadrenie stredového premietania budeme predpokladať, že priemetňa je kolmá na os z.

Najprv predpokladajme, že priemetňa je vo vzdialenosti d od začiatku sústavy súradníc a jej analytický zápis je $z = d$. Stred premietania umiestníme do začiatku sústavy súradníc. Z obrázku 10.3 vidíme na základe podobnosti trojuholníkov, že pre súradnice bodu (x, y, z) a priemetu (x', y', d) platia nasledujúce rovnosti:

$$x' = d/z\ .\ x, \qquad y' = d/z\ .\ y.$$



Obr. 10.3. Stredové premietanie so stredom S = (0, 0, 0)

Stredové premietanie môžeme vyjadriť nasledujúcou maticou:

$$M_{cen} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

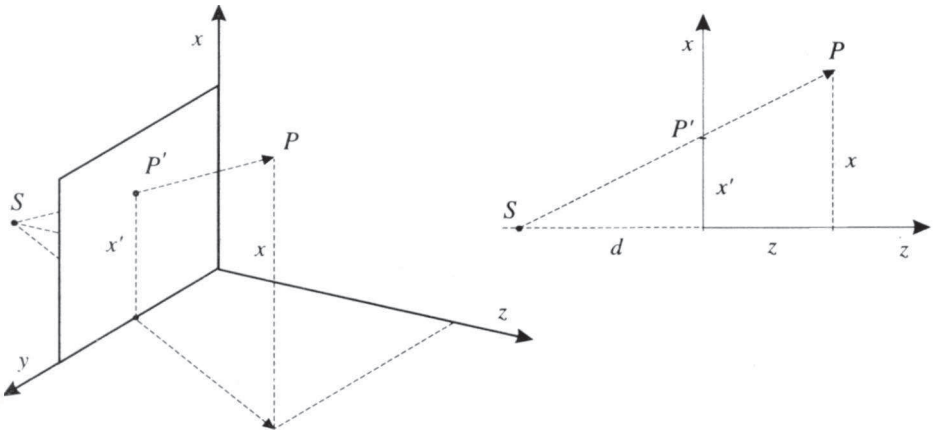
Posledný zápis matice pre stredové premietanie si overíme. Bod $P = (x, y, z, 1)$ má svoj priemet bod $P^*\boldsymbol{M}_{cen}$ a vypočítané súradnice:

$$P^*\boldsymbol{M}_{cen} = (x, y, z, z/d) = (d/z.x, d/z.y, d, 1).$$

Využijeme nasledujúcu vlastnosť homogénnych súradníc:

$$(x, y, z, 1) = (kx, ky, kz, k).$$

Homogénne súradnice môžeme násobiť skalárom d/z a upraviť na práve uvedený tvar, čo zodpovedá zápisu pre stredové premietanie uvedenému predtým.



Obr. 10.4. Stredové premietanie so stredom S = (0, 0, -d).

Pri inom spôsobe vyjadrenia stredového premietania stred umiestnime do bodu $(0, 0, -d)$ a priemetňu stotožníme s rovinou xy. Z obrázku 10.4 vidíme z podobnosti trojuholníkov, že pre súradnice bodu (x, y, z) a priemetu (x', y', d) platí:

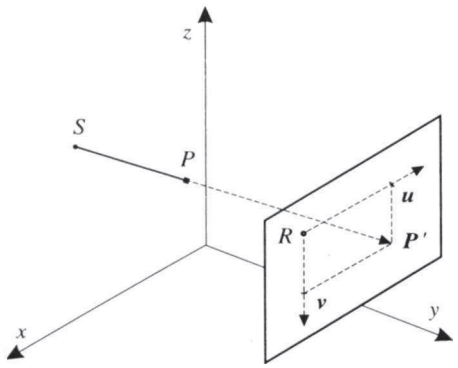
$$x' = d/(z+d) \,.\, x\,, \quad y' = d/(z+d) \,.\, y\,.$$

Takéto stredové premietanie môžeme vyjadriť nasledujúcou maticou:

$$M_{cen} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Veta 10.2. Nech pre stredové premietanie je priemetňa zadaná referenčným bodom R a vektormi $\boldsymbol{u}$ a $\boldsymbol{v}$ a stred premietania bodom S. Nech ľubovoľný bod $P = (x, y, z)$ v sústave súradníc xyz má priemet $P' = (x', y')$ v rovine $R\ \boldsymbol{u}\ \boldsymbol{v}$, potom pre súradnice priemetu platí:

$$x' = \frac{(P-S)\cdot(v\times(R-S))}{(P-S)\cdot(u\times v)}, \qquad y' = -\frac{(P-S)\cdot(u\times(R-S))}{(P-S)\cdot(u\times v)}.$$



Obr. 10.5. Priemet bodu pri stredovom premietaní.

Dôkaz. Na obrázku 10.5 možno body P a P' vyjadriť v dvoch sústavách súradníc a platí medzi nimi nasledujúci vzťah:

$$P' = R + x'.\boldsymbol{u} + y'.\boldsymbol{v} = z'.P + (1\text{-}z').S\,.$$

Poslednú rovnosť môžeme prepísať na rovnosť vektorov:

$$S - R = x'.\boldsymbol{u} + y'.\boldsymbol{v} + z'.(S\text{-}P)\,.$$

Ak vynásobíme skalárne vektorom $(\boldsymbol{v} \times (P\text{-}S))$, potom dostaneme:

$$(S - R).(\boldsymbol{v} \times (P\text{-}S)) = x'.\ \boldsymbol{u}.(\boldsymbol{v} \times (P\text{-}S)),$$

odkiaľ jednoduchou úpravou vyjadríme požadovanú súradnicu x'. Ak vynásobíme skalárne vektorom $(\boldsymbol{u} \times (P\text{-}S))$, potom dostaneme:

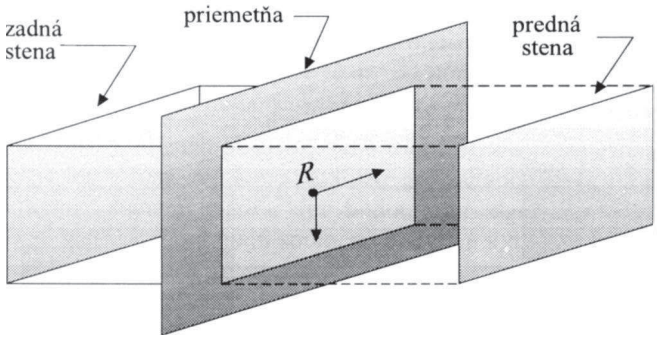
$$(S - R).(\boldsymbol{u} \times (P\text{-}S)) = y'.\ \boldsymbol{v}.(\boldsymbol{u} \times (P\text{-}S)),$$

odkiaľ opäť jednoduchou úpravou vyjadríme súradnicu y'.

10.2.3 Zobrazenie na kanonický tvar

V tejto časti ukážeme, ako môžeme definovať v priestore viditeľný objem. Videteľný hranol budeme transformovať do kvádra, ktorý bude mať hrany rovnobežné s osami x, y, z. Priemetňa je definovaná referenčným bodom R a normálovým vektorom $\boldsymbol{n}$. V projekčnej rovine zavedieme dva vektory $\boldsymbol{u}$ a $\boldsymbol{v}$, ktoré s bodom R vytvárajú sústavu

súradníc v priemetni. V tejto sústave súradníc zadáme 2D okno (*u_min, u_max, v_min, v_max*), pomocou ktorého definujeme viditeľný objem (pozri obr.10.6). Pre rovnobežné premietanie vytvárame pomocou smeru premietania nekonečný štvorboký hranol nad definovaným oknom. Aby sme dostali konečný viditeľný priestor, zadávame okrem okna aj dve roviny rovnobežné s priemetňou, a to prednú a zadnú orezávaciu rovinu (obr. 10.6).



Obr. 10.6 Zadanie oriezávacieho viditeľného hranola

Ukážeme, ako transformovať viditeľný hranol na kanonický tvar kvádra, t.j. vektor ***u*** do osi *x* a vektor ***v*** do osi *y*. Postupujeme podobne ako v časti 2.11. Zobrazenie rozložíme na tieto postupné transformácie:

1. posunutie referenčného bodu *R* do začiatku sústavy súradníc;

2. otočenie okolo osi *y* tak, aby obraz vektora ***u*** ležal v rovine *xy*;

3. otočenie okolo osi *z* tak, aby výsledný obraz vektora ***u*** ležal na kladnej osi *x*;

4. otočenie okolo osi *x* tak, aby sa vektor ***v*** stotožnil s osou *y*;

5. prípadná súmernosť podľa roviny *xy*, aby ľavotočivá sústava súradníc prešla do pravotočivej;

6. vyrovnanie šikmého 4-bokého hranola do kvádra.

Pretože 2. a 3. krok je podobný postupu v kapitole 2 pre 3D transformácie, uvedieme len označenie transformačných matíc. Podrobnejšie si rozpíšeme posledné tri kroky.

1. Transformačnú maticu posunutia bodu *R* označíme T.

2. Transformačnú maticu otočenia okolo osi *y* o uhol a označíme R_y.

3. Otočenie okolo osi *x* o uhol β zapíšeme maticou R_z.

4. Pre otočenie okolo osi *x* o uhol otočenia γ platia nasledujúce rovnosti:

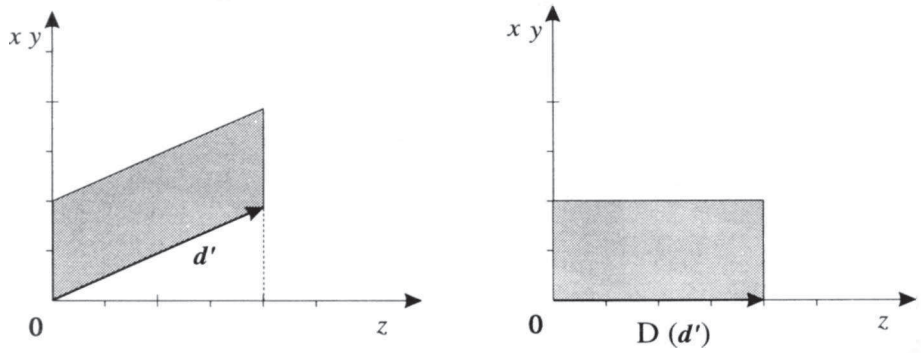
$$\cos\gamma = \boldsymbol{v}'_x / |\boldsymbol{v}| \quad \text{a} \quad \sin\gamma = \boldsymbol{v}'_y / |\boldsymbol{v}|,$$

kde $\boldsymbol{v}'$ je obraz vektora $\boldsymbol{v}$ pri transformácii $\mathrm{T}*\mathrm{R}_y*\mathrm{R}_z$ a $|\boldsymbol{v}|$ je dĺžka vektora $\boldsymbol{v}$.

Príslušnú transformačnú maticu otočenia označíme R_x.

5. Súmernosť podľa roviny *xy* sme uviedli v predchádzajúcej kapitole a označíme jej transformačnú maticu S_{xy}.

6. Ak smer premietania je daný vektorom ***d*** a zobrazuje sa pomocou transformácií $T*R_y*R_z*R_x*S_{xy}$ do vektoru ***d'***, potom potrebujeme uskutočniť takú transformáciu (skosenie), pri ktorej vektor ***d'*** prejde do vektora na osi *z* (pozri obrázok 10.7):



*Obr. 10.7. Transformovanie (skosenie) vektora **d'** do osi z*

Stačí použiť nasledujúcu transformačnú maticu

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -d'_x/d'_z & -d'_y/d'_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

pomocou ktorej sa ***d'***zobrazí do $d' * D = (0, 0, d'_z, 1)$ na osi *z*.

10.2.4 Orezávanie v priestore

Priestorový algoritmus orezávania je zovšeobecnením algoritmu pre orezávanie úsečiek Cohen-Sutherlanda, uvedeného v časti 3.3. Predpokladáme, že viditeľný objem je daný kvádrom o rozmeroch *a*, *b*, *c*, ktorého tri hrany ležia na osiach a jeden vrchol v začiatku sústavy súradníc. Každému bodu priradíme 6-bitový kód podľa pravidla, či daný bod leží v príslušnom polpriestore pre danú stenu kvádra alebo je mimo neho. Pravidlo charakterizuje polohu bodu vzhľadom ku kvádru takto:

1. bit = 1, ak bod leží zľava od kvádra t.j. $x < 0$;
2. bit = 1, ak bod leží sprava od kvádra t.j. $x > a$;
3. bit = 1, ak bod leží nižšie od kvádra t.j. $y < 0$;

4. bit = 1, ak bod leží vyššie od kvádra t.j. $y > b$;

5. bit = 1, ak bod leží zozadu za kvádrom t.j. $z < 0$;

6. bit = 1, ak bod leží spredu pred kvádrom t.j. $z > c$.

Ak nie je splnená podmienka, potom príslušný bit je nulový. Bod leží v kvádri, ak jeho kód je rovný (000000). Úsečka leží v kvádri, ak obidva jej koncové body majú kód (000000). Ak je celá úsečka dole, hore, vľavo, vpravo, vpredu alebo vzadu od kvádra, potom je celá úsečka mimo a podľa nastavených bitov kódu koncových bodov sa ľahko identifikuje, pretože je nastavený bit na rovnakom mieste. V týchto prípadoch stačí pre kódy bodov overiť, či ich logický súčin je rôzny od kódu 000000. Ak je logický súčin kódov rovný 000000, tak úsečku nemôžeme vylúčiť z orezávania.

Všetky ostatné úvahy sú analogické ako sme robili pri dvojrozmernom prípade orezávania. Rozdiel je len v tom, že máme parametrické zadanie úsečky a prienik robíme postupne so všetkými 6 stenami kvádra. Nevýhodou tohto algoritmu je, že môže niektoré úsečky orezávať šesťkrát, hoci majú spoločné body len s dvoma stenami viditeľného objemu.

10.3 Modelovanie 3D objektov

V kapitole 4 "Krivky a plochy" sme popísali možnosti vytvárania plôch, ktoré však nutne nemuseli uzatvárať objem. V mnohých aplikáciach však potrebujeme vytvoriť skutočné trojrozmerné teleso, aby sme preň mohli zisťovať iné technické a fyzikálne charakteristiky, napríklad ťažisko. Pri metóde konečných prvkov treba teleso rozložiť na základné stavebnicové prvky ako sú kocky alebo štvorsteny, prípadne hranicu rozložiť na trojuholníky alebo štvoruholníky. Musíme vedieť pre objem ohraničený hranicou zistiť, či hranica je korektne definovaná. Často sa stretávame s aproximáciou telesa mnohostenom, i keď z matematického hľadiska sa dá hovoriť o interpolácií jeho časti (napr. pri rotačných a translačných telesách). V aplikačných softveroch sa vypracovali rôzne techniky reprezentácie a modelovania telies. V tejto časti priblížime základné techniky modelovania objektov.

10.3.1 Hraničná reprezentácia

Pri hraničnej reprezentácii sa 3D objekt reprezentuje plochou, ktorá tvorí hranicu objektu. Môžeme ju vytvotiť viacerými spôsobmi: rotačne, translačne, analyticky, splajnovo, objemovo, CSG stromom prípadne inými. Pri definovaní niektorých objektov v priestore používame aproximáciu mnohostenom pre ich vykresľovanie. Podrobnejšie o hraničnej reprezentácii mnohostenom, čo podrobnejšie rozoberieme v časti 10.4.

Rotačné teleso

Rotačná plocha vznikne rotáciou krivky okolo priamky, ktorú nazývame *osou rotačnej plochy*. Pre zjednodušenie zápisu zvolíme sústavu súradníc tak, aby sme mali zadanú os rotácie totožnú s osou z a krivka bola zadaná v rovine xz, inak uskutočníme transformáciu v priestore. Nech parametrické vyjadrenie krivky je:

$x = f(t), \quad z = g(t)$.

Každý bod na rotačnej ploche je určený voľbou t a parametrom otočenia u. Parametrické rovnice rotačnej plochy môžeme ľahko odvodiť:

$$x = f(t).\cos(u),$$

$$y = f(t).\sin(u),$$

$$z = g(t),$$

kde parameter t je z intervalu zo zadania krivky a parameter u je z intervalu $\langle 0, 2\pi \rangle$. Ak plocha vytvára v priestore teleso, potom ho nazývame **rotačným telesom**. Napríklad keď je definujúca krivka uzavretá (napr. torus) alebo krivka sa začína a končí na osi z (napr. guľa). Pretože guľa môže byť tiež vytvorená rotačnou plochou, prevzali sa pojmy ako meridián a rovnobežkové kružnice pre všeobecnú rotačnú plochu. Guľová plocha je zadaná nasledujúcimi rovnicami :

$$x = r.\cos(t).\cos(u),$$

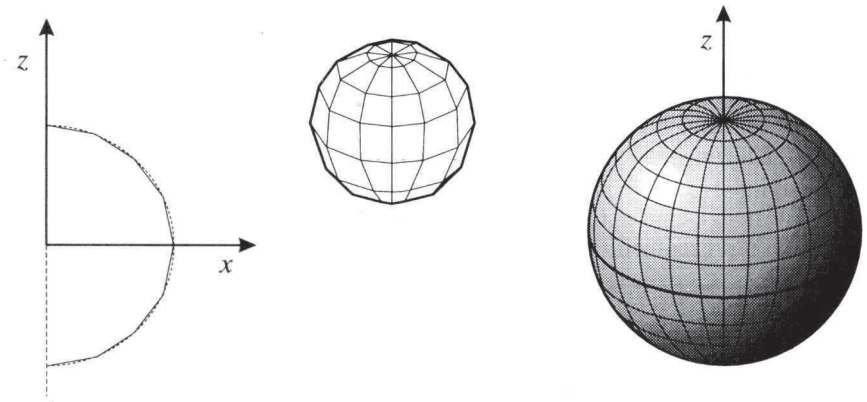
$$y = r.\cos(t).\sin(u),$$

$$z = r.\sin(t),$$

kde $t \in \langle -\pi/2, \pi/2 \rangle$ a $u \in \langle 0, 2\pi \rangle$.

Lineárne interpolujeme krivku v rovine xz výberom n bodov (napríklad s prírastkom dt pre parameter t). Zvolíme si prírastok du pre rotáciu tak, aby sme aproximovali rotáciu m otočeniami od 0 po $2.\pi$. Týmto vytvoríme na rotačnej ploche sieť u- a t-kriviek, ktoré lineárne interpolujú meridiány a rovnobežkové kružnice (pozri obr.10.8).

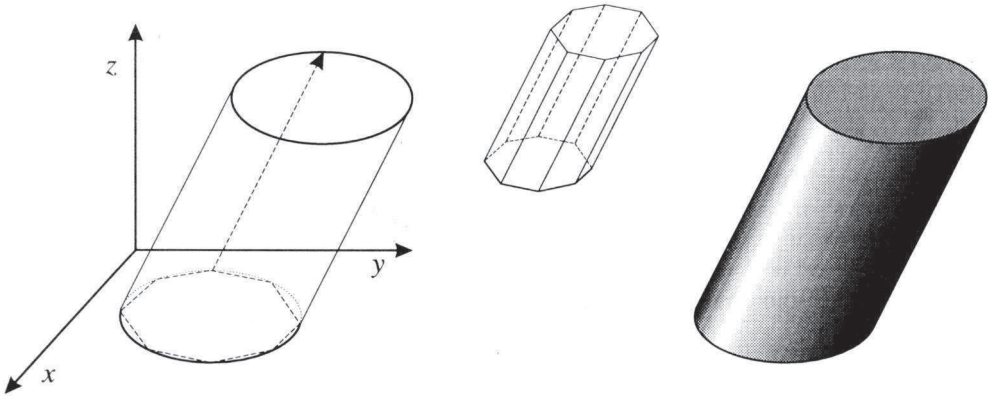
Ak krivka prechádza cez os z, potom je výhodné vybrať tento bod krivky. Po rotácii interpolujúce steny s vrcholom na osi z vytvárajú trojuholníky a ostatné steny zo siete vytvárajú štvoruholníky.



Obr. 10.8 Zobrazenie hranice gule pomocou rotačnej plochy

Translačné teleso

Translačné teleso vznikne posunutím niektorej uzavretej krivky v rovine, pričom vektor posúvania nesmie v tejto rovine ležať. Takýmto spôsobom vytvárame zovšeobecnené valcové plochy. Pre reprezentáciu v počítači využívame aproximáciu valcovej plochy mnohostenom. Zadanú krivku rozdelíme na n bodov. Takto dostaneme pre podstavy (dolnú a hornú) dva n-uholníky a bočné steny vytvoria štvoruholníky (pozri obr. 10.9).



Obr. 10.9 Zobrazenie hranice valca pomocou translačnej plochy

Ak uzavretú krivku posúvame pozdĺž všeobecnej krivky, hovoríme o **posuvnom šablónovaní**. Teleso je určené šablónou a krivkou, po ktorej sa šablóna posúva. Sú možné dva spôsoby pohybu: pri prvom posune sa rovina, v ktorej leží krivka iba rovnobežne posúva a pri druhom posune sa táto rovina otáča tak, aby bola v každom bode krivky kolmá na jej dotykový vektor.

Translačné teleso je príkladom šablónovania prvým spôsobom posunu a rotačné teleso je vlastne posuvné šablónovanie krivky pozdĺž kružnice druhým spôsobom posunu. Obidve uvedené metódy popisu telies sa často označujú ako 2,5 D modelovanie objektov.

Analytické a splajnové plochy

V niektorých prípadoch zadáme plochu a vhodným modelovaním vytvoríme teleso (napr. pridáme steny alebo vytvoríme konvexný obal a pod.). Najprv uvedieme v niektorých aplikáciach využívané zobrazovanie funcie dvoch premenných.

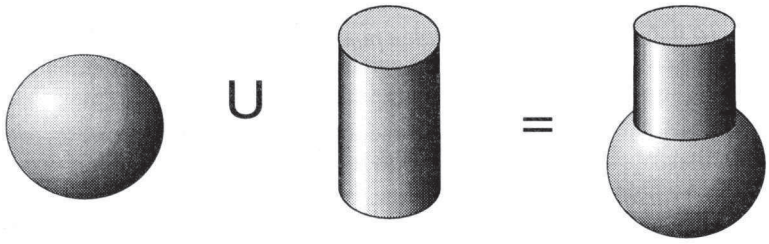
Predpokladajme, že máme zadanú funkciu dvoch premených $z = f(x, y)$, ktorú máme definovanú na obdĺžniku $\langle x_1, x_2 \rangle \times \langle y_1, y_2 \rangle$. Chceme funkciu interpolovať niektorým mnohostenom. Vo vyšetrovanej časti definičného oboru funkcie vytvoríme pravouhlú sieť. V priesečníkoch siete vypočítame príslušné funkčné hodnoty $f(x, y)$. Takto môžeme podobne ako pre rotačné teleso vytvoriť sieť bodov na ploche, x-krivky a y-krivky, t.j. krivky odpovedajúce jednotlivým rezom rovinami x = const a y = const. Tieto sú obvykle interpolované lomenými čiarami. Plocha je potom vytvorená štvoruholníkmi.

Vrcholy však nemusia ležať v jednej rovine, a preto namiesto štvoruholníkov používame dva trojuholníky.

Okrem uvedených metód existuje celá škála ďalších spôsobov generovania alebo aproximovania plôch splinom. V predchádzajúcej kapitole 4 sme uviedli Coonsov, Fergusonov a B-spline. Na tieto metódy popisu plôch obvykle naväzujú aplikačné výpočty ako napríklad pevnosť a tuhosť objektu. Ide o pomerne zložitú problematiku, ktorú tu nebudeme podrobnejšie popisovať.

10.3.2 Konštrukčno geometrické modelovanie (CSG strom)

CSG (Constructive Solid Geometry) reprezentuje metódu, ktorá umožňuje modelovanie geometrických telies pomocou jednoduchých **základných telies** (guľa, kváder, valec) a **logických operácií** ako sú prienik, zjednotenie a rozdiel množín. Pre každé základné teleso môžeme voliť určité parametre nastavenia (napr. pre guľu polomer, pre kváder jeho rozmery, pre valec polomer podstavy, vektor posuvu a výšku hornej podstavy) a definovať jeho umiestnenie v priestore. Tento postup pre základné teleso môžeme nahradiť prípustnou transformáciou pre jeho základný tvar.

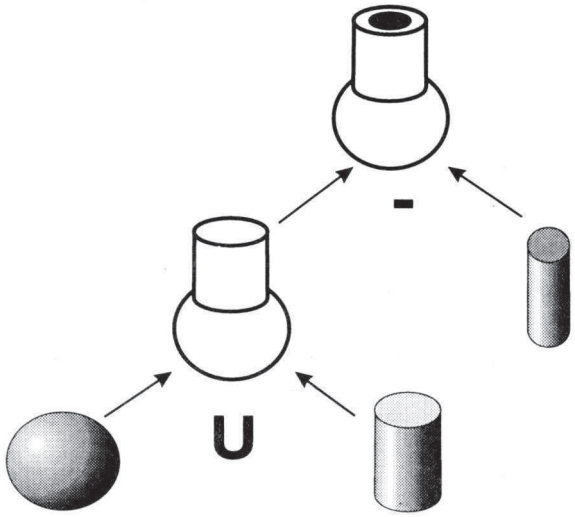


Obr. 10.10 Zjednotenie dvoch základných telies gule a valca

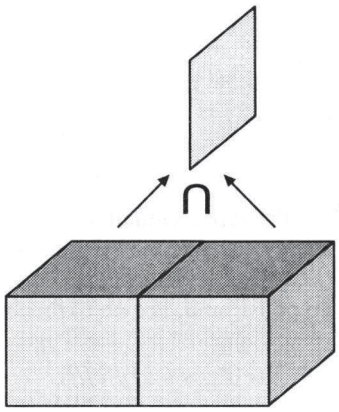
Obrázok 10.10 znázorňuje zjednotenie gule a valca, ktoré boli umiestnené do spoločného bodu. Geometrické modelovanie je vytvorené pomocou stromu, kde koreňom stromu je výsledné teleso a listom základné telesá. Jednotlivým nelistovým uzlom odpovedajú logické operácie na jeho podstromoch. Za logické operácie najčastejšie vyberáme prienik, zjednotenie a rozdiel. Hoci operáciu rozdielu môžeme nahradiť pomocou doplnku množiny a prieniku, ponechávame ju pre jej praktické využitie pre návrty alebo výrezy. Jednoduchý strom na obr. 10.11 symbolicky demonštruje použitie zjednotenia arozdielu na dvoch úrovniach.

Aby sme sa vyhli singulárnym prípadom, aký máme napríklad znázornený na obr. 10.12 pre dve kocky, zavedieme regularizované množinové operácie nasledujúcim spôsobom:

$A \ op^* \ B$ = closure(interior($A \ op \ B$)), kde closure znamená uzáver a interior vnútro množiny. Potom výsledkom prieniku z obr. 10.12 bude prázdna množina.



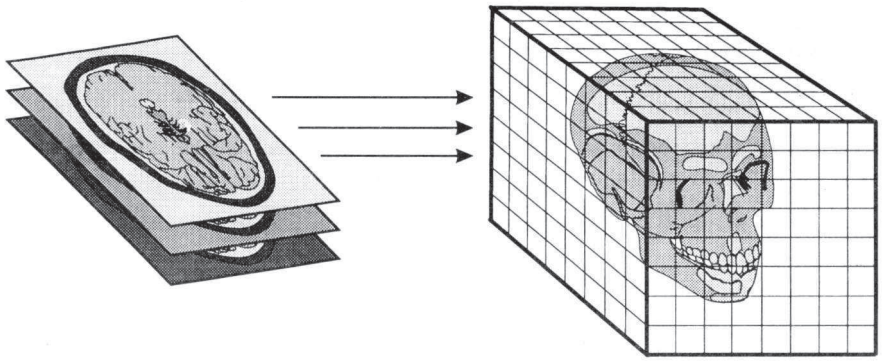
Obr. 10.11 Znázornenie stromu pre CSG teleso



Obr. 10.12 Singulárny prípad pre CSG teleso

10.3.3 Objemové modelovanie

Ďalším spôsobom modelovania je dekompozícia priestoru na bunky. Buď urobíme rozklad objektu na jednoduchšie telesá, ktoré nemusia byť nutne rovnaké, alebo urobíme rozklad priestoru na elementárne prvky a objekt transformujeme do týchto objemových prvkov (voxlov). Podobne sme transformovali v prípade roviny objekt do rastra (pixlov). Objemové modelovanie je rozšírené pri aplikáciach v medicíne, kde objekt skladáme z 2D rezov do 3D kocky (pozri obr. 10.13).



Obr. 10.13 Skladanie CT rezov do 3D kocky vytvorenej voxlami

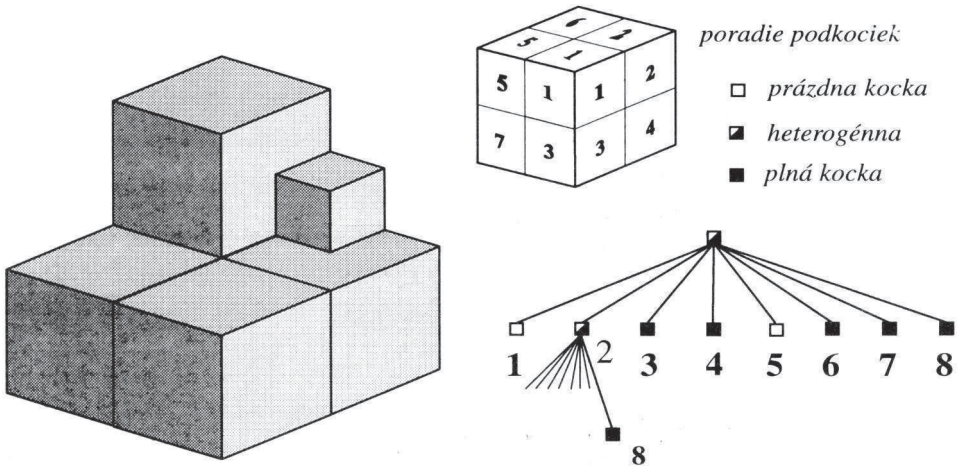
Model objektu záujmu sa vytvára segmentovaním obrazu. Rôzne zobrazovacie techniky sa snažia zviditeľniť čo najprirodzenejšie vytvorený objekt. Podobne ako pre CSG strom aj pre objemové modelovanie je vhodné použiť na zobrazenie upravený algoritmus ray-tracing.

Niekedy je výhodnejšie priestorovú kocku reprezentovať oktálnym stromom. Na obrázku 10.14 máme jednoduchý objekt tvorený z kociek a jeho oktálny strom. Priestor s telesom je uzavretý do kocky, ktorú postupne delíme na osem rovnakých podkociek, pokiaľ nie sú homogénne (buď v telese alebo mimo). Teleso modelujeme zoznamami kociek, ktoré teleso obsahuje.

Pre reprezentáciu oktálneho stromu v počítači sa výhodnejšie využívajú lineárne zoznamy. Označíme si prázdnu kocku ako *e* (empty), plnú *f* (full) a heterogénnu (heterogen). Heterogénny uzol stromu zapíšeme v zozname pomocou uzátvorkovania jeho podstromu. Pre objekt z obr. 10.14 zápis vyzerá nasledovne:

(*e*(*eeeeeeef*)*ffefff*).

V nasledujúcej časti ukážeme ako využívame informácie o oktálnom strome pre vykreslenie na základe transformovania do tetrálneho stromu. Kombinácia popísaných metód modelovania telies niekedy prináša zrýchlenie zobrazovania telies.



Obr. 10.14 Oktálny strom pre objemové modelovanie

10.4 Kódovanie mnohostenov

V trojrozmernej grafike sa najčastejšie využíva geometrické modelovanie objektov pomocou hrán a stien. Informácie o objekte sa dajú rozdeliť do troch typov :

1. geometrické kódovanie vrcholov (súradnice bodov),

2. kódovanie spájania vrcholov a štruktúry stien (polygónov),

3. pomocné informácie o špecifických údajoch objektu napr. materiál, farba a pod.

10.4.1 Jednoduché kódovanie

Jedným z najjednoduchších spôsobov kódovania je zadanie každej hrany:

$h = ((x1, y1, z1), (x2, y2, z2))$

a každej steny (polygónu) pomocou súradníc vrcholov napríklad takto:

$P = ((x1, y1, z1), (x2, y2, z2),...,(xn, yn, zn))$.

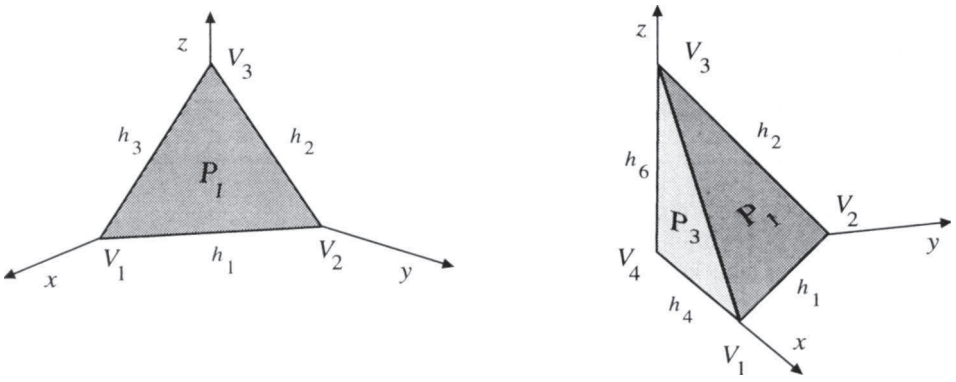
Jednoduchou úpravou môžeme prepísať informácie do nasledujúceho tvaru. Osobitne kódujeme súradnice vrcholov mnohostena v zozname vrcholov :

$V = ((x1, y1, z1), (x2, y2, z2),...,(xm, ym, zm))$

a hrany a steny zadávame indexami (smerníkmi) do zoznamu vrcholov:

$h = (1, 2)$

$P = (1, 2, ...,n)$.



Obr. 10.15 Príklad kódovania štvorstena

Na nasledujúcom príklade (obr. 10.15) ukážeme spôsob kódovania štvorstena, ktorý má 4 vrcholy, 6 hrán a 4 steny. Najprv máme uložené súradnice vrcholov v zozname:

$V_1 = (1, 0, 0)$, $V_2 = (0, 1, 0)$, $V_3 = (0, 0, 1)$, $V_4 = (0, 0, 0)$.

Hrany štvorstena sú zadané indexami zo zoznamu vrcholov:

$h_1 = (1, 2)$, $h_2 = (2, 3)$, $h_3 = (3, 1)$, $h_4 = (4, 1)$, $h_5 = (4, 2)$, $h_6 = (4, 3)$

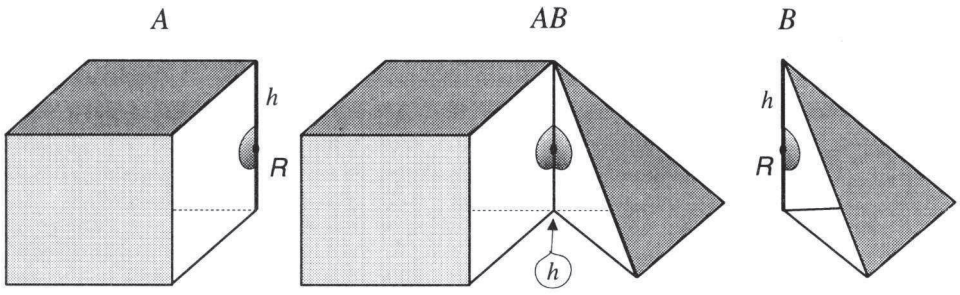
Podobne steny (trojuholníky) definujeme indexami vrcholov:

$P1 = (1, 2, 3)$, $P2 = (4, 2, 1)$, $P3 = (4, 1, 3)$, $P4 = (4, 3, 2)$.

Vrcholy sa zadávajú v takom poradí, aby vytvárali kladne orientovanú stenu daného mnohostena. To znamená, že normála steny smeruje "von" z objektu.

10.4.2 Eulerova formula

Predchádzajúce zoznamy nedávali možnosť priamo zistiť, či je objekt korektne definovaný. V diferenciálnej geometrii a topológii existujú objekty, ktoré sa lokálne správajú ako euklidovský priestor. Tieto telesá nazývame **variety** *(manifolds)*. V priestore na obr. 10.16 máme príklad telesa *AB*, ktoré nie je varietou. Variety sa dajú charakterizovať pomocou okolia. Pre každý bod variety existuje jeho okolie, ktoré je topogicky ekvivalentné s kruhom. Na obrázku máme pre bod *R* na hrane *h* kocky *A* znázornené požadované okolie, podobne aj pre štvorsten *B*. Pre zlepené teleso *AB* pozdĺž hrany *h* v bode *R* neexistuje okolie topologicky ekvivalentné s kruhom. Nutná podmienka pre variety je, aby pre každú hranu existovali len dve steny, ktoré ju obsahujú.



Obr. 10.16 Nevarietový (non-manifold) mnohosten AB

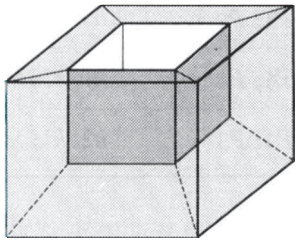
V nasledujúcej časti predpokladajme varietový mnohosten. Označíme počet vrcholov *V (vertices)*, počet hrán *E (edges)* a počet stien *F (faces)*. Pre kocku aj štvorsten máme v nasledujúcej tabuľke vypočítané hodnoty, ktoré predstavujú ujú Eulerovu charakteristiku telesa:

teleso	*V*	*E*	*F*	*V-E+F*
štvorsten	4	6	4	2
kocka	8	12	6	2
torus	16	32	16	0

Platí všeobecne, že všetky telesá, ktoré sú topologicky ekvivalentné kocke majú Eulerovu charakteristiku rovnú 2. Pre teleso zobrazené na obr. 10.17 ekvivalentné s torusom máme Eulerovú charakteristiku rovnú 0. Všeobecne pre variety môžeme vyjadriť

globálnu Eulerovu charakteristiku ako dvojnásobok rozdielu počtu komponentov súvislosti C a rodov G (genus, počet dier):

$$V - E + F = 2.\ (C - G)\ .$$



$V = 16$

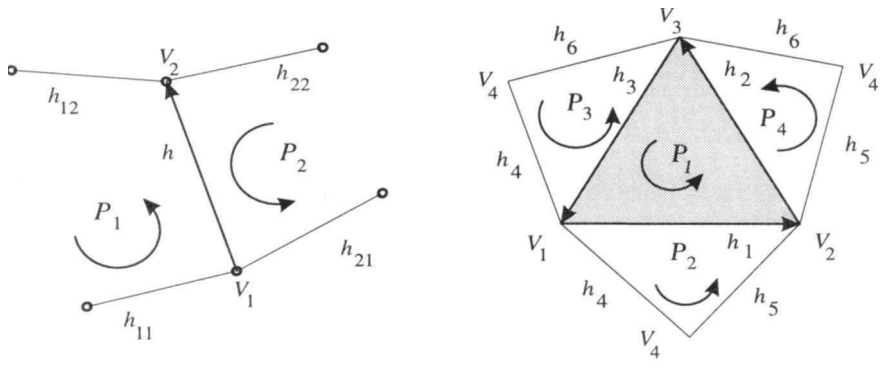
$E = 32$

$F = 16$

Obr. 10.17 Eulerova charakteristika pre torus

10.4.3 Štrukturálne kódovanie hrán

Pri predchádzajúcom kódovaní sme mali o štruktúre mnohostena málo informácií. Nevieme priamo určiť koreknosť zadania varietového mnohostena t.j., ktoré dve steny majú spoločnú hranu. Z tohto dôvodu pre mnohosteny rozširujeme uvedenú dátovú štruktúra na popis pomocou okrídlených hrán *(winged edge)*. Záznam hrany obsahuje odkazy na všetky geometrické prvky: vrcholy, hrany a steny, s ktorými má spoločné body. To znamená, že máme smerníky na vrcholy okrídlenej hrany V_1 a V_2, na susedné hrany h_{11}, h_{12}, h_{21}, h_{22} z dvoch incidentných stien a tieto steny P_1, P_2 (pozri obr. 10.18 a).



a) b)

Obr. 10.18 Dátová štruktúra okrídlenej hrany h pre incidentné steny

Na obr. 10.18 b) je znázornená hrana štvorstena z predchádzajúceho príkladu, kde pre hranu h_3 sú práve dve incidentné steny $P1$ a $P3$. Prvé tri hrany štvorstena z obrázku 10.15 sú zapísané ako okrídlené hrany v nasledujúcej tabuľke:

hrana	vrcholy	incidentné steny	susedné hrany
*h*1	1, 2	*P*1; *P*2	*h*3, *h*2; *h*4, *h*5
*h*2	2, 3	*P*1; *P*4	*h*1, *h*3; *h*5, *h*6
*h*3	3, 1	*P*1; *P*3	*h*2, *h*1; *h*6, *h*4

10.4.4 Zadanie roviny

Z dôvodu interpolácie ďalšej informácie o stene objektu (napr. neskôr pri Phongovom modeli pre určenie farby,...) často potrebujeme vyjadriť jej analytické vyjadrenie v tvare:

$$A.x + B.y + C.z + D = 0.$$

Pre zadané tri body $(x1, y1, z1)$, $(x2, y2, z2)$, $(x3, y3, z3)$ vyjadríme koeficienty predchádzajúcej rovnice takto:

$$A = y1.(z2-z3) + y2.(z3-z1) + y3.(z1-z2),$$

$$B = x1.(z3-z2) + x2.(z1-z3) + x3.(z2-z1),$$

$$C = x1.(y2-y3) + x2.(y3-y1) + x3.(y1-y2),$$

$$D = x1.(y3.z2-y2.z3) + y1.(x2.z3-z2.x3) + z1(x3.y2-.x2.y3).$$

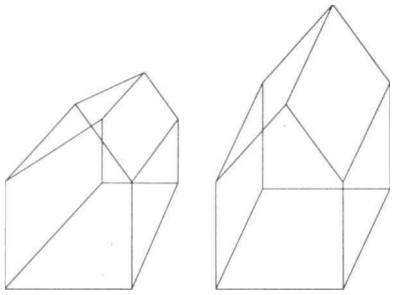
Koeficient D môžeme získať aj tak, že dosadíme súradnice bodu (napríklad prvého) do rovnice a vyjadríme z nej neznámu D:

$$D = - A.x1 - B.y1 - C.z1.$$

10.5 Metódy zobrazovania

V predchádzajúcej časti sme sa zaoberali projekciami. V tejto časti rozoberieme metódy na znázornenie trojrozmernej informácie. Porovnáme rovnobežné a stredové premietanie na zobrazovanie 3D objektov v priestore. Obr. 10.19 nás presvedčí o tom, že tie isté objekty pri stredovom premietaní dávajú jasnejšiu predstavu o priestore ako pri rovnobežnom premietaní. Zdá sa nám, že pri stredovom zobrazení sa nestráca toľko informácií o hĺbke ako pri rovnobežnom premietaní. Je to spôsobené subjektívnym zrakovým vnímaním, ktoré je založené na stredovom premietaní.

Na druhej strane pri konštrukciách a technických výkresoch sa často stretávame so zobrazením predmetov v troch projekciách, ktoré sú na seba kolmé. Majú tú výhodu, že nedochádza ku skresľovaniu vzdialeností a môžeme z výkresu merať jednotlivé dĺžky.

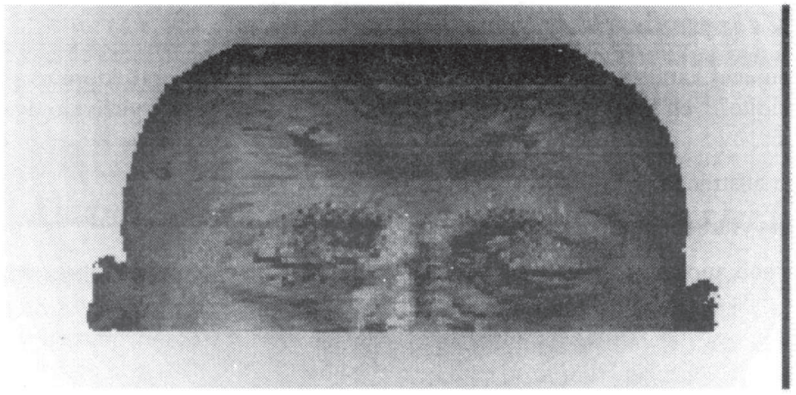


Obr. 10.19 Zobrazenie objektov pri stredovom a rovnobežnom premietaní

Existuje viacero spôsobov, ktoré simulujú priestorové vnímania človeka. I keď sa robia stále nové psychofyzikálne výskumy zrakového vnímania, nie sú všetky psychologické faktory vnímania priestoru ešte dopodrobna vysvetlené. Uvedieme doposiaľ využívané metódy zobrazovania, ktoré dávajú najviac informácií o priestore.

10.5.1 Prenos informácií o hĺbke pomocou jasu

Pre zvýraznenie súradníc hĺbky predmetov, môžeme časti predmetov umiestených bližšie k pozorovateľovi vykresľovať jasnejšie a časti predmetov, ktoré sú ďalej od pozorovateľa slabšie. Tieto postupy sa najčastejšie využívajú v medicíne, napr. pri tomografických aplikáciach.



Obr. 10.20 Zobrazenie CT rezov v priestore pomocou jasu

10.5.2 Prenos informácií pomocou dynamickej projekcie

Ak máme možnosť pozorovať jeden a ten istý predmet z rôznych smerov pohľadu, prípadne ho otáčať, získame lepšiu priestorovú predstavu o telese. Preto časový rozmer animácie nám môže nahradiť stratenú informáciu z projekcie.

Jednou z možností takého zobrazovania je aj dynamické orezávanie v hĺbke. Pri tejto metóde sa zobrazujú len tie časti objektu, ktoré sú pred určitou rovinou. Túto rovinu postupne posúvame ďalej od pozorovateľa, a tak vytvárame dojem trojrozmernosti objektu.

10.5.3 Stereoskopia

Vo všetkých predchádzajúcich metódach sa predpokladala jedna projekcia. Avšak existujú metódy, ktoré imitujú situáciu zrakového vnímania v tom, že každým okom vnímame iný aj keď podobný obraz. Preto vytvoríme dva zodpovedajúce obrazy objektu pre ľavé a pravé oko, tak ako ich vidíme v skutočnosti pri bezprostrednom pozeraní. Existujú v podstate tri postupy pre vytvorenie priestorového (plastického) videnia obidvoma očami, pri ktorých možno rozoznať, ktoré predmety sú bližšie a ktoré vzdialenejšie.

V prvom prípade sa využívajú **dve rôzne obrazovky** a pomocou špeciálneho optického systému sa prenášajú obrazy do oka pozorovateľa. V druhom prípade sa využíva jedna obrazovka a **synchronizáciou pohľadu** ľavého a pravého oka vidíme správne obrazy. V treťom prípade sa na jednej obrazovke vytvára *obraz pre ľavé a súčasne aj pre pravé oko*. Pomocou špeciálnych stereoskopických okuliarov, ktoré využívajú farebné alebo polarizované filtre, vnímame očami zodpovedajúce obrazy.

Existujú aj iné spôsoby stereoskopie, ktoré využívajú nové fyzikálne objavy ako napríklad hologram, ktoré sa doposiaľ v praxi natoľko nerozšírili.

10.5.4 Zobrazenie viditeľných hrán alebo stien

V nasledujúcej kapitole sa budeme podrobne zaoberať algoritmami, ktoré realizujú zobrazenie viditeľných hrán alebo stien. Pre tieto algoritmy sa často využívajú dva nasledujúce spôsoby :

1. spôsob odstránenia neviditeľných hrán,

2. spôsob vyfarbenia viditeľných plôch.

Prvý spôsob môžeme realizovať na rastrových aj na vektorových obrazovkách, avšak druhý spôsob môžeme realizovať len na rastrových zariadeniach.

11

Určenie viditeľného povrchu

11.1 Úvod

Našou úlohou bude určiť viditeľné hrany resp. steny pre objekty v priestore a zobraziť ich v priemete. Predpokladajme, že máme zadané objekty a projekciu. Ak sa pozeráme zo stredu projekcie (pre stredové premietanie), potom zobrazujeme na obrazovku len viditeľné časti. Úloha sa formuluje pomerne jednoducho, avšak skutočná realizácia na počítači si vyžaduje presne sformulované algoritmy s vyriešením mnohých geometrických problémov (singularít).

Algoritmy závisia od analýzy dvoch základných prístupov k riešenému problému. V prvom prístupe sa viditeľnosť objektov vyšetruje v priemetni rastrovo. Zisťujeme, ktoré časti stien sú viditeľné v obrazových bodoch záberu. Potrebujeme preveriť polohu všetkých n stien ku každému obrazovému bodu, aby sa vyjasnilo, ktorá z nich je najbližšie k pozorovateľovi. Pre b bodov je počet výpočtov priamo úmerný hodnote $n.b$, kde b je približne od 250 000 do 1 000 000 v závislosti od rastra obrazovky (512 - 1024).

V druhom prístupe sa každá stena objektu porovnáva s ostatnými stenami. Počet porovnaní je v tomto prípade úmerný hodnote n^2 stien. Zdalo by sa, že tento prístup bude lepší pri počte do 200 000 stien. Pri porovnávaní dvoch stien však potrebujeme podstatne viac výpočtov, a preto je často rýchlejší prvý prístup.

Sutherland, Sproull a Schumacker nazvali algoritmy viditeľnosti podľa uvedeného typu prístupu ako algoritmy pracujúce buď **v obrazovom priestore** alebo **v objektovom priestore**. V tejto časti uvedieme niektoré z popísaných prístupov. Okrem toho metódy určovania viditeľnosti delíme na čiarové (*hidden line elimination*) a plošné (*hidden surface elimination*).

11.2 Zjednodušenie stredového premietania a obálky

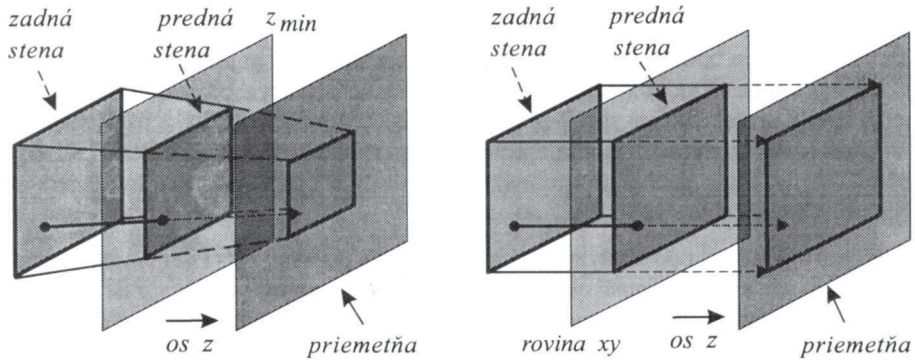
Odstránenie neviditeľných častí stien sa musí uskutočniť v trojrozmernom priestore ešte pred projekciou do roviny, pretože sa pri nej stratí informácia o tretej súradnici potrebnej pre porovnávanie hĺbky. Jeden zo základných problémov je určenie **viditeľného bodu** z dvoch bodov P1 a P2, ktoré ležia na jednej projekčnej priamke (t.j. projektore).

Porovnávanie hĺbky sa obvykle aplikuje až po uskutočnení normalizujúcej transformácie, ktorá nám zabezpečí buď kolmé rovnobežné premietanie do roviny xy alebo stredové premietanie so stredom v začiatku súradnicovej sústavy. Čiže, pre rovnobežné

premietanie ležia dva body na jednom projektore, ak platí pre súradnice bodov $x1 = x2$ a $y1 = y2$. Pri stredovom premietaní musí zas platiť $x1/z1 = x2/z2$ a súčasne $y1/z1 = y2/z2$.

Môžeme uskutočniť takú transformáciu objektov, pri ktorej sa stred premietania premiestni do nekonečna. Predpokladajme, že stred projekcie sa premiestni pozdĺž osi z. Čiže stredové premietanie prejde do rovnobežného premietania a projektívna priamka prejde do priamky kolmej na rovinu xy. Preto pre každý bod vypočítame nové súradnice predpisom:

$$x' = x/z, \qquad y' = y/z.$$



Obr. 11.1 Rovnobežné a stredové premetanie

Ak chceme, aby sa kanonický tvar stredového premietania transformoval do kanonického tvaru rovnobežného premietania (pozri obr. 11.1), potom musíme požadovať, aby rovina rovnobežná s rovinou xy vo vzdialenosti z_{min} prešla do roviny xy. Pričom stred premietania necháme konvergovať na osi z do nekonečna. To bude splnené, ak bude platiť pre súradnicu z:

$$z' = \frac{z}{1-z_{min}} - \frac{z_{min}}{1-z_{min}}$$

Táto transformačná matica sa zapíše nasledovne:

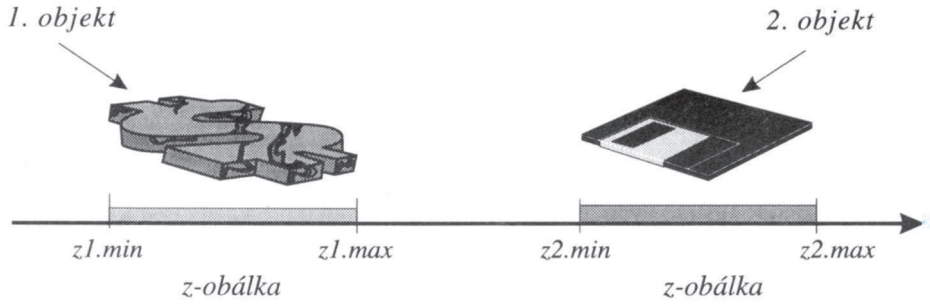
$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1-z_{min}} & 1 \\ 0 & 0 & \frac{-z_{min}}{1-z_{min}} & 0 \end{pmatrix}$$

Orezávanie objektov je treba uskutočniť pred touto transformáciou, pretože pri nej sa nezachováva orezávanie objektov. V nasledujúcich algoritmoch budeme často hovoriť o stenách v priestore a o ich priemete, ktoré budeme nazývať mnohouholníkmi v priemetni.

Aby sme urýchlili algoritmus a vylúčili zbytočné porovnávanie dvoch stien, často sa využíva pojem obálky. Obálkou objektu rozumieme taký minimálny obdĺžnik (interval),

ktorý obsahuje daný priemet objektu. Pri kolmom premietaní do roviny *xy* obálku steny určíme jednoducho prehľadaním priemetov jej vrcholov a nájdením minima a maxima. Ak sa obálky priemetu dvoch stien neprekrývajú, potom môžeme steny v priemete vykresľovať nezávisle. Ak sa obálky priemetov dvoch stien pretínajú, potom sa môžu, ale nemusia príslušné mnohouholníky (steny v priemete) pretínať. Preto v tom prípade musíme zistiť, či sa dané mnohouholníky pretínajú. Niekedy využívame obálky stien v rovine *xy*, inokedy zase v projekcii na os *z*. Napríklad dve *z*-obálky sa neprekrývajú pri projekcii na os *z*, ak sú splnené tieto nerovnosti (pozri aj obrázok 11.2) :

$z1.max <= z2.min$ alebo $z2.max <= z1.min$.



Obr. 11.2 Prázdny prienik dvoch z-obálok

11.3 Algoritmus využívajúci z-bufer

Tento algoritmus je jedným z najjednoduchších algoritmov pracujúcich v priestore obrazu. Počas jeho práce sa udržiava najmenšia *z*-súradnica jednotlivých obrazových bodov (*z*-bufer) a tiež bufer ich farieb pre zobrazenie. Na začiatku *z*-bufer naplníme maximálnymi hodnotami a bufer farieb inicializujeme farbou pozadia. Každú stenu transformujeme do rastrovej formy a vypočítanú *z*-súradnicu porovnáme s odpovedajúcim bodom *z*-bufra. Ak je hodnota menšia t.j. bod je bližšie k pozorovateľovi, potom sa bod vykreslí a hĺbková súradnica sa odloží do *z*-bufra.

Algoritmus *z*-bufer sa dá zapísať do týchto krokov:

Z-bufer algoritmus

1. Pre každý obrazový bod (*x, y*) inicializujeme *z*-bufer a bitovú mapu t.j.

$$Z_{buf}(x,y) = z_\max \quad \textbf{and} \quad M_{bit}(x,y) = backgr$$

2. Pre každú stenu P určíme v priemete mnohouholník a urobíme jeho rozklad do rastrovej formy. Vypočítame z súradnicu (t.j.hĺbku) pre každý obrazový bod (*x, y*).

if $z(x, y) < Z_{buf}(x,y)$ **then begin**

$$Z_{buf}(x,y) = z(x,y);$$
$$M_{bit}(x,y) = \text{colour}(P);$$

end

Pre výpočet súradnice hĺbky daného bodu (x, y) steny využívame analytické vyjadrenie roviny určenej stenou:

$$A.x + B.y + C.z + D = 0.$$

Ak táto rovina nie je kolmá na priemetňu xy, potom koeficient C je rôzny od nuly a súradnicu z môžeme vypočítať zo vzťahu

$$z = \frac{-D - A \cdot x - B \cdot y}{C}.$$

Pri rastrovom rozklade mnohouholníka je výhodné postupovať po riadkoch (pozri časť 5.6). Predpokladajme, že spracovávame jeden riadok a poznáme súradnicu $z1$ pre bod ($x1$, $y1$). Ak máme pre nasledujúci bod ($x1+1$, $y1$) vypočítať súradnicu z, potom môžeme využiť predchádzajúcu hodnotu $z1$:

$$z = z1 - A/C .$$

Hodnotu A/C sa vypočíta vopred pre celý mnohouholník, a preto pre jeden bod potrebujeme iba jednu operáciu sčítania.

Týmto zjednodušíme výpočet hĺbky z v jednom riadku pre každý bod až na prvý bod v riadku. Výpočet spracovania scény rastie s počtom stien lineárne a z toho dôvodu sa algoritmus z-bufer často využíva v grafických procesoroch. Algoritmus má však aj jeden vážny nedostatok - potrebuje pre z-bufer príliš veľkú pamäť.

11.4 Riadkovo skanovací algoritmus

Tento algoritmus rieši problém viditeľnosti z hľadiska redukcie potrebnej pamäti tým, že udržuje informáciu o viditeľnosti len pre jeden riadok. Tento prístup zovšeobecňuje rozklad mnohouholníka do rastrovej formy. Rozdiel je v tom, že robíme rozklad viacerých mnohouholníkov naraz.

V prvom kroku si vytvoríme tabuľku hrán (*TH*), ktorá obsahuje pre všetky hrany e premietaných stien nasledujúce informácie:

1. x-súradnica x_e koncového bodu, kde sa dosiahne minimálna y-súradnica hrany e

2. maximálna y-súradnica y_{max} hrany e

3. prírastok $dx = 1/m$ (zo smernice hrany), využívaný pri prechode na nasledujúci riadok

4. smerník na mnohouholník P, ktorý obsahuje danú hranu e.

Pre tieto účely môžeme definovať **array** [1..maxedge] **of record** x, $y$max, dx : **real**;

polygon: **integer**; **end;**

Okrem toho si vytvoríme tabuľku mnohouholníkov (*TM*) s týmito informáciami :

1. koeficienty A, B, C, D analytického vyjadrenia roviny daného mnohouholníka,

2. farba mnohouholníka,

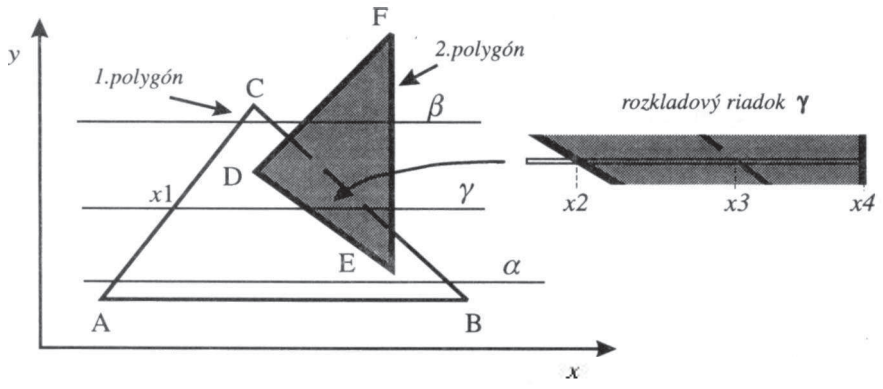
3. boolovská premenná na nastavenie parametra (vnútri, mimo) daného mnohouholníka.

Na obrázku 11.3 sú zobrazené projekcie dvoch trojuholníkov do roviny *xy*. Pri usporiadaní v tabuľke hrán máme tieto hrany: AB, AC, FD, FE, CB a DE. V tabuľke mnohouholníkov máme dva trojuholníky ABC a DEF.

V druhom kroku si pre zvolený skanovací riadok vytvoríme tabuľku aktívnych hrán (*TAH*). Do tabuľky *TAH* pridávame hrany zo zoznamu *TH*, ktoré začnú byť aktívne pre zvolený riadok. Tento zoznam usporiadame vždy podľa súradnice *x*.

V treťom kroku algoritmu budeme vyšetrovať úseky x_{i-1}, x_i z tabuľky aktívnych hrán pre zvolený riadok. V prípade skanovacej priamky (α) z obrázku 11.3 bude tabuľka aktívnych hrán obsahovať len dve hrany AC a BC, ktoré patria len jednému mnohouholníku. V takomto prípade sa urobí vykreslenie daného úseku. Pre skanovací riadok (β) bude tabuľka aktívnych hrán obsahovať 4 hrany AC, BC, ED a EF. V úseku od hrany AC ku hrane BC sa najprv nastaví boolovská premenná pre trojuholník ABC (vnútri) a na konci na hrane BC (mimo). V tomto prípade je nastavený príznak vnútri vždy len pre jeden mnohouholník, a preto zodpovedajúci úsek sa vykreslí farbou daného mnohouholníka.

Situácia bude zložitejšia v prípade skanovacej priamky (γ), pre ktorú budeme mať tiež 4 aktívne hrany ako pre riadok (β). Avšak v tomto prípade bude nastavená boolovská premenná vnútri naraz u dvoch mnohouholníkov a to na úseku od hrany ED k hrane BC. Preto musíme v bode na prieniku priamky (γ) s hranou DE, zistiť *z*-súradnicu pre obidva mnohouholníky. Trojuholník DEF je na tomto úseku v popredí (súradnica *z* je menšia), a preto vykreslíme úsek farbou podľa tohto mnohouholníka.



Obr. 11.3 Rozkladový riadok pre scan line algoritmus

V nasledujúcej časti popíšeme algoritmus Scan-Line a jeho funkcie. Funkcia *merge* pridáva informácie z tabuľky hrán (TH) do tabuľky aktívnych hrán (TAH). Funkcia *number* vráti počet aktívnych polygonov na danom úseku. Funkcia *zmin* nám vráti pre daný úsek mnohouholník z tabuľky aktívnych mnohouholníkov, ktorý je najbližšie k pozorovateľovi. Formálne máme označený cyklus v 3. a 4. kroku podľa počtu hrán v tabuľke aktívnych hrán. V 3. kroku algoritmu vykresľujeme jednotlivé úseky podľa farieb viditeľných mnohouholníkov. V 4. kroku upravujeme tabuľku aktívnych hrán. Funkcia *remove* odstráni zadanú hranu z tabuľky TAH.

Algoritmus Scan-Line

```
Procedure Scan_Line;
begin
1. Inicializácia tabuľky TH (hrán) a TM (mnohouholníkov);
   tabuľka TAH := ∅ a M_bit (x, y) := background;          { pre všetky (x, y) obrázku }
2. for y = ymin to ymax do
      begin
      merge ( TAH, y );                  { pridanie zoznamu z TH pre aktuálne y }
      sort ( TAH );                      { usporiadanie TAH podľa x }
3.    for (x_i, e) = (x_1, e_1) to (x_n-1, e_n-1) do   { pre všetky body podľa hrán TAH}
         begin
         flag(P_e) := invert (P_e);      { invertuj flag vnútri/mimo mnohouholníka P_e }
         nap = number ( TAM );           { počet aktívnych mnohouholníkov}
         if nap = 0 then draw ( x_i, x_i+1, background );
         if nap = 1 then draw ( x_i, x_i+1, colour (P_e) );
         else begin                      { algoritmus nebude fungovať pre pretínajúce
            P := zmin ( x_i, x_i+1, TAM );     sa mnohouholníky }
            draw ( x_i, x_i+1, colour (P) );
            end
         end { od kroku 3 }
4.    for (x_i, e) = (x_1, e_1) to (x_n, e_n) do
         begin
         if y_max (e) = y then remove (e, TAH)          { odstráň hranu z TAH }
         else x := x + dx;              { parameter dx vypočítame zo smernice hrany }
         end
end;
```

Tento typ algoritmu rozpracovali Watkins a Bouknight už v 70. rokoch. Sú známe rôzne modifikácie tohto algoritmu, ktoré sa snažia urýchliť výpočet úseku viditeľnosti s využitím riadkovej súvislosti (koherencie). Napríklad, ak sa nezmení usporiadanie hrán v **TAH** pre dva riadky za sebou, potom môžeme využiť predchádzajúce vykresľovanie bez výpočtu hĺbky. Tento algoritmus je vhodný aj pre objekty, ktoré sú popisované plochou alebo aproximáciou pomocou plátov.

11.5 Algoritmus hĺbkového triedenia

Tento algoritmus navrhli Newell a Sancha. Základná myšlienka algoritmu spočíva v usporiadaní stien podľa ich vzrastajúcej z-súradnice v priestore t.j. v smere pohľadu. Budeme preto hovoriť o z-hĺbke alebo o vzdialenosti od pozorovateľa. Najnižšiu prioritu majú steny od pozorovateľa najvzdialenejšie a najvyššiu budú mať steny najbližšie. Postupne sa budú zobrazovať projekcie stien ako vyplnené mnohouholníky podľa priority. Pretože najbližšia stena má najväčšiu prioritu, vykreslíme jej mnohouholník v priemete ako posledný a prekryje tie mnohouholníky, ktoré sa zobrazili pred ním.

Algoritmus sa dá rozdeliť na tri časti :

1. Usporiadanie stien podľa ich maximálnej z-súradnice v sústave pozorovateľa.

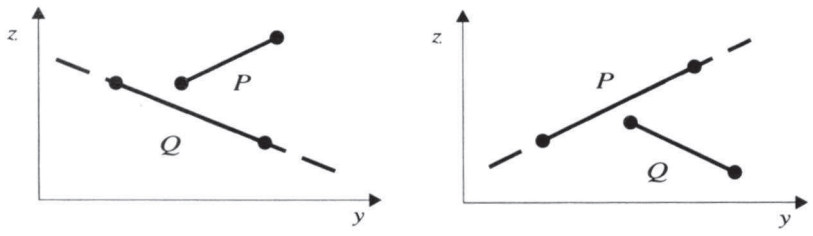
2. Vyriešenie niektorých prípadov stien prekrývajúcich sa v *z*-obálke, pre ktoré nebolo možné uskutočniť jednoznačne usporiadanie.

3. Zobrazenie mnohouholníkov do rastrovej formy podľa vzrastajúcej priority.

Časti 1 a 2 sa navzájom dopĺňajú. Ak sa po usporiadaní posledná stena v zozname neprekrýva v hĺbke s inými, potom sa môže uskutočniť jej rozklad. V prípade, že sa steny prekrývaju v hĺbke, potom musíme zmeniť ich poradie.

Predpokladajme, že stena P je na konci usporiadaného zoznamu. Predtým ako sa zobrazí do rastrovej formy, musí sa porovnať s každou stenou Q, ktorej *z*-obálka sa prekrýva s *z*-obálkou P. Správne poradie sa overí postupne podľa nasledujúcich piatich podmienok (testov). Ak je splnená aspoň jedna podmienka pre každý mnohouholník Q, potom môžeme mnohouholník P zobraziť do rastrovej formy :

1. *x*-obálky mnohouholníkov P a Q sa neprekrývajú (procedúra '*test1*(P, Q)'),
2. *y*-obálky mnohouholníkov P a Q sa neprekrývajú (procedúra '*test2*(P, Q)'),
3. všetky vrcholy steny P ležia vo vzdialenejšom polpriestore od pozorovateľa, ktorý je vytvorený stenou Q (pozri obr. 11.4.a) - *test3*(P, Q),
4. všetky vrcholy steny Q ležia v bližšom polpriestore k pozorovateľovi, ktorý je vytvorený stenou P (pozri obr. 11.4.b) - *test4*(P, Q),
5. projekcie stien t.j. mnohouholníky P a Q v rovine *xy* sa neprekrývajú - *test5*(P, Q).

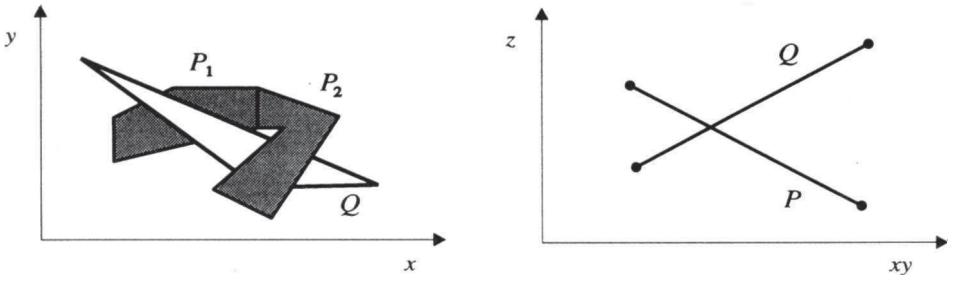


a) b)

Obr. 11.4. Test3 a test4 pre mnohouholníky P a Q

Ak nie je splnená ani jedna podmienka, musíme vymeniť poradie stien. Urobíme opäť kontrolu na splnenie usporiadania a overíme príslušné podmienky. Môže však nastať **cyklická zámena** niekoľkých mnohouholníkov a algoritmus sa zacyklí (pozri obr. 11.5).

Situáciu cyklického prekrývania nerieši ani výmena stien. Pre správne určenie poradia musíme urobiť dodatočnú zmenu stien. Polygón P v priestore rozdelíme na dve časti podľa roviny steny Q, ktorá ho pretína. Namiesto steny P vložíme do zoznamu stien dve nové steny P_1 a P_2. Z obr. 11.5 vidíme, že pre tieto steny sa cyklická zámena stien odstráni, lebo postupne vykreslíme mnohouholník P_1, Q a nakoniec mnohouholník P_2 (splnená podmienka 3 a 4).



Obr. 11.5 Cyklické prekrývanie mnohouholníkov

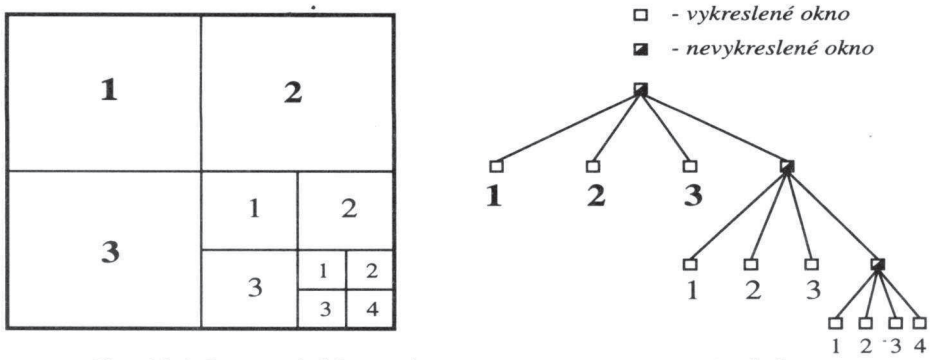
Zapíšeme **algoritmus priority**. Funkcia *zsort* usporiada steny podľa maximálnej *z* súradnice. Funkcia *zmax* vybere z usporiadaného zoznamu stien tú, ktorá je najvzdialenejšia. Funkcia *test0* nám naplní nový zoznam TQ všetkých stien, ktoré majú prienik so stenou P v *z*-obálke. V 4. kroku testujeme tieto steny prípadne mnohouholníky v priemete na podmienky 1-5. Ak pre niektorú stenu Q nie sú splnené podmienky, potom zameníme steny P a Q v usporiadanom zozname. V algoritme nemáme uvedené riešenie cyklickej zámeny stien.

Algoritmus priority

```
Procedure Paint_priority;
begin
1. Inicializuj TM-tabuľku mnohouholníkov;
   zsort (TM);                          { usporiada steny podľa ich maximálnej z súradnice }
repeat
    repeat
2.  P := zmax (TM);                     { vyber posledný mnohouholník z tabuľky }
3.  TQ:= test0 (P, TM);                 { vyhľadaj všetky mnohouholníky Q prekrývajúce sa s
                                          mnohouholníkom P v z-obálke}
4.  convert:= true;                     { nastav príznak rozkladu mnohouholníka P }
    for Q ∈ TQ do
        begin
        if convert then                          { testuj pre P a Q }
        if test1 (P, Q) = false and test2 (P, Q) = false and
        test3 (P, Q) = false and test4 (P, Q) = false and
        test5 (P, Q) = false then begin          { ak nie sú splnené podmienky 1-5 }
                                    swap (P, Q);
                                    convert:= false;
                                  end
        end
    until (convert = true);
5.  draw (P);
    TM := TM - { P };
until ( TM = ∅ );
end.
```

11.6 Warnockov algoritmus delenia okna

Warnock bol jeden z prvých, ktorý rozpracoval **algoritmus delenia okna**. Hlavná myšlienka algoritmu spočíva v tom, že problém viditeľnosti sa snažíme vyriešiť v okne naraz. Ak to nie je možné, potom problém riešime rekurzívne v menšom okne (okno postupne delíme na štyri časti, ako to vidíme na obr. 11.6). Takýto prístup vytvára tetrálny strom s listami tých okien, pre ktoré vieme vykresliť mnohouholníky v správnom poradí. Môže sa stať, že pri rekurzívnom riešení sa postupne dostaneme na najnižšiu úroveň stromu, čo znamená, že okno má najmenší rozmer a to jeden pixel. V tomto prípade zistíme, pre ktorú pokrývajúcu stenu je *z*-hĺbka minimálna a vykreslíme pixel farbou tejto steny.



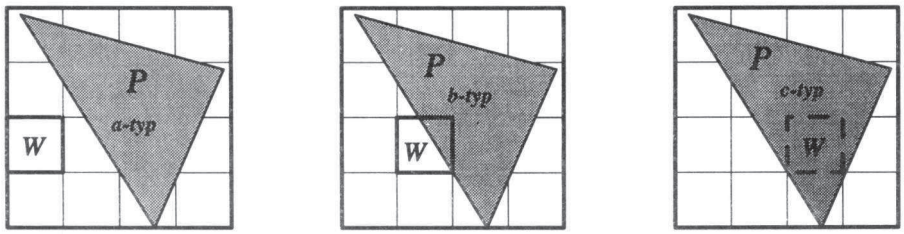
Obr. 11.6 Postupné delenie okna a vytvorenie stromovej štruktúry

Postupne skúmame vzťah priemetu steny k uvažovanému podoknu. Potom podľa prieniku s daným oknom každý mnohouholník zaradíme do nasledujúcich typov (pozri obr. 11.7) :

a) Priemet mnohouholníka je mimo daného podokna.

b) V podokne sa nachádza priemet celého mnohouholníka alebo jeho časti.

c) Celé podokno je zakryté priemetom mnohouholníka.

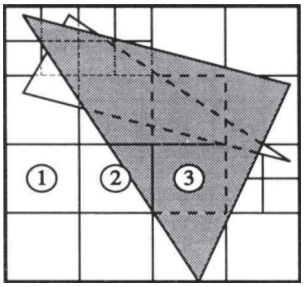


Obr. 11.7 Rozdelenie mnohouholníkov podľa prieniku s oknom

Pre dané okno (podokno) existujú tri jednoduché prípady, kedy môžeme rozhodnúť o jeho zobrazení (pozri obr. 11.8) :

1. V prvom prípade nie je v uvažovanom okne ani jeden mnohouholník, vtedy môžeme vykresliť okno farbou pozadia. To znamená, že všetky priemety stien sú *a-typu* vzhľadom k tomuto oknu.
2. Ak len jeden mnohouholník pretína dané okno, potom ho môžeme vykresliť farbou pozadia a na to vykreslíme orezaný mnohouholník, t.j. len jeden mnohouholník je *b-typu.*
3. V prípade, že existuje len jeden mnohouholník, ktorý obsahuje celé zadané okno (*c-typ*), potom situáciu riešime vykreslením podokna farbou mnohouholníka.

Všetky ostatné situácie považujeme za zložitejšie, a preto vyšetrované okno rozdelíme na 4 menšie časti. Rekurzívne zisťujeme situáciu na nižšej úrovni tetrálneho stromu (*quad tree*), či už situácia nemôže byť vyriešená predchádzajúcimi troma prípadmi. Prípadne rekurziu ukončíme, ak má podokno veľkosť jedného pixlu. V tomto prípade viditeľnosť vyriešime určením minimálnej hĺbky v danom bode pre skúmané mnohouholníky.



Obr. 11.8 Najbližší pokrývajúci trojuholník

Na popis algoritmu budeme používať funkciu *location* (W, P), kde W je zadané podokno a P mnohouholník. Funkcia vráti v prípade a) obrázku 11.7 hodnotu 0, v prípade b) hodnotu 1 a nakoniec pre c) hodnotu 2. Na každej úrovni stromu budeme mať k dispozícii dva zoznamy mnohouholníkov pre vyšetrované podokno. Prvý zoznam L bude obsahovať potenciálne viditeľné steny a zoznam M bude obsahovať steny v priemete pokrývajúce podokno. Predpokladáme, že zadané okno ma rozmer $2^n \times 2^n$ obrazových bodov.

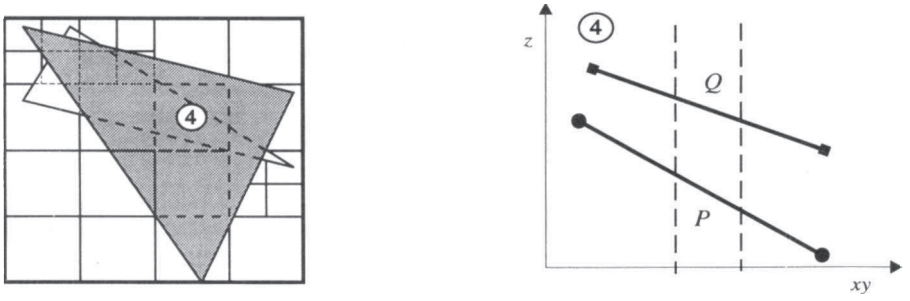
Algoritmus delenia okna

```
begin
0. Inicializujeme W := celé okno;
    zoznam L := { všetky steny scény };
    zoznam M := ∅;
    i := 1;
    Warnock ( W, L, M, i );
end
```

```
Procedure Warnock ( W, L, M, i ) ;
begin
1. if  i = n  then drawmin_z (L, M)
   else
   begin
2. for mnohouholník P ∈ L do
     begin
3.     k:= location (W, P);                    { zistíme polohu mnohouholníka k podoknu }
4.     if k = 0 then L := L - {P};             { mnohouholník P vymažeme zo zoznamu L }
5.     if k = 2 then M:= M + {P}; L:= L - {P}  { mnohouholník P pridáme do zoznamu M }
     end
6. if (number (L) + number (M)) = 0 then draw_box (W, background);     { podmienka 1 }
7. if number (L) = 1 and number (M) = 0 then begin                     { podmienka 2 }
                                              draw_box (W, background);
                                              draw_poly (P, col(P));
                                              end
8. if number (L) = 0 and number (M) = 1 then draw_box (W, col(P));     { podmienka 3}
   else begin
9.       L1:= L; M1:= M;
         Warnock ( W1, L1, M1, i+1);       { rekurzívne zavolanie funkcie pre 4 podokná }
         Warnock ( W2, L1, M1, i+1);
         Warnock ( W3, L1, M1, i+1);
         Warnock ( W4, L1, M1, i+1);
         end
end { návrat z rekurzie }
end.
```

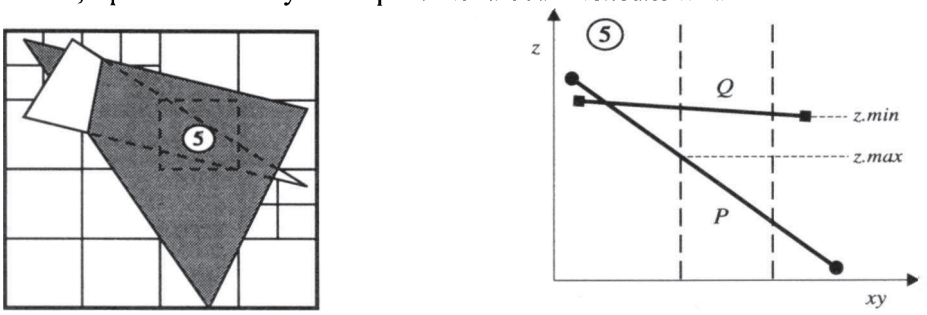
Uvedený algoritmus (v 6., 7. a 8. kroku) ukončí vykreslenie okna. Na zrýchlenie algoritmu k týmto trom podmienkam pridáme ešte nasledujúce dva prípady:

4. Existuje aspoň jeden mnohouholník v zozname M, a pritom môže byť niekoľko potencialne viditeľných v zozname L. Vyhľadáme najbližší pokrývajúci mnohouholník zo zoznamu M a zistíme, či je najbližší aj pre všetky mnohouholníky zo zoznamu L. Overenie najmenšej hĺbky uskutočnujeme pre jednotlivé vrcholy podokna. Ak takýto mnohouholník existuje, potom môžeme vykresliť podokno farbou mnohouholníka P. Napríklad na obrázku 11.9 vidíme takýto mnohouholník P, ktorý má tú vlastnosť, že je najbližší aj pre ostatné mnohouholníky zoznamu L (na obrázku pre Q).



Obr. 11.9 Pokrývajúci mnohouholník s vlastnosťou 4

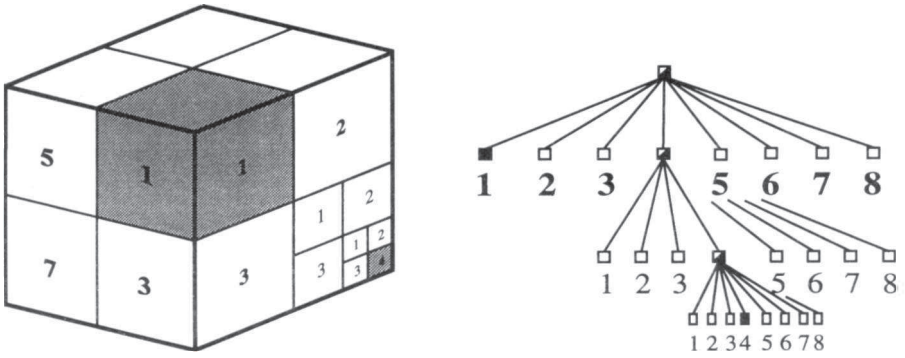
5. V zozname $\boldsymbol{M}$ existuje aspoň jeden mnohouholník, ktorý je najbližšie k pozorovateľovi. Overenie najmenšej hĺbky uskutočnujeme pre vrchol podokna s maximálnou z-súradnicou pokrývajúceho mnohouholníka a minimálnou z-súradnicou mnohouholníka zo zoznamu $\boldsymbol{M}$. Na obrázku 11.10 vidíme, že mnohouholník P má túto vlastnosť, a preto môžeme vykresliť podokno farbou mnohouholníka P.



Obr. 11.10 Pokrývajúci mnohouholník s vlastnosťou 5

11.7 Algoritmus octree

Podstatou tohto algoritmu je priame zobrazovanie objektov zakódovaných pomocou voxlov t.j. základných objemových prvkov. Oktálny strom je strom s jedným koreňom, ktorého každý vrchol je buď listom alebo má osem synov. Koreň oktálneho stromu zodpovedá maximálnej kocke, ktorá obsahuje zadané objekty scény. Túto kocku rozdelíme na 8 rovnakých častí ako je to vidieť na obr. 11.11. Každej časti zodpovedá jeden syn. Pokiaľ je časť homogénna t.j. objekt ju obsahuje, potom tento syn je koncovým vrcholom stromu. Inak sa rozdelí na 8 častí.



Obr. 11.11 Kocka vytvárajúca oktálny strom

Objekt môžeme reprezentovať ako premennú octree typu oct_node nasledujúcim spôsobom :

```
type
    oct_node_ptr = ^oct_node;
    oct_entry = record
                    case homogen : boolean of
                        true : (colour : integer);
                        false : (child : oct_node_ptr); { smerník na nižšiu úroveň }
                    end;
    oct_node = array [0..7] of oct_entry;
```

Tento typ je záznam podľa typu boolovskej premennej *homogen*. Ak je pravda, že je homogénny, potom sa uloží farba homogénnej časti. Ak nie je voxel homogénny, potom je tam smerník na osem synov. Farba niektorej časti bude prirodzené číslo (0, 1,.., max_colour) a pre prázdnu časť nastavíme farbu (-1). Predpokladáme, že voxle sú očíslované tak, ako je to vidieť na obrázku 11.11.

Pri zobrazovaní objektu reprezentovaného oktálnym stromom môžeme postupovať zásadne dvoma spôsobmi. Prvým spôsobom odzadu dopredu ako pri hĺbkovom prioritnom algoritme, kde vieme jednoducho uskutočniť usporiadanie voxlov alebo druhým spôsobom odpredu dozadu. Pri prvej metóde sa voxle prekresľujú a pri druhej sa vykresľuje len nová informácia a z toho dôvodu je táto metóda zobrazovania rýchlejšia. Ukážeme si na jednoduchom príklade, ako vykreslíme oktálny strom pri kolmom premietaní do roviny *xy*. Pri tomto spôsobe sa objekt zakódovaný ako oktálny strom zmení na tetrálny strom. Premenná quadtree pre tetrálny strom je definovaná podobným spôsobom:

```
type
   quad_node_ptr = ^quad_node;
    quad_entry = record
                    case homogen : boolean of
                        true : (colour : integer);
                        false : (child : quad_node_ptr); { smerník na nižšiu úroveň }
                    end;
    quad_node = array [0..3] of quad_entry;
```

Nasledujúca procedúra ukazuje ako sa dá pretransformovať premenná *octree* na *quadtree*, a potom jednoducho vykresliť:

Algoritmus prevodu octree na quadtree

```
procedure convert ( octree : oct_node; var quadtree : quad_node);
begin
  for k:= 0 to 3 do begin
    quadtree[k].homogen := true;
    if octree[k].homogen then
        if (octree[k].colour > -1 ) then                    { predný oktant je plný }
        quadtree[k].colour := octree[k].colour
        else                                                { predný oktant je prázdny }
            if octree[k+4].homogen then                     { predný prázdny, zadný plný }
                if (octree[k+4].colour > -1 ) then
                    quadtree[k].colour := octree[k+4].colour
                else                                        { predný aj zadný prázdny }
                    quadtree[k].colour := backcolor;
```

```
        else begin                                  { predný prázdny, zadný
            quadtree[k].homogen := false;                          heterogénny }
            new ( newquadtree );
            quadtree[k].child := newquadtree;
            convert ( octree[k+4].child^, newquadtree^);
            end
        else begin                                  { predný heterogénny, zadný
            quadtree[k].homogen := false;                          neznámy }
            new ( newquadtree );
            quadtree[k].child := newquadtree;
            convert ( octree[k+4].child^, newquadtree^);
            convert ( octree[k ].child^, newquadtree^);
        end
  end { for }
end.
```

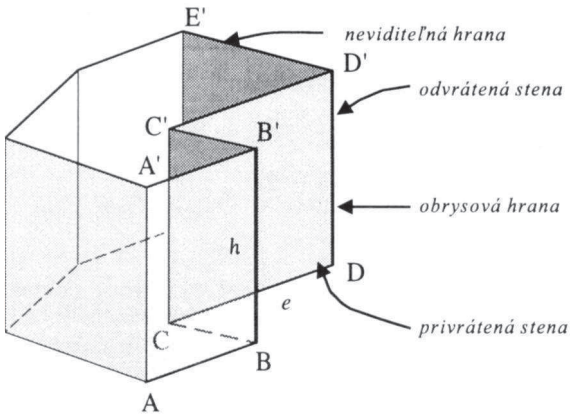
Všimnime si prípad predného oktantu prázdneho a zadného heterogénneho. V tomto prípade musíme pre vytvorenie tetrálneho stromu zaviesť smerníky na nižšiu úroveň pre synov zadnej časti. Ak je predný oktant heterogénny a zadný zatiaľ nešpecifikovaný, potom musíme rekurzívne spracovať najprv zadnú a potom prednú časť.

11.8 Appelov algoritmus

Tento algoritmus pracuje v priestore objektov a vychádza z toho, že každá hrana je prienikom dvoch stien. Prvým krokom je odstránenie neviditeľných stien. Najprv rozdelíme steny na **privrátené** (potenciálne viditeľné) a **odvrátené** (neviditeľné). Privrátená stena je tá, ktorá zviera so smerom premietania tupý uhol, a naopak odvrátená stena zviera ostrý uhol. Na obrázku 11.12 vidíme, že stena BCC'B' je odvrátená od pozorovateľa. Ak sú všetky steny kladne orientované, potom ľahko určíme normálový vektor vychádzajúci z mnohostena pomocou postupnosti troch susedných vrcholov. Rozdelenie na privrátené a odvrátené steny nám ľahko odlíši skalárny súčin normály steny s vektorom premietania (rozumieme tým určený smer vektora od pozorovateľa k priemetni). Pre privrátenú stenu je skalárny súčin záporný a pre odvrátenú stenu je kladný. V prípade, že skalárny súčin je nulový, potom takú stenu budeme považovať za neviditeľnú.

V ďalšom kroku rozdelíme hrany na tri typy podľa toho, s akými stenami sa pretínajú :

1. typ - priesečník dvoch odvrátených stien budeme nazývať neviditeľnou hranou. Na obrázku 11.12 je to napríklad hrana EE'.
2. typ - priesečník privrátenej a odvrátenej steny bude potenciálne viditeľnou obrysovou hranou. Na obrázku 11.12 vidíme, že je to hrana BB'.
3. typ - priesečník dvoch privrátených stien budeme nazývať potenciálne viditeľnou hranou. Na obrázku 11.12 je to hrana AA'.



Obr. 11.12 Zobrazenie mnohostena a jeho viditeľnosť

Čiže potenciálne viditeľné hrany sú 2. a 3. typu. V prípade konvexného mnohostena sú všetky potenciálne viditeľné hrany skutočne viditeľné. U nekonvexných mnohostenov je treba zistiť, či takáto hrana nie je náhodou zakrytá inou stenou. Bod hrany, v ktorom dochádza ku zmene viditeľnosti je daný v priemete priesečníkom s niektorou obrysovou hranou. Na obrázku 11.12 vidíme, že hrana e v priemete je zakrytá stenou ABB'A' a zmena viditeľnosti je na prieniku hrany e a h.

Pre každý bod hrany môžeme určiť počet zakrývajúcich stien (potenciálne viditeľných). Pre potenciálne viditeľné hrany sa počet zakrytí v priemete zmení len na priesečníku s obrysovou hranou. Preto sa každá potencialne viditeľná hrana rozdelí na úseky podľa prienikov s obrysovými hranami. Ak poznáme počet zakrytí v jednom bode (napríklad vo vrchole hrany), potom môžeme postupne prechádzať po úsekoch a na každom prechode zmeníme počet zakrytí o +1 alebo -1.

Napríklad pre hranu e je v bode D počet zakrytí rovný nule a v bode C rovný 1. Obrysová hrana h nám rozdelí hranu e na dva úseky. Špecialne v priesečníku nám znamienko skalárneho súčinu $p.(e \times h)$ rozhodne o zmene znamienka zakrytia, kde p je vektor premietania. Závisí to od orientácie hrany h.

Vo všeobecnosti, ak máme skalárny súčin $p.(e \times h)$ záporný, potom sa počet zakrytí zväčší o 1, kde e je smerový vektor potenciálne viditeľnej hrany a h je smerový vektor obrysovej hrany pre vyšetrovaný bod prieniku. A naopak, ak je súčin $p.(e \times h)$ kladný, potom sa počet zakrytí zmenší o 1. Hrany e a h sú orientované podľa orientácie stien, s ktorými sú incidentné.

Popis algoritmu. Procedúra *intersection_p* zistí, či zadané dve hrany e a h majú spoločný bod v priemete. Ak majú, potom v premennej n vráti hodnotu +1 alebo -1 podľa znamienka skalárneho súčinu $p.(e \times h)$. Procedúra *cover* nám zistí počet zakrytí bodu stenami mnohostena. Procedúra *sort* usporiada úseky zadanej potenciálne viditeľnej hrany. Procedúra *draw* vykreslí zadaný úsek.

Appelov algoritmus

```
procedure Appel;
begin
1. Inicializácia. Určenie normálových vektorov stien;
       Rozdelenie na privrátené a odvrátené steny;
       Vytvorenie množiny poteciálne viditeľných hrán V;
       Vytvorenie množiny obrysových hrán C;
2. for i (e_i ∈ V) do
   begin
3.    for j (c_j ∈ C) do
         intersection_p ( c_j, e_i , n_j );     { určenie prieniku hrany s obrysovými hranami}
      sort (u_j, e_i );                         { usporiadanie úsekov hrany e_i }
      n := cover ( A(u_1, e_i));                { zistenie zakrytia pre začiatočný úsek hrany }
4.    for j (u_j ⊆ e_i) do                      { cyklus po jednotlivých úsekoch hrany }
      begin
         if n = 0 then draw ( u_j );            { vykreslenie nezakrytých úsekov }
         n:= n + n_j ;                          { zmena zakrytia úseku hrany o ± 1}
               { n:= cover ( A(u_j, e_i));      zisti počet zakrytí na danom úseku }
      end
   end
end.
```

11.9 Zobrazenie grafu funkcie dvoch premenných

V niektorých aplikáciách potrebujeme zobraziť graf funkcie dvoch premenných, ktorá je zadaná rovnicou $z = f(x, y)$, definovanej na obdĺžniku $x_1 < x < x_2$, $y_1 < y < y_2$. Pre riešenie môžeme použiť niektorý predchádzajúci algoritmus. Pre tento špeciálny prípad však existuje však aj jednoduchší algoritmus. V tejto časti uvedieme hlavnú myšlienku týchto postupov.

Hľadanú plochu aproximujeme pomocou kriviek. Vyberieme $m \times n$ bodov (x_i, y_j) zadanej plochy, kde indexy $i = 1,..,m$ a $j = 1,..,n$ určujú usporiadané súradnice bodov:

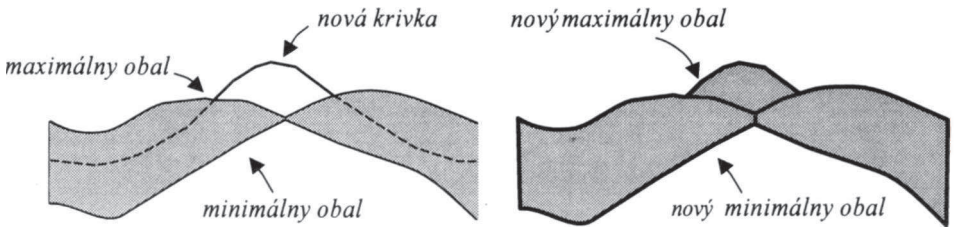
$$x_1 < x_2 < ... < x_m, \quad y_1 < y_2 < ... < y_n.$$

Hodnoty x_i a y_j sa obvykle vyberajú s pravidelným prírastkom na osiach x a y t.j.

$$x_{i+1} = x_i + dx, \; y_{j+1} = y_j + dy.$$

Plochu aproximujeme systémom kriviek $z_i = f(x_i, y)$ a $z_j = f(x, y_j)$.

Priestorové riešenie viditeľnosti spočíva v postupnom zobrazovaní týchto kriviek funkcie spredu dozadu (front to back prístup). Preto určíme poradie kriviek pre vykresľovanie v závislosti od pozorovateľa, napríklad poradím z_i ($i = 1,..,m$) a z_j ($j = 1,..,n$). Najprv vykreslíme prvú krivku z_1 a postupne zobrazujeme ďalšie susedné krivky. Pri vykresľovaní musíme však sledovať obalové krivky a vykresľovať len tie časti, ktoré sú nad alebo pod nimi. Zobrazované krivky takto dotvárajú vždy nové a nové obalové krivky maxím a miním, tak ako to vidíme na obrazku 11.13.



Obr.11.13 Zobrazenie minimálneho a maximálneho obalu kriviek

Výpočet testu viditeľnosti sa zjednoduší, ak uvažujeme o rastrovej forme zobrazovaných kriviek. Potom pre každý skanovací stĺpec rastra ukladáme minimálnu a maximálnu hodnotu v poli min(s) a max(s). Pri vykreslení bodu (x_s, y_s) overíme, či skutočne platí

$$y_s < \min(s) \quad \text{alebo} \quad y_s > \max(s).$$

Ak áno, potom bod vykreslíme a zároveň upravíme minimálnu prípadne maximálnu hodnotu tohto stĺpca.

12

Achromatické a farebné svetlo

12.1 Úvod

Po odstránení neviditeľných plôch treba zobraziť viditeľné časti priestorových objektov určitými farebnými odtieňami. Ak by sme zobrazovali steny len zadanou farbou telesa, nedostali by sme skutočne reálny priestorový obraz, pretože vnímaný obraz závisí od zdroja svetla, povrchu materiálu a ešte ďalších fyzikálnych parametrov. Preto našou snahou bude popísať adekvátny model pre realistické zobrazovanie.

Prvú exaktnú teóriu svetla podal Huygens už v 17. storočí na základe vlnovej dĺžky a analógie šírenia zvukového vlnenia. Isaac Newton objavil rozklad bieleho svetla na spojité spektrum farieb pri prechode svetla cez sklenený hranol.

Najprv si popíšeme základné fyzikálne vlastnosti svetla, **lom a odraz svetla**. Uvedieme **farebné modely** a kolorimetrický priestor farieb, tak ako bol prijatý medzinárodnou komisou ISO pre osvetlenie CIE (*Commission International de l'Eclairage*) v roku 1931. Dáme do súvisu niektoré používané farebné modely a to RGB, CMY a HSV. V závere kapitoly ukážeme techniky, ktoré dokážu z obmedzenej palety farieb vytvoriť ilúziu viacfarebnej škály, špeciálne na čiernobielej tlačiarni vytvoriť paletu šedých farieb.

12.2 Základné fyzikálne vlastnosti svetla

To, čo vnímame okom ako svetlo, je z hľadiska fyziky úzke frekvenčné pásmo v oblasti elektromagnetického spektra. **Elektromagnetické vlnenie** sa zadáva dĺžkou periódy alebo frekvenciou. Súvis farby a dĺžkou periódy je daný rozsahom viditeľného vlnenia od 380 nm do 780 nm. Túto oblasť ohraničuje z jednej strany (pod 380 nm) **ultrafialové** a z druhej strany (nad 780 nm) **infračervené** žiarenie.

Svetlo sa šíri vlnoplochami, pre ktoré platí **Huygensov princíp**. **Svetelný lúč** je definovaný ako dráha svetla medzi dvoma bodmi. Pozdĺž smeru svetelného lúča sa šíria dve vlnenia: **elektrického vektora E** a **magnetického vektora B**, ktoré sa zhodujú vo fáze a sú na seba kolmé (obr. 12.1 a). Žiarenie má periodickú povahu a môžeme ho rozložiť na zložky so sínusovým priebehom monochromatického žiarenia. **Monochromatické žiarenie** sa prejavuje vždy jednou určitou farbou, preto sa nazýva aj jednofarebné svetlo alebo **spektrálna farba**. **Spektrálna krivka** je graf intenzity v závislosti od vlnovej dĺžky.

Geometrická optika je založená na zákonoch odvodených z pozorovaní a skúseností. Sú to nasledujúce štyri zákony: **zákon priamočiareho šírenia svetla**, **zákon o vzájomnej nezávislosti lúčov**, **zákon odrazu** a **zákon lomu**.

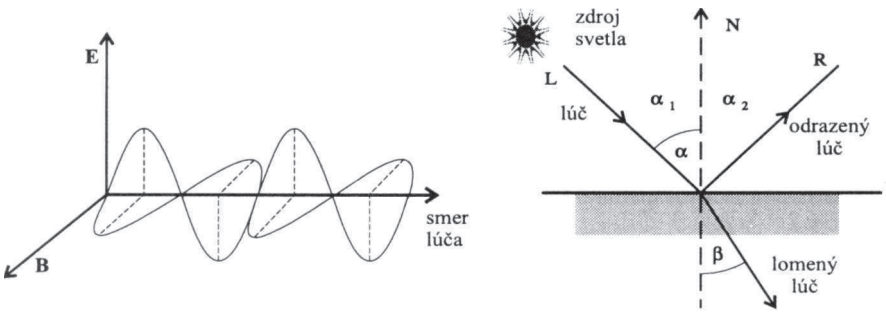
Z Huygensovho princípu vyplýva, že odrazený lúč leží v rovine dopadu (určenej normálovým vektorom v danom bode dopadu). Postupuje na druhú stranu od kolmice dopadu a zviera s ňou rovnaký uhol ako dopadajúci lúč (pozri obr. 12.1 b). Ak uvažujeme neorientované uhly, potom ich vzťah môžeme zapísať rovnosťou:

$$\alpha_1 = \alpha_2.$$

Podľa zákona lomu lúč, ktorý prechádza z jedného prostredia do druhého prostredia, mení na rozhraní oboch prostredí svoj smer (**Snellov zákon**). Tento zákon sa dá zapísať matematicky nasledujúcou rovnosťou:

$$\frac{\sin\alpha}{\sin\beta} = \frac{v_1}{v_2} = \frac{n_t}{n_i},$$

kde v_1, v_2 sú rýchlosti svetla v týchto prostrediach a α je uhol dopadu a β je uhol lomu. Index lomu n_t udáva pomer rýchlosti svetla vo vákuu k rýchlosti v danom prostredí. Závisí na vlnovej dĺžke a jeho hodnoty sa pohybujú v rozpätí od 1 pre plyny, cez približne 1,6 pre sklo až po viac než 2 pre diamant.



a) b)

Obr. 12.1 Odraz a lom svetla

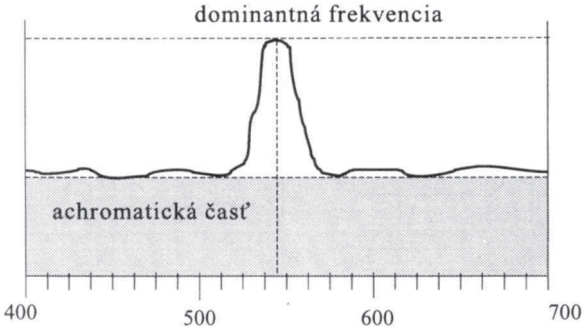
Ako každé vlnenie i žiarenie sa vyznačuje prenosom energie. Tieto vlastnosti žiarenia popisujú energetické veličiny. Fotometrické veličiny rešpektujú rôznu citlivosť oka na svetlo rôznych vlnových dĺžok a posudzujú svetlo z hľadiska zrakového vnemu. Nás budú zaujímať iba niektoré z existujúcich veličín.

Fotometrické pojmy sa odvodzujú od svetelného toku (Φ), čo je **žiarivý tok** zhodnotený normálnym ľudským okom vzhľadom na rozdielne citlivosti na rôzne farby. Podrobnosti k fyzikálnym veličinám pre osvetlenie uvedieme v 13. kapitole. Je všeobecne známe, že infračervené a ultrafialové žiarenie nevyvoláva v oku žiadne zrakové vnemy ani pri väčšej intenzite. Ľudské oko vníma najcitlivejšie zelené svetlo (s vlnovou dĺžkou približne 555 nm). V nasledujúcej časti sa sústredíme na fyziologické vnímanie svetla a farby.

12.3 Fyziologické vlastnosti svetla

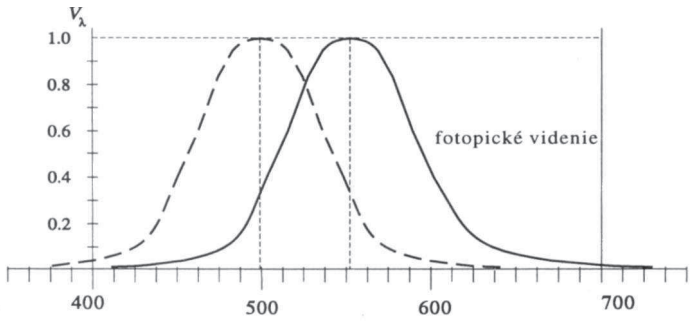
Na popis charakteristík svetla sú užitočné okrem frekvencie i ďalšie vlastnosti. Ak pozorujeme zdroj svetla, naše oko reaguje nielen na farbu, ale i na ďalšie podnety: jas, sýtosť a svetlosť.

Jas, zodpovedá intenzite svetla. Čím vyššia je intenzita svetla, tým sa javí zdroj jasnejšie. **Sýtosť** udáva čistotu farby svetla. Je tým vyššia, čím užšie je spektrum farebných frekvencií obsiahnutých v svetle. **Svetlosť** určuje veľkosť achromatickej zložky obsiahnutej v svetle s určitou dominantnou frekvenciou. Pod dominantnou frekvenciou rozumieme príslušnú dĺžku, v ktorej dosahuje spektrálna hustota farieb maximum. Čistotu farby definujeme ako pomer energie dominantnej frekvencie k celkovej energii určenej bielou farbou - achromatickou časťou (pozri obr. 12.2 a 12.4). Jas definujeme ako plochu, ktorá sa nachádza pod krivkou zadanou spektrálnou hustotou energií.



Obr. 12.2 Dominantná frekvencia a achromatická časť

Označme K_m *maximálnu svetelnú účinnosť* ľudského oka t.j. pre spektrálnu farbu s vlnovou dĺžkou 555 nm a K_λ svetelnú účinnosť jednofarebného žiarenia vlnovej dĺžky λ, potom pomer $V_\lambda = K_\lambda / K_m$ nazývame *pomernou svetelnou účinnosťou*. Na obr. 12.3 sú znázornené priemerné hodnoty tejto pomernej svetelnej účinnosti ľudského oka.

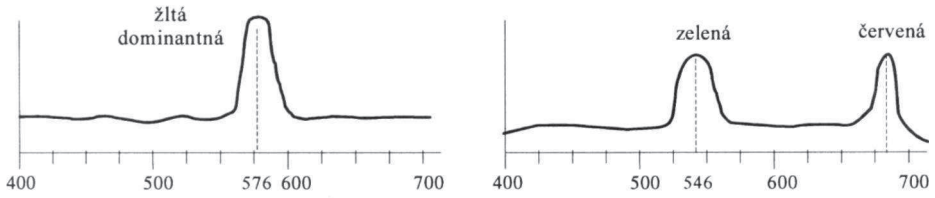


Obr. 12.3 Fotopické a skopické videnie

Tieto sa však líšia pri dennom osvetlení (**fotopické videnie**, keď vnímame čípkovými telieskami v oku - graf ——) a za šera (**skotopické videnie**, keď vnímame tyčinkovými telieskami v oku - graf - - -).

Experimentálne boli zisťované reakcie oka na farby. Ukázalo sa, že v ľudskom oku sú tri typy zrakových buniek, ktoré sú citlivé hlavne na farbu blízku modrej (okolo 425 nm), zelenej (530 nm) a červenej (650 nm), pričom oko je najcitlivejšie na zelenú farbu. To ilustruje aj graf na obr. 12.2, na ktorom vidíme spektrálnu citlivosť oka, t.j. súčet citlivostí všetkých troch receptorov. Človek dokáže rozlíšiť od 300 000 až do 400 000 rôznych farebných odtieňov, avšak v prípade čistých farieb sa tento počet redukuje približne na 150 farieb (individuálne u človeka od 100 až do 200 farieb).

Ak by sme chceli definovať farby len pomocou *spektrálnej hustoty energií,* potom na nasledujúcom obr. 12.4 vidíme prípad nejednoznačnosti znázorňovanej farby. Na obrázku a) máme znázornenú čistú žltú farbu a na obrázku b) je zobrazená zložená farba pomocou červenej a zelenej. Ľudské oko ich však vníma rovnako. **Aditívnym skladaním** môžeme dostať dve svetlá s rôznym spektrálnym zložením, ale rovnakou vnímanou farbou. Preto sa vychádza z experimentálneho skladania farieb.



a) b)

Obr. 12.4 Porovnanie aditívneho skladania farieb

Existuje aj iný spôsob skladania farieb a to **subtraktívne**. Toto skladanie farieb môžeme realizovať prechodom svetla postupne cez farebné filtre. Napríklad keď necháme biele svetlo prechádzať postupne modrým a žltým filtrom, výsledné svetlo bude zelené. Pri aditívnom skladaní by sme získali z modrého a žltého svetla výsledné biele svetlo.

Ak pozorujeme svetlo vytvorené aditívnym skladaním svetiel z dvoch alebo viacerých zdrojov, vnímame výsledné svetlo s charakteristikami, ktoré sú určené pôvodnými zdrojmi. Ak použijeme dva svetelné zdroje s rôznymi farbami a intenzitami, môžeme vytvoriť škálu rôznych farieb. Je zrejmé, že kombináciami farieb možno vytvárať množstvo farebných odtieňov. Preto je prirodzená otázka, či nemožno nájsť základné farby, pomocou ktorých by bolo možné skladaním vytvoriť všetky existujúce farby.

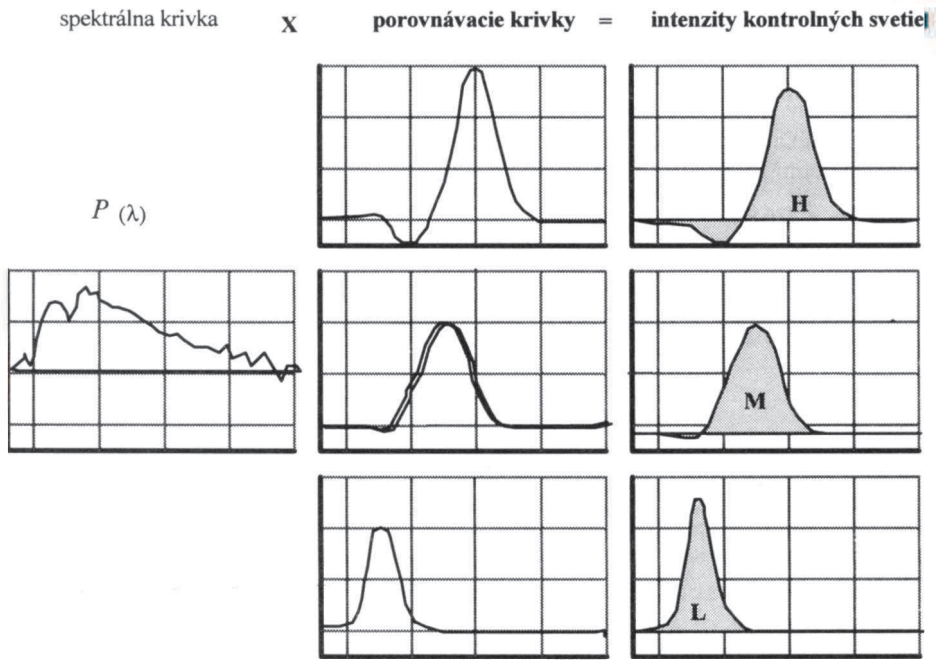
Z vlastností skladania farieb vyplýva, že neexistuje konečný počet farieb, pomocou ktorých skladaním by vznikli všetky ostatné farebné odtiene. Napriek tomu môžeme vybrať niekoľko základných farieb, ktoré tvoria základ pre veľkú časť existujúcich farieb. Experimentálne bol ich počet stanovený na tri.

Predstavme si, že na jednej strane od pozorovateľa leží testovacie svetlo ľubovoľnej farby a na druhej strane tri kontrolné svetlá (*H, M, L*) základných farieb, ktoré skladáme

aditívne [HALL 89]. Jedinou podmienkou je, aby sa ani jedno kontrolné svetlo nedalo získať kombináciou zvyšných dvoch. Regulovaním intenzít kontrolných svetiel sa snažíme získať farbu testovacieho svetla. Bolo by užitočné, keby sme tieto intenzity vedeli predpovedať z danej spektrálnej krivky testovacieho svetla.

Na dosiahnutie nášho cieľa sa používajú tzv. **porovnávacie krivky** kontrolných svetiel. Sú to grafy intenzít kontrolných svetiel ako funkcie vlnovej dĺžky. Ich vynásobením so spektrálnou krivkou testovacieho svetla a následným integrovaním dostávame požadované intenzity (obr.12.5):

$$H = \int P(\lambda).\, h(\lambda)\, d\lambda, \quad M = \int P(\lambda).\, m(\lambda)\, d\lambda, \quad L = \int P(\lambda).\, h(\lambda)\, d\lambda.$$



Obr. 12.5 Určenie intenzít kontrolných svetiel pomocou porovnávacích kriviek

12.4 Farebné modely

Na určenie farieb existuje viacero fariebných atlasov. Najznámejší Munsellov atlas obsahuje farby zoradené podľa *tónu farby*, *čistoty farby* a *jasu*. Pri rozklade bieleho svetla na farebné spektrum vnímame svetlo čistých farieb od modrej až po červenú. Rozlišujeme šesť základných oblastí farieb: fialovú (*magenta*), modrú (*blue*), modrozelenú (cyan), zelenú (*green*), žltú (*yellow*) a červenú (*red*).

Je známe, že väčšina reálnych farieb môže byť vytvorená pomocou 3 základných farieb, a to buď červenou (*R*), zelenou (*G*) a modrou (*B*) v aditívnom **RGB** modeli alebo modrozelenou (*C*), fialovou (*M*) a žltou (*Y*) v subtraktívnom **CMY** modeli.

Aditívny model RGB sa najčastejšie používa pre farebné monitory (ide o sčítanie intenzít svetelných zdrojov), zatiaľ čo subtraktívny systém CMY sa používa pri zázname napr. na filmový materiál alebo na papier pomocou farebných atramentov (farby sa odčítavajú od bieleho pozadia). Medzinárodná komisia pre osvetlenie **CIE** stanovila vlnové dĺžky základných farieb takto:

modrá (*B*) - 435,8 nm, zelená (*G*) - 546,1 nm, červená (*R*) - 780,0 nm.

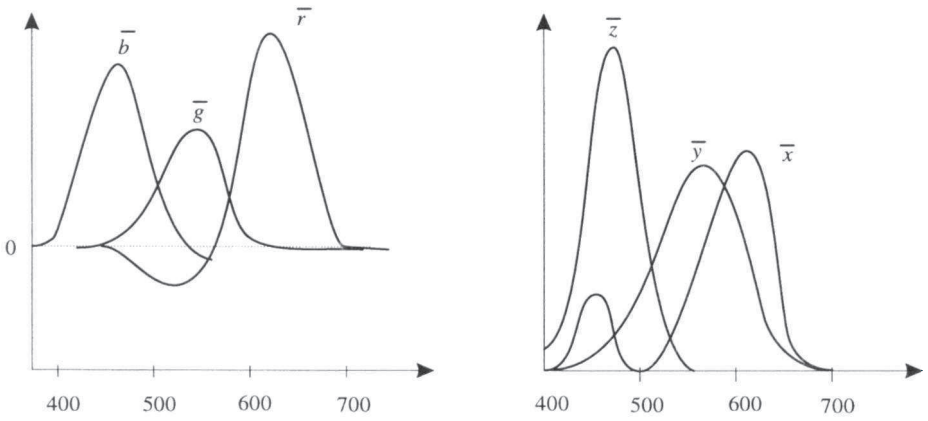
12.4.1 Model XYZ

V roku 1931 medzinárodná komisia CIE zaviedla tri základné farby *X*, *Y*, *Z*, ktoré sú číselne definované [COLO88] a ich porovnávacie krivky *x*, *y*, *z* sú znázornené na obr. 12.6. Na obrázku vidíme, že sa podľa dominantnej frekvencie blížia k farbám *R*, *G*, *B*. Zavedením nových virtuálnych súradníc sa vyhneme problému záporných koeficientov, avšak niektoré farby budú mať sýtosť väčšiu ako 100 %, a ide teda o farby neskutočné (nie je ich možné získať rozkladom bieleho svetla). Komisia CIE tiež definovala prevod medzi modelmi RGB a XYZ:

$$X = 2,7689 \cdot r + 1,7518 \cdot g + 1,1302 \cdot b$$

$$Y = 1,0000 \cdot r + 4,5907 \cdot g + 0,0601 \cdot b$$

$$Z = 0,0000 \cdot r + 0,0565 \cdot g + 5,5943 \cdot b$$

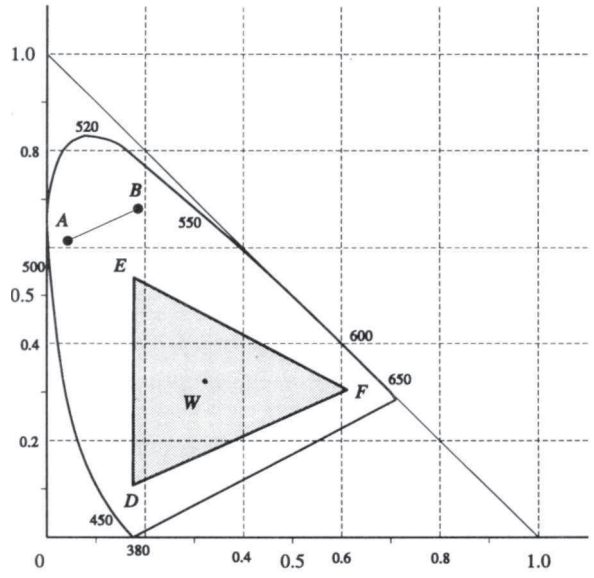


Obr. 12.6 Graf porovnávacích farieb r, g, b a x, y, z

Pre XYZ model farieb môžeme uskutočniť normalizáciu nasledujúcim spôsobom:

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z}, \quad z = \frac{Z}{X+Y+Z}$$

V tomto prípade platí $x + y + z = 1$ (t.j. normalizované premenné sú závislé), a preto nás bude napríklad zaujímať priemet normalizovaných bodov do roviny xy. Krivku reprezentujúcu čisté farby nazývame CIE diagramom. Biela farba vzniká zložením všetkých farieb a je v ťažisku. Tento bod W nazývame **štandardným zdrojom** (pozri obr 12.7).



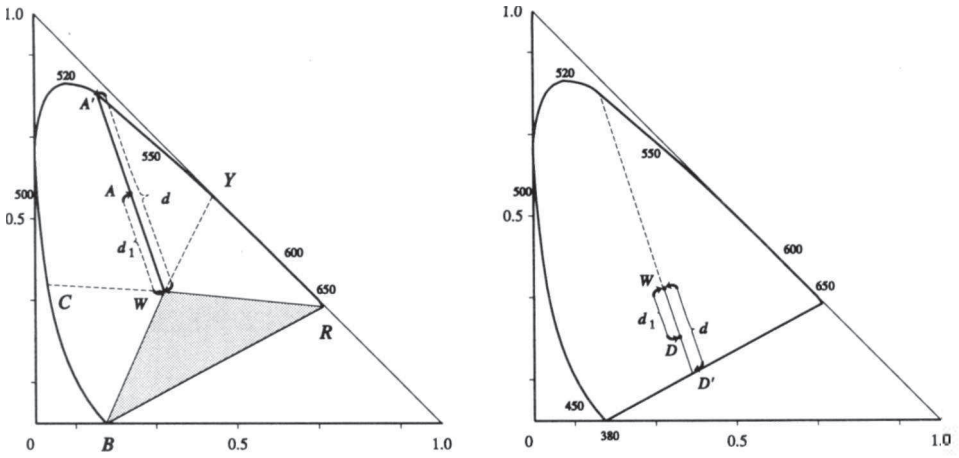
Obr. 12. 7 CIE diagram a aditívne skladanie farieb

Farebné rozsahy sú v chromatickom diagrame reprezentované úsečkami, ktoré spájajú farebné body určené základnými farbami. Na obrázku 12.7 je nakreslená spojnica medzi bodmi A a B. Všetky body na tejto úsečke možno získať aditívnym zložením týchto dvoch farieb. Podobne možno vytvoriť farby zložením farieb D, E a F ako vnútorné body trojuholníka DEF kombináciou základných farieb jeho vrcholov. Ako vyplýva z geometrie chromatického diagramu, nie je možno nájsť také tri základné farby, ktoré by určovali všetky farby, pretože ich konvexný obal (trojuholník) nemôže pokrývať celý CIE diagram.

Diagram CIE nám umožňuje určiť **dominantnú frekvenciu** a **čistotu farby**. Ak máme zadanú ľubovoľnú farbu, potom v normalizovaných súradniciach sa nám zobrazí do niektorého bodu A, ktorý sa nachádza vo vnútri plochy (obr. 12.8). Spojíme štandartný zdroj svetla (bod W) s bodom A a priesečník polpriamky WA s krivkou CIE nám dáva čistú farbu, ktorá určuje *dominantnú frekvenciu* pre farbu zadanú bodom A. *Čistotu farby* určíme ako pomer:

$$d_1 / d,$$

kde d_1 je vzdialenosť bodov AW a d je vzdialenosť bodov $A'W$.



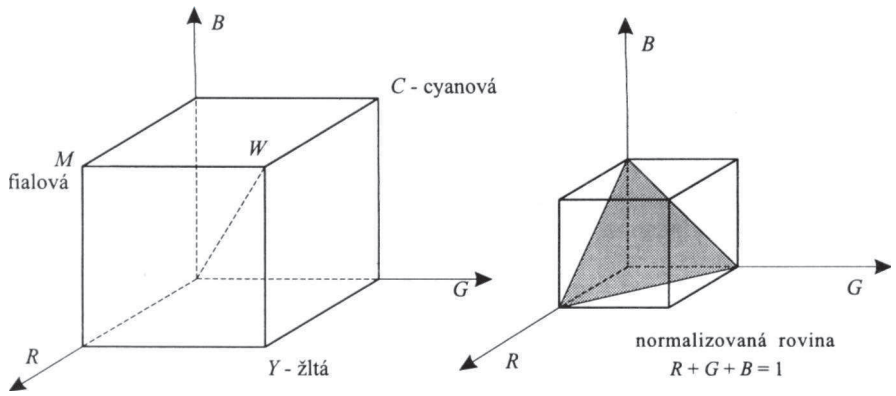
Obr. 12.8 Určenie čistoty farby pomocou CIE diagramu

Doplnkovými farbami sa nazývajú také dve farby, ktoré po zložení dajú bielu farbu. Na obr. 12.8 vidíme, že doplnkovou farbou červenej je cyanová (*R-C*), zelenej fialová (*G-M*) a modrej žltá (*B-Y*). Na obrázku vidíme, že fialová farba nie je čistá farba, ale skladaním dvoch farieb: červenej a modrej. Každý odtieň fialovej farby takto môžeme vyjadriť ako lineárnu kombináciu na úsečke *RB*. Pre fialovú farbu, ako napríklad pre bod *D*, sa určí čistá farba predĺžením za bod, t.j. na priesečníku polpriamky *DW* s krivkou CIE t.j. bod čistej doplnkovej farby. Čistota sa určí ako predtým, ale so záporným znamienkom. Takýto postup určenia dominantnej frekvencie a čistoty sa musí robiť v trojuholníku *BWR*.

12.4.2 Model RGB

Ak si vyberieme tri základné farby červenú (*R*), zelenú (*G*) a modrú (*B*), potom ostatné farby vyjadríme pomocou váhového súčtu jednotlivých zložiek. Ak zoberieme do úvahy viditeľné spektrum vlnových dĺžok, potom každej vlnovej dĺžke zodpovedá istá farba. Na obr. 12.6 sú znázornené priebehy jednotlivých váhových koeficientov *r*, *g*, *b*, ktoré sa tiež nazývajú **trichromatickými spektrálnymi súradnicami**. Ich hodnoty, získané kolorimetrickými meraniami, sú určené tak, aby sme dostali príslušné farby zodpovedajúce jednotlivým vlnovým dĺžkam vo viditeľnom spektre (380 - 780 nm).

Poznamenajme, že trichromatické spektrálne súradnice závisia na danom pozorovateľovi. Z obr. 12.6 a 12.7 vyplýva, že nie všetky farby sú reprezentovateľné pomocou skladania troch základných farieb, pretože pre určitú časť vlnových dĺžok viditeľného spektra je hodnota koeficientov záporná.



Obr. 12.9 Model RGB farieb a normalizovaná rovina

Farebný rozsah môžeme v modeli RGB zobraziť priestorovo pomocou jednotkovej kocky umiestnené v osiach označených ako *r*, *g* a *b* (obr. 12.9). Začiatok súradníc zodpovedá čiernej farbe, kým vrchol so súradniciami (1, 1, 1) zodpovedá bielej (*W*). Vrcholy kocky, ktoré ležia na osiach, predstavujú základné farby *R*, *G*, *B* a ostatné vrcholy kocky reprezentujú doplnkové farby *C*, *M* a *Y*.

V predchádzajúcom texte dosahovali úrovne jasu hodnoty od 0 po 1. Túto konvenciu je vhodné dodržať. Ak premietneme trichromatické súradnice do normalizovanej roviny (obr. 12.6), potom dostaneme nové súradnice :

$$\bar{r} = \frac{r}{r+g+b}, \quad \bar{g} = \frac{g}{r+g+b}, \quad \bar{b} = \frac{b}{r+g+b}.$$

Pre tieto je zrejmé, že môžeme vyjadriť tretiu súradnicu z dvoch predchádzajúcich. Pre určenie druhu farby, t.j. chromatičnosti (tónu a sýtosti bez udania jasu), teda stačí rovinné znázornenie farieb v rovine *RB*.

Úrovne šedých farieb tvoria hlavnú uhlopriečku kocky RGB, a preto zmenou jasu šedej sa pohybujeme po diagonále. Musíme si uvedomiť, že rovnako dlhé vektory rôznych farieb nezodpovedajú v skutočnosti rovnakým jasom, napríklad (obr. 12.6) pre pomer základných farieb dostávame pomer zodpovedajúcich farebných jasov ($|r| = |g| = |b| = 1$):

1 : 4,6 : 0,06

12.4.3 Model CMY

Podľa toho, aké 3 základné farby sa používajú hovoríme o systéme RGB alebo o systéme CMY. Ak zobrazujeme farebne na bielom podklade napríklad na papieri,

potom prechádzame na doplnkové farby CMY, pre ktoré sú podstatne lepšie výsledky farebného zobrazovania. Prepočet farieb RGB na farby CMY a naopak je jednoduchý:

$(C, M, Y) = (1\text{-}R, 1\text{-}G, 1\text{-}B)$,

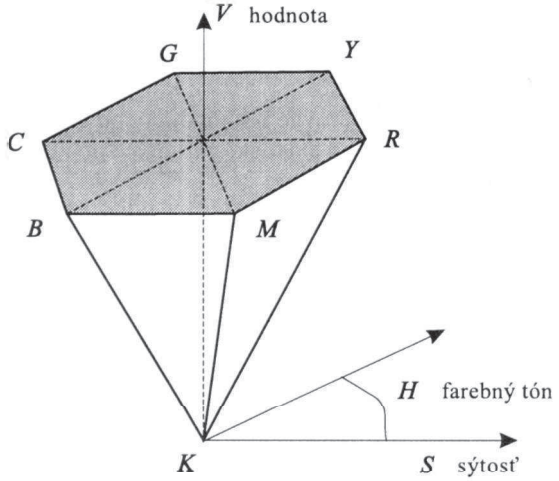
$(R, G, B) = (1\text{-}C, 1\text{-}M, 1\text{-}Y)$.

Kocka CMY je do istej miery zhodná s kockou RGB. Na osiach ležia základné farby a v ostatných vrcholoch kocky ležia k nim farby doplnkové. Hlavná diagonála obsahuje stupne šedej farby. Skladaním farieb však vzniká tmavší odtieň, takže vrchol (1,1,1) predstavuje čiernu farbu.

12.4.4 Model HSV

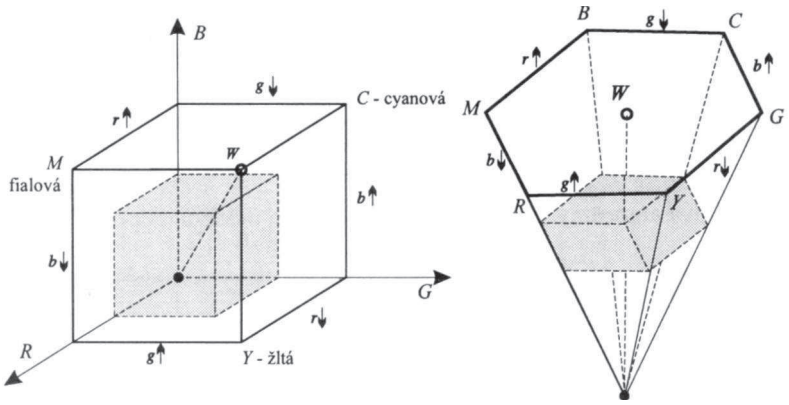
Tento model je vhodnejší pre intuitívne vnímanie farieb podľa **tónu farby** (*h-hue*), **sýtosti** (*s-saturation*) a **jasu** (*v-value*). Vychádza sa zo 6 základných farieb *R*, *G*, *B*, *C*, *M* a *Y*, ktoré sú umiestnené vo vrcholoch šesťuholníka.

Prevrátený šesťboký ihlan je umiestnený tak, že jeho vrchol leží v začiatku sústavy súradníc a podstava je v jednotkovej výške (obr. 12.10). Vrchol ihlana zodpovedá čiernej farbe (*K*-blac*K*), t.j. súradnica $v = 0$. Pre sýtosť môže byť definovaná ľubovoľná hodnota s z intervalu $\langle 0, 1 \rangle$. Bodu so súradnicami $s = 0$, $v = 1$ zodpovedá biela farba (*W*). Medzi týmito dvoma bodmi (bielou a čiernou farbou) sú body šedej farby.



Obr. 12.10 Model HSV zobrazený ako šesťboký ihlan

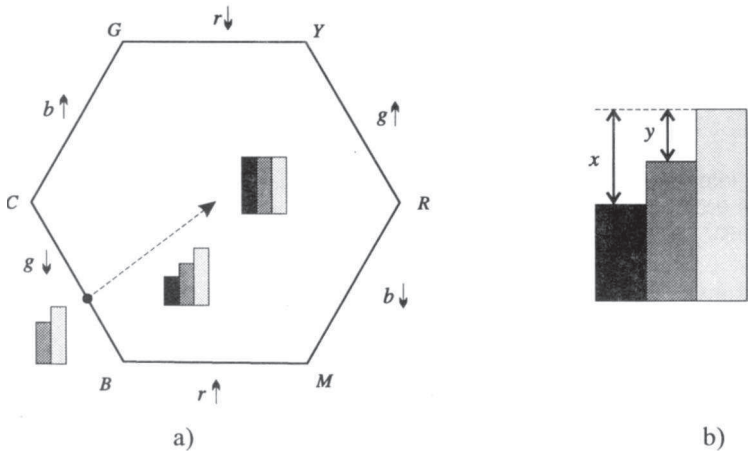
Ak s sa rovná nule, potom hodnota h nie je definovaná a zodpovedá podľa hodnoty v niektorej šedej farbe. Ak s je rôzne od nuly, hodnota h je z intervalu $\langle 0, 360^\circ \rangle$. Napríklad, červená farba má súradnice $h = 0$, $s = 1$, $v = 1$. Pridanie bielej farby do čistej farby spôsobuje zväčšenie hodnoty v.



Obr. 12.11 Transformácia RGB kocky na HSV ihlan

Ukážeme transformáciu RGB kocky na HSV ihlan. Podstavu šesťbokého ihlana predstavuje šesťuholník, ktorý možeme vidieť, ak sa pozeráme pozdĺž hlavnej diagonály RGB kocky z vrchola, zodpovedajúceho bielej farbe. Táto transformácia je zobrazená na obrázku 12.11. Ak sa pozeráme na menšiu kocku pozdĺž jej hlavnej diagonály, javí sa nám ako menší šesťuholník. Každá rovina s konštantnou súradnicou v v priestore HSV vytvára šesťuholník, zodpovedajúci zobrazeniu menšej kocky v priestore RGB. Z predchádzajúceho obrázku 12.11 môžeme sformulovať základné pravidlá závislosti r, g, b súradníc od zmeny polohy bodov v sústave HSV :

1. pri cyklickej zmene tónu h sa mení vždy len jedna zo súradníc r, g, b (na obr. 12.11 a 12.12 je to znázornené šípkou)
2. veľkosť najväčšej zo súradníc r, g, b je zhodná s hodnotou v
3. na hranách rezu (na plášti ihlanu) má vždy aspoň jedna zo súradníc r, g, b hodnotu 0, v modeli HSV pre tieto body platí $s = 1$



Obr. 12.12 Pomer hodnôt r, g, b pri zmene sýtosti s

Na obr. 12.12 potom môžeme sledovať závislosť *r*, *g*, *b* pri zmene *s*, keď *h* i *v* zostávajú nezmenené:

1. pri zmene *s* sa menia len dve menšie zložky z trojice *r*, *g*, *b*, a to lineárne,
2. pomer rozdielov menších zložiek a maximálnej zložky je pri zmene *s* konštantný, ide o pomer *x*/*y* na obr. 12.12 b).

Doleuvedené dva algoritmy určujú prepočet z jedného modelu na druhý:

Algoritmus prevodu RGB na HSV

```
{ Zadané sú hodnoty r, g, b, každá z intervalu [0,1]                                    }
{ Treba určiť h z intervalu [0,360], s a v z intervalu [0, 1], vylúčiac s = 0, pre ktoré h je
    nedefinované                                                                        }
procedure RGB_TO_HSV ( r, g, b : real ; var h, s, v : real );
 var max, min, delta: real;
begin
    max:= MAXIMUM (r, g, b);
    min := MINIMUM (r, g, b);
    v := max;                                   { jas }
    if max <> 0 then s := (max - min)/max       { sýtosť }
        else s := 0;
    if s = 0 then h := undefined
        else                        { sýtosť sa nerovná nule, preto určujeme farebný tón }
        begin
            delta:=max-min;
            rc  := (max-r)/delta;               { rc je vzdialenosť farby od červenej }
            gc := (max-g)/delta;
            bc := (max-b)/delta;
            if r = max then h := bc - gc        { výsledná farba leží medzi žltou a
                                                      fialovočervenou }
            else if g = max then h := 2 + rc - bc   { výsledná farba leží medzi
                                                      bledomodrou a žltou }
            else if b = max then h := 4 + gc - rc;  { výsledná farba leží medzi fialovou a
                                                      bledomodrou }
            h := h*60                           { prevedieme na stupne }
            if h < 0 then h := h + 360          { urobíme kladným }
        end                                     { chromatický prípad }
end.


{ Zadané h z intervalu [0, 360] alebo undefined, s a v z intervalu [0, 1] }
{ Je potrebné určiť hodnoty r, g, b, z ktorých každé je z intervalu [0, 1] }
procedure HSV_TO_RGB ( var r, g, b : real ; h, s, v : real);
begin
    if s = 0 then                   { achromatická farba, chýba farebný tón }
        if h = undefined then
        begin                       { achromatický prípad - odtiene šedej }
            r := v;
            g := v;
            b := v;
        end
        else ERROR                  { ak s = 0 a h je definované, tak nastala chyba }
```

```
    else                                    { chromatická farba - má farebný tón }
        begin
            if h = 360 then h := 0;
            h := h/60;                      { h teraz leží medzi [0, 6] }
            i := FLOOR (h);                 { celá časť h - teraz 0,..,6 }
            f := h - i;                     { zvyšok }
            p := v* (1 - s);                { pomocné premenné }
            q := v*(1 - (s*f));
            t := v*(1 - (s*(1-f)));
            case i of
                0: (r, g, b) := (v, t, p);  { zadanie r, g, b }
                1: (r, g, b) := (q, v, p);
                2: (r, g, b) := (p, v, t);
                3: (r, g, b) := (p, q, v);
                4: (r, g, b) := (t, p, v);
                5: (r, g, b) := (v, p, q);
            end {case}
        end { farebný tón }
end { HSV_TO_RGB}.
```

13

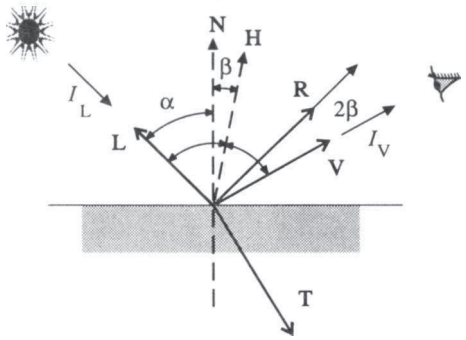
Osvetľovacie modely a tieňovanie

13.1 Úvod

Človek vníma farby v závislosti od vlastnosti svetla, ktoré na teleso dopadá a od intenzity svetla, ktoré sa od telesa odráža a vyvoláva vnem na sietnici oka. V poslednom období zaznamenávame vzostup záujmu o reálne zobrazovanie scén. Odráža sa to vo vývoji globálnych i lokálnych osvetľovacích modelov. Úlohou globálneho modelu je určiť, ktorý z objektov zobrazovanej scény vplýva na daný bod. Lokálne osvetľovacie modely popisujú vzťah medzi dopadajúcim a opúšťajúcim svetlom povrchu v danom bode v závislosti od smeru, intenzity a vlnovej dĺžky svetla.

Svetlo má mnoho spoločných vlastností s rozsiahlym oborom fyzikálnych javov, ktoré nazývame elektromagnetické žiarenie. Uvedieme fyzikálne vlastnosti svetelného žiarenia a charakterizáciu povrchu telesa pre odraz. Budeme sa zaoberať lokálnymi svetelnými modelmi zobrazovania, ktoré budeme využívať v nasledujúcich kapitolách pre algoritmus **sledovania lúča** (*ray-tracing*) a **radiačnú metódu** (*radiosity*).

V nasledujúcich častiach používame označenie vektorov podľa obrázku 13.1: **N** - normála povrchu, **V** - vektor pozorovateľa, **L** - vektor svetelného zdroja, **R** - vektor odrazu, **H** - bisektor medzi vektormi **L** a **V**, **T** - vektor lomu. Všetky vektory sú normalizované a smerujú od povrchu telesa (okrem **T**). Intenzitu dopadajúceho svetla označíme I_L a odrazeného svetla I_V.



Obr. 13.1 Označenie vektorov

13.2 Empirické osvetľovacie modely

Uviedli sme niektoré zo základných fyzikálnych princípov, dotýkajúce sa povahy svetla a vzťahy platiace pri dopade svetla na povrch zatiaľ nešpecifikovaného objektu. Lokálne osvetľovacie modely používané v praxi sa líšia od teoretických znalostí. Podľa stupňa rešpektovania fyzikálnych vzťahov rozdeľujeme lokálne osvetľovacie modely do troch skupín.

Prvou skupinou sú **modely empirické**. Sú to modely pomerne nenáročné na počítanie a niekedy s pomerne realistickými výsledkami, ale minimálne rešpektujú skutočné fyzikálne deje. Väčšinou sa spočítajú až po transformovaní objektov do 2D priestoru obrazovky. Používa ich Gouraudovo alebo Phongovo tieňovanie.

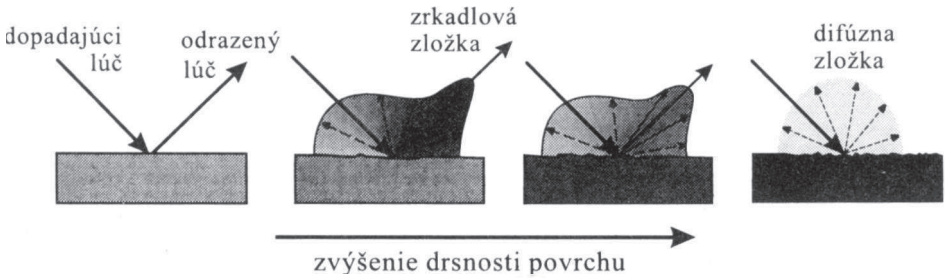
Do druhej skupiny patria **modely prechodové**, s fyzikálnym základom. Spolu s rastúcou silou hardveru poskytujú nesporne realistickejšie výsledky ako predchádzajúca skupina. Lokálny osvetľovací model sa spočíta ešte pred transformáciou do 2D priestoru, preto sú odraz, lom i tiene geometricky korektné. Typickým predstaviteľom tieňovacej techniky používajúcej tieto modely je **sledovanie lúča** (*ray-tracing*).

Poslednou skupinou sú analytické modely, zameriavajúce sa na fyzikálnu podstatu svetelných javov. Základným stavebným kameňom je šíriaca energia spolu so zákonmi, ktoré pre ňu platia. Tieto modely využíva **radiačná metóda**, ktorú podrobnejšiu uvedieme v nasledujúcej kapitole. Na **tieňovacej technike** závisí, ktorý z modelov bude aplikovaný.

13.2.1 Rozklad na zložky

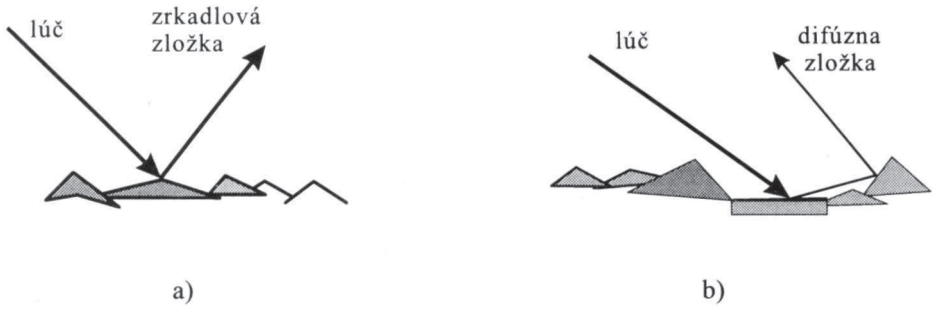
Vieme, že väčšina objektov, ktoré chceme zobrazovať, nemá hladký povrch. Charakterizovať drsný povrch však nie je jednoduché. Preto sa vytvoril ideálny drsný povrch, ktorý pozostáva z veľkého množstva dokonale hladkých rovinných plôšok, nazývajúcich sa mikroplôšky.

Pri osvetľovaní drsného povrchu dochádza k tieneniu a maskovaniu, ku ktorým nedochádza pri osvetľovaní hladkých povrchov. K tieneniu mikroplôšky dochádza vtedy, ak na ňu iná mikroplôška vrhá tieň. Maskovanie odrazeného svetla nastáva v prípade, ak odrazený lúč dopadá na inú mikroplôšku a nie je priamo vnímaný pozorovateľom [WATT 89]. Oba tieto javy znižujú intenzitu odrazeného lúča ako vidno na obr. 13.2.



Obr. 13.2 Odrazovosť ako funkcia drsnosti povrchu

Povrch telesa pri veľkom zväčšení vyzerá podobne ako je to znázornené na obr. 13.6. Lúč môže opustiť povrch telesa a dostať sa k pozorovateľovi dvoma spôsobmi. V prvom prípade sa lúč zrkadlovo odrazí od povrchu a v druhom prípade je viackrat odrazený prípadne lomený. Výslednú intenzitu odrazeného svetla vyjadríme ako súčet týchto dvoch zložiek.



Obr. 13.3 Odraz na mikroskopickej úrovni

Zrkadlová zložka (obr. 13.3 a) je charakterizovaná smerovosťou. Čím je povrch drsnejší, sú mikroplôšky viac zoskupené a smerovosť je menšia, a preto aj intenzita zrkadlovej zložky. Svetlo odrazené od dokonalého zrkadla má iba túto zložku.

Difúzna zložka (obr. 13.3 b) popisuje lúč, ktorý je podrobený viacnásobnému odrazu a lomu a výsledný smer odrazeného lúča je náhodný. Intenzita tejto zložky závisí iba na uhle dopadu a podľa Lambertovho zákona je priamo úmerná kosínusu tohoto uhla.

13.2.2 Bouknightov model

Najjednoduchší z modelov je Bouknightov z roku 1970. Používa iba difúznu a ambientnú zložku. **Ambientná zložka** reprezentuje okolité svetlo, ktoré vzniká mnohonásobnými odrazmi a rozptylmi a spôsobuje, že odvrátené povrchy nie sú úplne čierne. Základný vzorec pre intenzitu (1) pôvodne zahŕňal iba jeden zdroj svetla umiestnený v nekonečnej vzdialenosti od osvetľovaného bodu v smere pozorovateľa:

$$I(\lambda) = K_a + K_d(\mathbf{N}.\mathbf{L}) \qquad (K_a + K_d) < 1 \qquad (1)$$

Vzťah má zmysel iba v prípade, že skalaárny súčin vektorov $\mathbf{N}.\mathbf{L} > 0$. Tento vzťah môžeme jednoducho upraviť a brať do úvahy viacero svetelných zdrojov ľubovoľne umiestnených:

$$I(\lambda) = K_a(\lambda)I_a(\lambda) + K_d(\lambda)\sum_{n=1}^{l}(\mathbf{N}.\mathbf{L}_n)I_n(\lambda), \qquad (2)$$

kde l určuje počet zdrojov svetla v scéne, K_a je koeficient odrazu pre ambientné svetlo, K_d je koeficient difúzneho odrazu, I_a modeluje ambientné svetlo v scéne, I_n je intenzita n-tého svetelného zdroja.

13.2.3 Phongov model

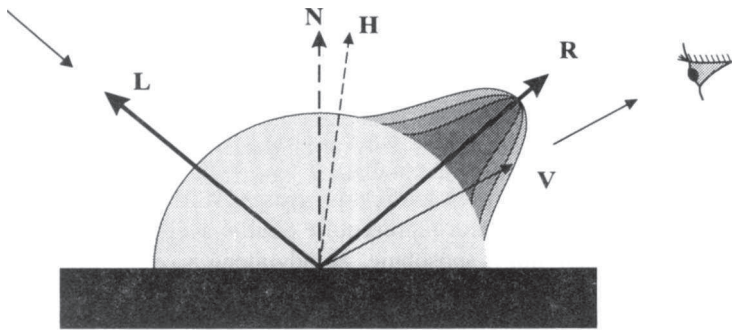
Klasickým modelom, najpoužívanejším v počítačovej grafike, je Phongov lokálny osvetľovací model. Okrem ambientnej a difúznej zložky zahrňuje i zložku zrkadlovú. Jej charakteristickou vlastnosťou je smerovosť a je príčinou vzniku odleskov.

Odlesky môžeme pozorovať ako svetlé miesta na zobrazovanom objekte. Sú odrazom svetelných zdrojov alebo iných svietiacich objektov. Empiricky môžeme tento jav modelovať funkciou, ktorá dosahuje maximum v zrkadlovom smere od vektora svetelného zdroja a rýchlo klesá v smeroch odkláňajúcich sa od zrkadlového smeru. Základný vzťah, ktorý odvodil Phong má tvar:

$$I(\lambda) = K_a(\lambda).I_a(\lambda) + K_d(\lambda).I_L(\lambda).(\mathbf{N}.\mathbf{L}) + K_s.I_L(\lambda).(\mathbf{R}.\mathbf{V})^k , \qquad (3)$$

K_s je koeficient zrkadlového odrazu a k je miera drsnosti povrchu.

Zrkadlová zložka je nezávislá na vlnovej dĺžke, a tak farba odleskov závisí iba na farbe zdroja svetla. Vektor **R** je symetrický k vektoru **L** podľa normály a možno ho vyjadriť zo vzťahu **R** = 2(**L**.**N**)**N** - **L**. Intenzita nadobúda maximum vtedy, ak smer dopadu lúča svetla a smer pozorovateľa spĺňajú zákon odrazu. Toto maximum je tým výraznejšie, čím je parameter k väčší. S rastúcim k sa odlesky na telese stávajú menšie a ostrejšie, pre dokonalé zrkadlo parameter k je nekonečno (obr. 13.4). V prípade **V**.**R** < 0, je pozorovateľ odvrátený od zrkadlovej zložky a tento súčin považujeme za nulový.



Obr. 13.4 Intenzita svetla v závislosti od parametra k

Zrkadlovú zložku možno na základe vzťahov pre uhly vektorov prepísať na analogický vzťah. Náhradnú formu zrkadlovej zložky zaviedol Blinn, použijúc bisektor medzi vektormi **V** a **L**. Zrkadlovú zložku vyjadril ako

$$\cos\beta = \mathbf{N}.\mathbf{H}, \text{ namiesto } \cos 2\beta = \mathbf{R}.\mathbf{V}$$

kde **H** je vlastne normála hypotetického povrchu (obr. 13.4). Hypotetický povrch je orientovaný tak, aby odrážal maximálne svetlo smerom k pozorovateľovi. Výsledný vzťah, zahrňujúci farbu zdrojov a materiálov a viac svetelných zdrojov umiestnených ľubovoľne v scéne, má tvar:

$$I(\lambda) = K_a(\lambda)I_a(\lambda) + K_d(\lambda)\sum_{n=1}^{l} I_n(\lambda).(\mathbf{N}.\mathbf{L}_n) + K_s\sum_{n=1}^{l} I_n(\lambda).(\mathbf{N}.\mathbf{H}_n)^k . \qquad (4)$$

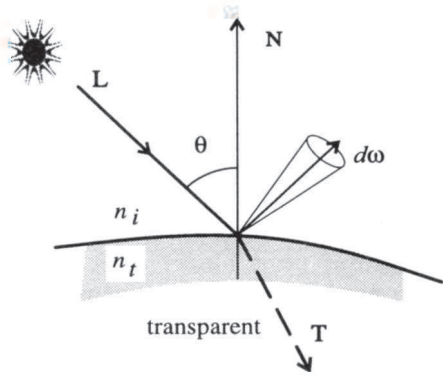
13.3 Model svetla

13.3.1 Fyzikálne vlastnosti svetla

Odraz a lom svetla vzniká pri dopade lúča na rozhranie dvoch prostredí. Dopadajúci lúč sa rozdelí na lúč odrazený a lomený. Pre odrazený lúč platí **zákon odrazu**, pre lomený lúč platí **Snellov zákon lomu**, podľa ktorého lomený lúč leží v rovine dopadu a pre uhly platí vzťah:

$$\sin\theta_t \cdot n_t = \sin\theta_i \cdot n_i,$$

kde n_i, n_t sú absolútne indexy lomu prostredí, v ktorých sa šíri lúč.



Obr. 13.5 Odraz a lom lúča

Žiarivý tok Φ je energia prechádzajúca určitou plochou za jednotku času. **Ožiarenie E** je definované ako žiarivý tok dopadajúci na jednotkovú plochu:

$$E = \frac{d\Phi}{dA},$$

kde $d\Phi$ je žiarivý tok dopadajúci na plochu dA (s jednotkou - $W.m^{-2}$). **Intenzita I** je definovaná ako žiarivý tok vyžarovaný v danom smere, vzhľadom na jednotkový priestorový uhol a jednotkový priemet plochy povrchu do smeru kolmého na smer vyžarovania:

$$I = \frac{d\Phi}{dA.\cos(\theta).d\omega},$$

kde $d\Phi$ je žiarivý tok, dA veľkosť plochy, θ je uhol vektora vyžarovania a normály, $d\omega$ priestorový uhol obsiahnutý vyžarovaním (s jednotkou - $W.sr^{-1}.m^{-2}$). Z predchádzajúcich vzťahov môžeme odvodiť vzťah medzi ožiarením a intenzitou:

$$E = (\mathbf{N} \cdot \mathbf{L}).I.d\omega.$$

Pomer odrazenej a prepustenej energie k energii dopadajúcej popisujú Fresnelove rovnice. Sú riešením Maxwellových rovníc pre správanie sa elektromagnetického vlnenia na hladkej hranici dvoch materiálov. Závisia od indexov lomu oboch prostredí, polarizácie dopadajúceho svetla a od uhla dopadu.

Pomer odrazenej energie k energii dopadajúcej F_r je určený vzťahom:

$$F_r = \frac{1}{2}\left(r_{\equiv}^2 + r_{\perp}^2\right) = \frac{\Phi_r}{\Phi_i},$$

pričom $r_{\equiv}$(resp. $r_{\perp}$) je pomer amplitúdy odrazených vĺn k dopadajúcim vlnám, pre polarizované svetlo rovnobežné s rovinou (resp. kolmé k rovine) určenej vektormi **L** a **N**. Tieto pomery Fresnel popísal vzťahmi:

$$r_{\equiv} = \frac{n_t(\mathbf{N}.\mathbf{L}) + n_i(\mathbf{N}.\mathbf{T})}{n_t(\mathbf{N}.\mathbf{L}) - n_i(\mathbf{N}.\mathbf{T})}, \qquad r_{\perp} = \frac{n_i(\mathbf{N}.\mathbf{L}) + n_t(\mathbf{N}.\mathbf{T})}{n_i(\mathbf{N}.\mathbf{L}) - n_t(\mathbf{N}.\mathbf{T})}.$$

Pomer prepustenej energie k energii dopadajúcej F_t sa vyjadrí jednoducho zákonom zachovania energie ako $F_t = 1 - F_r$.

13.3.2 Geometria povrchu

Všetky doteraz uvedené vzťahy sa týkali hladkých povrchov. Vieme však, že väčšina objektov, ktoré chceme zobrazovať nemá hladký povrch. Charakterizovať drsný povrch však nie je jednoduché. Preto sa vytvoril ideálny drsný povrch, ktorý pozostáva z veľkého množstva dokonale hladkých rovinných plôšok, nazývajúcich sa mikroplôšky. Drsnosť povrchu môžeme popísať funkciou $\zeta(x,y)$, ktorá každému bodu (x, y) povrchu priradí rozdiel výšok skutočného a priemerného povrchu v tomto bode. Uvedieme ďalšie tri veličiny popisujúce povrch telesa (obr. 13.6).

1. Priemerná odchýlka σ od priemernej výšky povrchu, pre ktorú platí:

$$\sigma = \left(\int\int \zeta^2(x,y)dxdy\right)^{\frac{1}{2}}.$$

2. Korelačná vzdialenosť τ definovaná ako priemerná vzdialenosť medzi lokálnym maximom a lokálnym minimom povrchu.

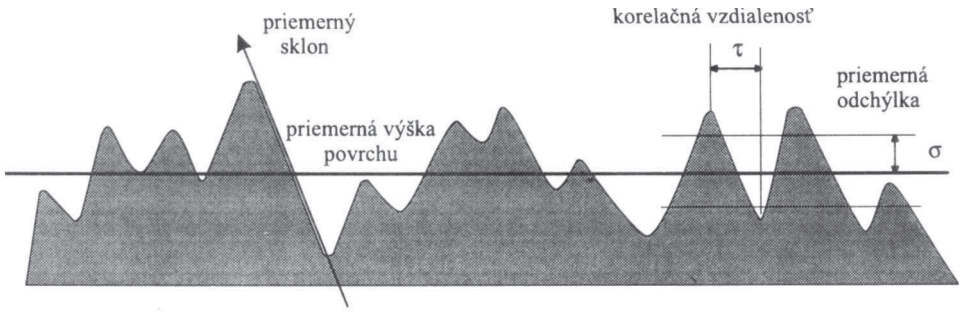
3. Priemerný sklon m vyjadrený vzťahom:

$$m = \frac{2\sigma}{\tau}.$$

Pre povrchy s funkciou ζ danou Gaussovským rozdelením možno popísať $\overline{m}$ vzťahom:

$$\overline{m} = \frac{\sigma\sqrt{2}}{\tau}.$$

Pri osvetľovaní drsného povrchu dochádza k tieneniu a maskovaniu, pri ktorých dochádza znižovaniu intenzity odrazeného lúča. **Geometrický útlmový faktor** G sa snaží vystihnúť toto zníženie intenzity.



Obr. 13.6 Geometria drsného povrchu

Torrance a Sparrow (1967) definovali v prípade, ak mikroplôšky majú tvar symetrického písmena V, funkciu G vzťahom:

$$G = \min\left(1, \frac{2(\mathbf{N}.\mathbf{H})(\mathbf{N}.\mathbf{V})}{(\mathbf{V}.\mathbf{H})}, \frac{2(\mathbf{N}.\mathbf{H})(\mathbf{N}.\mathbf{L})}{(\mathbf{V}.\mathbf{H})}\right)$$

Pre náhodne drsný povrch s gaussovským rozdelením výšky zaviedol Sancer (1969) náhradnú funkciu G:

$$G = \frac{1}{1 + C_i + C_r},$$

$$C_i = \frac{\exp(-c_1)}{2\sqrt{\pi c_1}} - \frac{1}{2}\mathrm{rand}(\sqrt{c_1}), \qquad c_1 = \frac{(\mathbf{N}.\mathbf{L})^2}{2\bar{m}^2\left(1 - (\mathbf{N}.\mathbf{L})^2\right)},$$

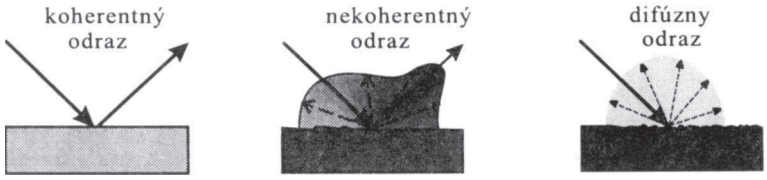
$$C_r = \frac{\exp(-c_2)}{2\sqrt{\pi c_2}} - \frac{1}{2}\mathrm{rand}(\sqrt{c_2}), \qquad c_2 = \frac{(\mathbf{N}.\mathbf{V})^2}{2\bar{m}^2\left(1 - (\mathbf{N}.\mathbf{V})^2\right)}.$$

13.4 Osvetlenie povrchu

Základnou požiadavkou, ktorú musí spĺňať fyzikálne korektný lokálny osvetľovací model je **zákon zachovania energie.** To znamená, že energia dopadajúca na povrch sa musí rovnať súčtu energií odrazenej a prechádzajúcej:

$$\Phi_i = \Phi_r + \Phi_t.$$

Ak je materiál nepriepustný, potom je energia prechádzajúca materiálom absorbovaná. Osvetľovanie možno rozdeliť na dve základné zložky: **koherentnú** a **nekoherentnú**. Treťou možnosťou je **difúzna** zložka, ale tá je iba špeciálnym prípadom nekoherentného osvetľovania (Obr. 13.7).



Obr. 13.7 Zložky osvetľovania pre odraz

13.4.1 Koherentné osvetlenie

Koherentné osvetľovanie prebieha na opticky hladkých povrchoch. Dopadajúce svetlo sa odráža podľa zákona odrazu a láme podľa Snellovho zákona lomu. Fresnelove vzťahy nám poskytujú koeficienty F_r a F_t, a výsledný vzťah pre osvetľovanie je:

$$\Phi_i = F_r\Phi_i + F_t\Phi_i \ , \qquad F_r + F_t = 1.0.$$

Použitím tohoto vzťahu, dostaneme pre intenzitu prichádzajúcu k pozorovateľovi z hladkého povrchu vzťah:

$$I_v = F_r I_r + F_t \frac{n_v^2}{n_t^2} I_t \ . \qquad (5)$$

Výsledná intenzita je vyjadrená ako súčet energií dopadajúcich na povrch zo zrkadlového a lomeného smeru vzhľadom na vektor pozorovateľa.

Opticky hladký povrch je v skutočnosti nedosiahnuteľný. Preto sa do vzťahu (5) pridávajú koherentné útlmové faktory pre odraz r_σ a lom t_σ a geometrický útlmový faktor G:

$$I_v = r_\sigma G F_r I_r + t_\sigma G F_t \frac{n_v^2}{n_t^2} I_t \ .$$

13.4.2 Nekoherentné osvetlenie

Nekoherentné osvetľovanie sa na celkovom osvetľovaní povrchu podieľa väčšou časťou ako koherentné. Dopadajúce svetlo sa v tomto prípade rozptyľuje vo všetkých smeroch od drsného povrchu. Objasnenie nekoherentného osvetľovania je pomerne náročné.

Definujme si dvojsmernú distribučnú funkciu R_{bd} pre odraz a T_{bd} pre lom. Tieto funkcie popisujú pomer odrazenej (prepustenej) intenzity k dopadajúcemu žiareniu. Závisia na vlnovej dĺžke, drsnosti povrchu a vektoroch **N**, **V**, **L**. Výsledný vzťah pre intenzitu I_v sa dá odvodiť ako:

$$I_v = \int^{2\pi} I_i R_{bd} \cos\theta_i d\omega_{front} + \int^{2\pi} I_i T_{bd} \cos\theta_i d\omega_{back}$$

K názornejšiemu popísaniu nekoherentného osvetľovania nám pomôže odraz a lom lúča a zavedený model drsného povrchu zloženého z mikroplôšok. Každá mikroplôška

sa správa ako opticky hladký povrch, ktorý je charakterizovaný koherentným osvetľovaním. Každá mikroplôška teda odráža svetlo zrkadlovo vzhľadom na svoju normálu a platia pre ňu Fresnelove rovnice. Na povrch sa pozeráme ako na súbor mikroplôšok. Príspevky jednotlivých mikroplôšok sa miešajú a povrch vnímame ako priemer interakcií všetkých mikroplôšok.

Na základe tohoto modelu stanovili Cook a Torrance funkciu pre R_{bd}:

$$R_{bd} = \frac{DGF_r}{\cos\theta_i \cos\theta_r} \quad ,$$

G a F_r poznáme, D je **distribučná funkcia normál mikroplôšok**. Popisuje pomer mikroplôšok povrchu, ktoré sú orientované tak, že normála je bisektor medzi dopadajúcim a odrazeným vektorom (v smere **H**). Funkcia D závisí na drsnosti povrchu a na uhle medzi normálov povrchu a normálov mikroplôšok **H**.

Osvetľovací model môžeme popísať ako pomer odrazenej žiarivosti a dopadajúceho ožiarenia:

$$R = \frac{DGF}{(\mathbf{N}.\mathbf{V})\ (\mathbf{N}.\mathbf{L})} \quad . \qquad (6)$$

Bližší popis oboch zložiek osvetľovania povrchu i konkrétne výsledky hľadaní distribučných funkcií možno nájsť v [HALL 89].

13.5 Lokálne fyzikálne modely

13.5.1 Transparentné modely

Transparentné lokálne osvetľovacie modely sú najčastejšie využívané pre metódu sledovania lúča. Okrem ambientnej, difúznej a zrkadlovej zložky sa na výslednej farbe podieľa i príspevok lomeného lúča. Všeobecný vzťah má tvar:

$$I(\lambda) = K_a(\lambda)I_a(\lambda) + K_d(\lambda)\sum_{n=1}^{l}(\mathbf{N}.\mathbf{L}_n)I_n(\lambda)$$

$$+ K_s(\lambda)\left(I_r(\lambda) + \sum_{n=1}^{l} f_r(\mathbf{V}, \mathbf{L}_n, \mathbf{N}, \zeta)I_n(\lambda)\right)$$

$$+K_t(\lambda)\left(I_t(\lambda) + \sum_{n=1}^{ls} f_t(\mathbf{V}, \mathbf{L}_n, \mathbf{N}, \zeta)I_n(\lambda)\right)$$

I_r a I_t reprezentujú osvetlenie zo smeru odrazu a lomu, f_r je smerová funkcia pre odraz, f_t je smerová funkcia pre lom a ζ je funkcia drsnosti povrchu.

13.5.2 Blinnov model

Blinnov lokálny osvetľovací model , podobne ako predchádzajúci, zahŕňa tri základné zložky - ambientnú, difúznu a zrkadlovú. Využijúc poznatky z fyziky a optiky, Blinn navrhol náhradnú formu zrkadlovej funkcie $f_r()$. Jej tvar je nasledovný:

$$f_r() = \frac{DGF_r}{(\mathbf{N}.\mathbf{V})} .$$

D je distribučná funkcia, popisujúca percento mikroplôšok, ktorých normály sú orientované v smere vektora **H** (časť 13.4.2), F_r je Fresnelova odrazivosť (časť 13.2), G reprezentuje geometrický útlmový faktor (13.3). Blinn uviedol tri možné tvary distribučných funkcií. Prvou je Phongova funkcia:

$$D = (\mathbf{R}.\mathbf{L})^k, \text{ alebo } D = (\mathbf{N}.\mathbf{H})^k .$$

Druhá distribučná funkcia je založená na gaussovskom rozdelení a je daná vzťahom:

$$D = \exp\left(-(C_1 \arccos(\mathbf{N}.\mathbf{H}))^2\right)$$

Tretia má tvar:

$$D = \left(\frac{C_2^2}{(\mathbf{N}.\mathbf{H})^2(C_2^2 - 1) + 1}\right)^2$$

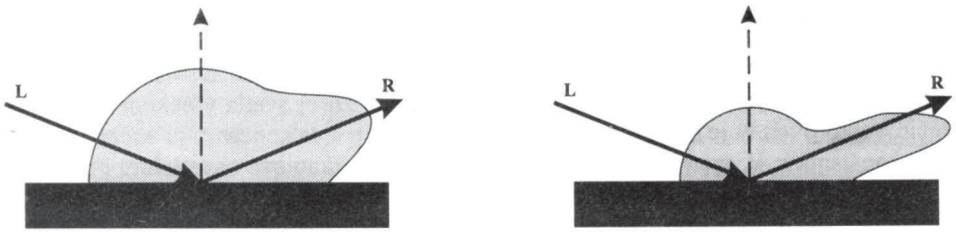
Konštanty k, C_1, C_2 odvodil Blinn v závislosti od uhla medzi vektormi **H** a **N**. Tento uhol označil ako β:

$$k = -\frac{\ln(2)}{\ln(\cos(\beta))},$$

$$C_1 = \frac{\sqrt{\ln(2)}}{\beta}, \qquad C_2 = \left(\frac{\cos^2\beta - 1}{\cos^2\beta - \sqrt{2}}\right)^{\frac{1}{2}}.$$

13.5.3 Cookov model

Cook [COOK82] použil ako zdroj informácií spektrálnu krivku materiálu a Fresnelove aproximácie. Hlavný výskum sústredil na fyzikálne zdôvodnenie nezrkaldových maxím pre uhly dopadu blízke 90°. Na obr. 13.8 máme znázornenú situáciu pre Phongov model a pre skutočné namerané hodnoty pre niektoré materialy pre uhol 70°.



Obr. 13.8 Zobrazenie plochy pre Phongov model a skutočné materiály

Cook zaviedol distribučnú funkciu v tvare:

$$D = \frac{1}{4m^2\,(\mathbf{N}.\mathbf{H})^4} \exp\left(\frac{1-1/(\mathbf{N}.\mathbf{H})^2}{m^2}\right),$$

kde parameter *m* sa vzťahuje k drsnosti materialu. Keď *m* je malé, potom funkcia *D* nadobúda ostré maximum. Pre Fresnelov činiteľ platí:

$$F = \frac{1}{2}\left(\frac{g-c}{g+c}\right)^2\left(1+\frac{c(g-c)-1}{c(g+c)+1}\right),$$

kde $c = \mathbf{L}.\mathbf{H}$, $g = \sqrt{\eta^2 + c^2 - 1}$ a $\eta = \frac{1+\sqrt{\mu}}{1-\sqrt{\mu}}$ pre odrazivosť μ.

Fresnelove rovnice sú teoreticky dobré a ľahko definovateľné, avšak ich použitie nie je také jednoduché pre realistické počítačové zobrazovanie. Príčinou je nedostatok vhodných materiálových dát a výpočtová náročnosť. Na riešenie týchto problémov navrhol Cook aproximačné metódy, zaoberajúce sa dvoma druhmi problémov. Prvým je aproximácia chýbajúcich informácií o materiáloch. Druhým je vhodné umiestnenie Fresnelových rovníc do lokálneho osvetľovacieho modelu.

Doteraz zavedené lokálne osvetľovacie modely nedokázali zohľadniť pomerne bežný jav skutočnosti - zrkadlenie. Whitted sa snažil postrehnúť práve tento jav, zavedením rekurzívneho ray-tracingu do počítačovej grafiky. V bode povrchu, ktorý vyhodnocujeme, zohľadnil informáciu získanú z povrchov nachádzajúcich sa v zrkadlovom a lomenom smere vzhľadom na dopadajúci lúč. Vypočítanie príspevkov týchto povrchov vyžaduje opakovať predchádzajúci postup pre každý z nich.

Lokálny osvetľovací model, ktorý prezentoval Hall a Greenberg, je najzložitejší z použitých modelov. Je podobný Blinnovmu osvetľovaciemu modelu, ale naviac zahŕňa osvetlenie zo zrkadlového a lomeného smeru a zoslabenie intenzity pri prechode materiálom. Používa tie isté materiálové vlastnosti ako Blinnov model, pridaná je informácia o zoslabení.

13.6 Tieňovanie

Tieňovaním (*shading*) rozumieme vykresľovanie objektov pomocou farebných odtieňov, ktoré prirodzene vznikajú pri zakrivených plohách a dávajú nám lepšiu predstavu o nich. Z predchádzajúcich modelov vieme určiť pre daný bod farbu Pre rýchlejší výpočet boli vypracované metódy, ktoré dokážu aproximovať farbu na základe vybratej vzorky bodov.

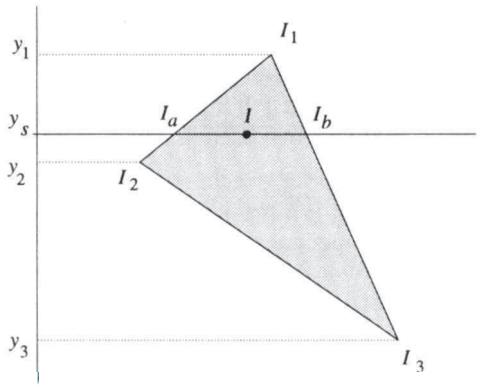
13.6.1 Konštantné tieňovanie (Constant Shading)

Najjednoduchšia metóda je prístup **konštantného tieňovania** [BOUK70]. Táto metóda vychádza z jednoduchého predpokladu, že ak je zdroj svetla v nekonečne, potom súčin vektorov L.N je konštantný a ak pozorovateľ je v nekonečne, je súčin vektorov L.N konštantný. V svetelnom modeli nie je zahrnuté zrkadlové osvetlenie. Pre kresby mnohostenov, alebo keď aproximujeme objekty lineárnymi plôškami, je konštantné tieňovanie postačujúce na prvotné zobrazenie umiestnenia objektov.

13.6.2 Gouraudovo tieňovanie

Gouraud navrhol metódu interpolácie intenzít [GOUR71]. Metódu môžeme použiť pre mnohosteny, ktoré aproximujú zakrivené objekty. Najprv vypočítame farbu zobrazenia vo vrcholoch takého mnohostena. Ak vrchol V je spoločným vrcholom niekoľkých stien, ktoré sú vytvorené pre aproximáciu oblého telesa, potom normálu v tomto vrchole určíme ako priemer normál vytvorených pre jednotlivé incidentné steny tohto vrchola. Preto pre spoločný vrchol V stien $s_1, s_2, ..., s_n$ vypočítame **normálový vektor vo vrchole** podľa vzťahu (priemer normálových vektorov $\mathbf{N}_i$):

$$\mathbf{N}_V = \frac{\sum_{i=1}^{n} \mathbf{N}_i}{\left|\sum_{i=1}^{n} \mathbf{N}_i\right|}.$$



Obr. 13.9 Interpolácia intenzity pre trojuholník

V ďalšom kroku určíme **intenzitu vrcholu** pomocou niektorého osvetľovacieho modelu. Nakoniec každý polygón tieňujeme lineárnou interpoláciou intenzít určených v jeho vrcholoch. Najlepšie si to ukážeme na trojuholníku pomocou rozkladu do riadkov. Na obr. 13.9 máme určené intenzity I_1, I_2, I_3 vo vrcholoch trojuholníka. Najprv vypočítame intenzity I_a, I_b na zvolenom riadku $y = y_s$ v krajných bodoch trojuholníka:

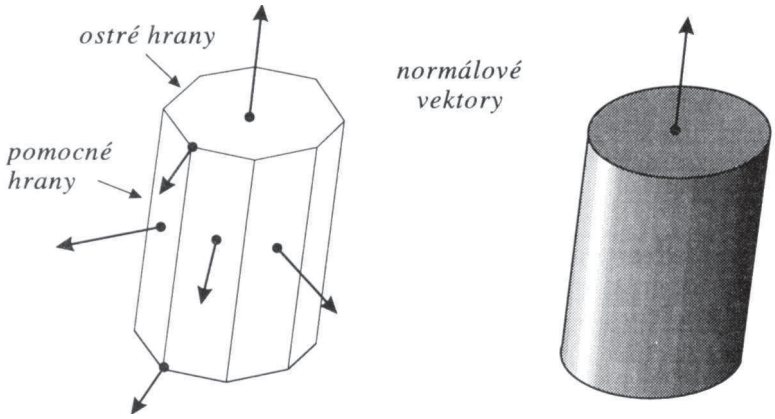
$$I_a = I_1 - (I_1 - I_2) \cdot (y_1 - y_s)/(y_1 - y_2) ,$$

$$I_b = I_1 - (I_1 - I_3) \cdot (y_1 - y_s)/(y_1 - y_3) .$$

Výslednú intenzitu I v bode (x, y_s) na úseku medzi dvoma krajnými bodmi opäť interpolujeme:

$$I_p = I_b - (I_b - I_a) \cdot (x_b - x)/(x_b - x_a) .$$

Predpokladáme, že v dátovej štruktúre máme informácie o ostrých a pomocných hranách tak, ako je to znázornené na obrázku 13.10 pre valec. Na obrázku vidíme, že pre vrchol interpolujúci bočné steny musíme brať do úvahy len priemer normálových vektorov stien, ktoré majú spoločné pomocné hrany.

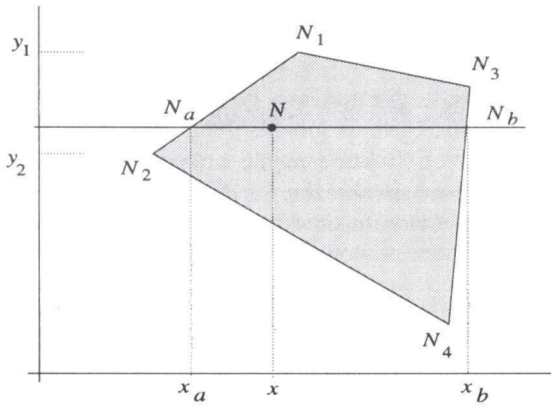


Obr. 13.10 Ostré a pomocné hrany pre valec

13.6.3 Phongovo tieňovanie (Phong Shading)

Phongovo tieňovanie je známe ako tieňovanie pomocou iterpolácie normálových vektorov. Tento postup je podobný Gouraudovmu tieňovaniu, líši sa v tom, že intenzitu osvetlenia bodu počítame interpoláciou normálových vektorov. Podobne ako v predchádzajúcej časti určíme normálový vektor vo vrcholoch mnohostena. Pri realizácii je opäť výhodné použiť algoritmus rozkladu normálových vektorov po riadkoch, pretože môžeme výpočet týchto vektorov vyjadriť krokovým algoritmom. Na obr. 13.11 máme skanovací riadok $y = y_s$, ktorý pretína znázornený štvoruholník v dvoch krajných bodoch hrany d, b. Vo vrcholoch sme určili normálové vektory označené ako N_1, N_2, N_3, N_4. Podľa počtu riadkov medzi dvoma vrcholmi hrany vieme určiť vektorový prírastok, špeciálne dN_d a dN_b na hranách d, b. Týmto výpočet normálových vektorov pri prechode z jedného riadku na druhý zjednodušíme na pripočítanie prírastku:

$$N_a = N_a + dN_d, \qquad N_b = N_b + dN_b.$$

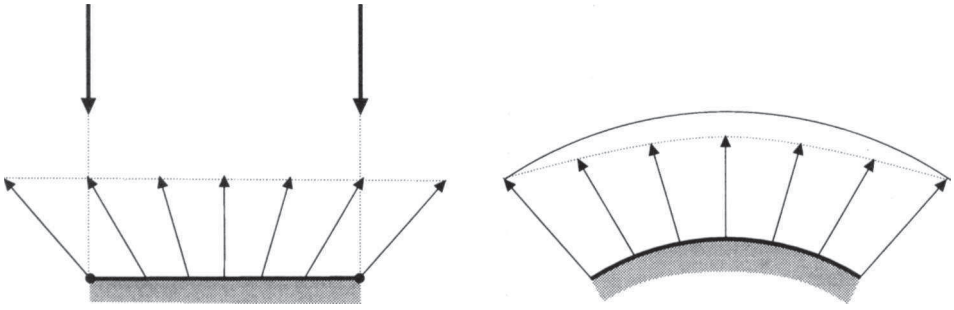


Obr. 13.11 Interpolácia normálových vektorov

Na úsečke medzi dvoma bodmi na riadku vypočítame výsledný normálový vektor interpoláciou súradnice *x:*

$$N = N_a.(x_b - x)/(x_b - x_a) + N_b.(x - x_a)/(x_b - x_a).$$

Výpočet je oproti Gouraudovmu tieňovaniu zložitejší, ale výsledná intenzita je presnejšia. Na jednoduché vysvetlenie si vezmeme príklad z obr. 13.10, ktorý zobrazoval valec. Ak osvetlenie je spredu kolmo na prednú aproximovanú stenu valca, potom vypočítaná intenzita na hranách je rovnaká, ale normálový vektor pozdĺž rozkladového riadku sa mení (obr. 13.12).



Obr. 13.12 Chyba pri Gouraudovej a Phongovej interpolácii

14

Fotorealizmus

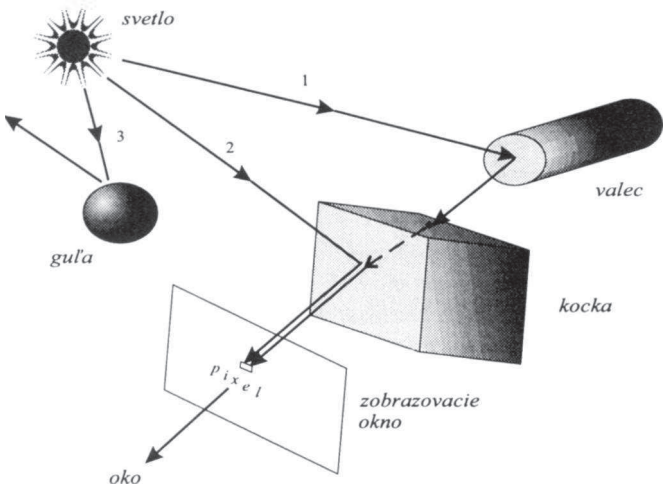
SLEDOVANIE LÚČA

14.1 Úvod

Neoddeliteľnou súčasťou počítačovej grafiky je fotorealistické zobrazovanie scén. Počítačom vytvorené obrazy poskytujú prístup k informáciám, vynímajúcim sa z nášho zrakového vnímania.

Metóda **sledovania lúča** (*ray-tracing*) je založená na vyhľadávaní svetelných lúčov, ktoré po prechode scénou uvidí oko pozorovateľa. V skutočnosti pri počítačovej realizácii obrazovka predstavuje okno, cez ktoré sa pozeráme na scénu. Určenie výslednej intenzity bodu na obrazovke je spojené s vyhľadaním trasy lúčov prechádzajúcich cez tento bod do oka. Pritom môže nastať skladanie lúčov, čiže sčítanie prispievajúcich intenzít, tak ako to vidíme na obr. 14.1.

Sledovanie lúča prvýkrat použil v optike René Descartes v roku 1637 na teoretické vysvetlenie vzniku dúhy [WATT92]. Tradičné použitie sledovania lúča je prevzaté z modelovania šírenia svetelných lúčov cez rôzne prostredia. Whitted [WHIT80] navrhol v roku 1980 prvý všeobecný zobrazovací model, do ktorého integroval odstránenie skrytých plôch, výpočet tieňa, odrazu a lomu lúča.



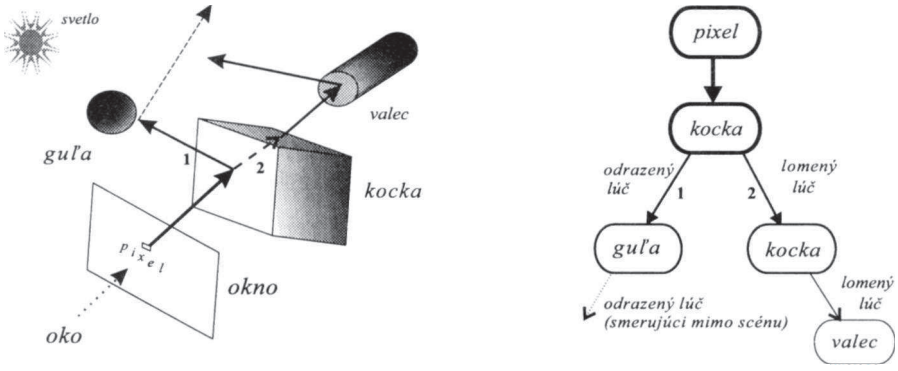
Obr. 14.1 Priamy a nepriamy lúč pre sledovanie lúča

Začneme krátkym popisom hlavnej myšlienky metódy sledovania lúča. Algoritmus odstráni zakryté plochy, výpočíta tieň, zohľadní odraz a lom svetla. Na obr. 14.1 vidíme ako pre sledovaný lúč v scéne počítame intenzitu obrazového bodu, cez ktorý lúč prechádza. Ak máme dve nepriehľadné telesá guľu a valec a čiastočne priehľadnú kocku, môžeme vzhľadom na šírenie svetla odvodiť tri typy lúčov :

1. nepriamy lúč zo svetelného zdroja, ktorý sa bude odrážať od valec a lámať cez kocku k aktuálnemu obrazovému bodu,
2. priamy lúč od svetelného zdroja, ktorý sa bude odrážať priamo od kocky k obrazovému bodu,
3. lúč od svetelného zdroja, ktorý bude vychádzať zo scény.

14.2 Základný rekurzívny algoritmus sledovania lúča

V klasickom algoritme *ray-casting* sledujeme lúče od oka pozorovateľa proti smeru šírenia svetla, pretože sa zaujímame o lúče, ktoré prechadzajú cez okno v priemetni. Sledujeme nekonečne tenký lúč cez obrazový bod okna do priestoru scény (pozri obr. 14.2) len po najbližší objekt. Inak pri algoritme *ray-tracing* sledujeme lúč aj po vyhľadaní najbližšieho objektu. Ak lúč dosiahne najbližší objekt, potom obyčajne vytvoríme dva lúče (**odrazený** a **lomený lúč**, podľa toho, či je objekt priehľadný). Pre nový lúč sledujeme trasu k nasledujúcemu najbližšiemu k objektu a rekurzívne pokračujeme až do danej hĺbky rekurzie. Klasický jednoduchý algoritmus hľadá pre každý lúč prienik s každým objektom v scéne a určí z nich najblížší. Ak lúč nepretne objekt, pridelíme mu intenzitu farby pozadia. Týmto spôsobom vytvárania lúčov od oka cez obrazový bod zabezpečíme zobrazenie iba viditeľných plôch.



Obr. 14.2 Strom pre sledovanie lúča

14.2.1 Osvetľovací model

V algoritme sledovania lúča uskutočnujeme pre každý pixel dva základné kroky:

1. Vytvoríme strom odrazených a lomených lúčov (obr. 14.3)

2. Podľa svetelného modelu vyhodnotíme výslednú intenzitu od listov ku koreňu stromu.

Intenzitu osvetlenia každého lokálneho bodu v strome popíšeme rovnicou (obr. 14.3):

$$I(P) = I_L(P) + k_r.I(P_r) + k_t.I(P_t),$$ kde:

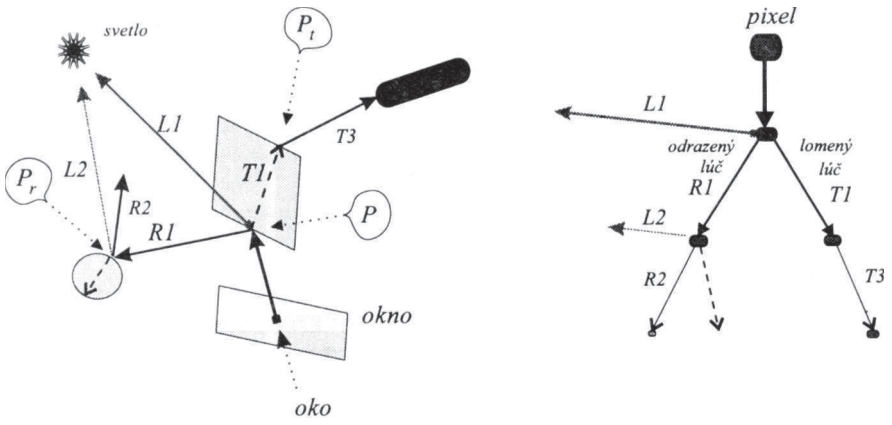
$I_L(P)$ intenzita osvetlenia smerujúceho od svetelného zdroja k bodu P,

$I(P_r)$ intenzita prichádzajúca z bodu P v smere odrazeného lúča od bodu P_r,

$I(P_t)$ intenzita prichádzajúca z bodu P v smere lomeného lúča od bodu P_t,

k_r koeficient odrazenej energie,

k_t koeficient prenesenej energie.



Obr. 14.3 Určenie intenzity odrazeného, lomeného a tieňového lúča

Tieňový lúč je označený na obrázku $L1$ a sleduje, či nie je medzi začiatočným bodom lúča a zdrojom svetla nejaký iný objekt. V prípade, že sa nachádza, potom daný bod je v tieni.

14.2.2 Algoritmus sledovania lúča

Najjednoduchšia varianta algoritmu bude sledovať lúč k priesečníku s najbližším objektom a zistí farbu pre tento bod. Funkciu *ray_trace* rekurzívne voláme na určenie farby pixla pre jeho zobrazenie. Funkcia *Gener_ray* vygeneruje polpriamku danú stredom oka pozorovateľa a pixlom na obrazovke. Funkcia *Intersection* vráti číslo objektu, ktorý je na polpriamke (lúči) najbližšie a bod, v ktorom sa lúč pretne s objektom. Ak neexistuje prienik so žiadnym objektom, potom vráti číslo objektu = 0. Funkcia *Illumination* vráti farbu v zadanom bode na objekte podľa niektorého zobrazovacieho modelu (napr. Phongov model).

V klasickej verzii algoritmu sledovania lúča sa funkcia *ray_trace* volá rekurzívne a ukončovacou podmienkou rekurzie je obvykle buď hĺbka rekurzie alebo ak lúč opustí

scénu. Ak nebudeme uvažovať rekurziu (hĺbka vnorenia je 1), môžeme zapísať hrubú kostru algoritmu sledovania lúča v nasledujúcej forme:

Algoritmus pre sledovanie lúča

```
Program Ray-casting;
function ray_trace (lúč): farba ;
begin
2.1   Intersection (lúč; var bod, objekt);          { zisti prienik lúča s objektom }
2.2   if objekt > 0 then farba := Illumination (lúč, bod, objekt, zdroj_svetla)
2.3   else farba := farba_pozadia;
end;

begin
   for (všetky body_obrazovky) do
   begin
1.    lúč := Gener_ray (pozorovateľ, pixel);        { vytvor lúč od pozorovateľa }
2.    farba := ray_trace (lúč);                     { zisti intenzitu bodu}
3.    plot (pixel, farba);                          { zobraz bod obrazovky intenzitou }
   end
end.
```

Vnasledujúcej časti predpokladáme rekurzívnosť funkcie *ray_trace*. Ak sú v scéne priehľadné objekty, nastáva vetvenie lúčov a v konečnom dôsledku vytvorenie binárneho stromu. Funkcia *lúč* určí lúč smerujúci k zdroju svetla, či je osvetlený. Ak je zatienený, potom vypočítame farbu z ambientného osvetlenia. Funkcia *reflex* vygeneruje odrazený lúč. Funkcia *transp* vygeneruje lomený lúč pre priesvitné telesá. Nasledujúca varianta algoritmu berie do úvahy aj tieň, zrkadlenie a lom svetla:

Rekurzívna funkcia *ray_trace*

```
function ray_trace (lúč) : farba ;
begin
   Intersection (lúč, bod, objekt);
   if objekt > 0 then
   begin
      farba1 := model (lúč, bod, objekt, zdroj_svetla){ farba1 sa určí podľa modelu }
      tieň_lúč := lúč (bod, zdroj_svetla);
      Intersection (tieň_lúč, tieň_bod, tieň_objekt);
      if tieň_objekt ≠ 0 then                    { zatienenie }
         farba1 := ambient;                      { farba1 sa určí ambientným osvetlením }
      zrkad_lúč := reflex (lúč, bod, objekt);
      if zrkad_lúč ≠ 0 then                      { odraz }
         farba2 := ray_trace (zrkad_lúč);        { farba2 zohľadňuje hodnotu k_r }
      lom_lúč := transp (lúč, bod, objekt);
      if lom_lúč ≠ 0 then                        { lom }
         farba3 := ray_trace (lom_lúč);          { farba3 zohľadňuje hodnotu k_t }
      farba := farba1 + farba2 + farba3;
   end
   else farba := farba_pozadia;
end;
```

14.3 Praktická realizácia algoritmu sledovania lúča

Klasický spôsob sledovania lúča sa skončí vtedy, keď lúč nepretne žiadny objekt alebo keď sa dosiahne nastavené **maximum hĺbky rekurzie**. Hall [HALL83] navrhuje používať **adaptívne riadenie hĺbky** rekurzie závisiace na vlastnostiach materiálu, od ktorého sa lúč odráža, alebo cez ktorý lúč prechádza. V krátkosti spomenieme techniky, ktoré môžu ovplyvniť rýchlosť algoritmu.

14.3.1 Všeobecné postupy na zrýchlenie sledovania lúča

Je zrejmé, že algoritmus sledovania lúča je časovo náročný. Z programu vidíme, že vonkajší cyklus uskutočňujeme cez všetky body obrazu a rekurzívnym delením počet lúčov môže narásť až na niekoľko miliónov. Preto je prirodzené zistiť, ktorá časť cyklu je časovo najnáročnejšia. Ukazuje sa, že je to procedúra 2.1 *Intersection*, ktorá vyhľadáva najbližší objekt pre daný lúč. Tieto výpočty môžu trvať relatívne 70 až 90 % celkového času výpočtu.

Problém zrýchlenia môžeme riešiť dvoma prístupmi :

- vhodnou reprezentáciou objektov, aby výpočet priesečníka s lúčom bol čo najrýchlejší,
- zmenšením počtu určení priesečníkov lúča s objektami (určením ohraničujúcich telies alebo delenia scény, pre ktoré je zistenie prieniku s lúčom jednoduchšie).

V nasledujúcej časti si ukážeme rôzne spôsoby vytvárania dodatočných informácií a reprezentácie objektov, napr. delenie priestoru na menšie časti. Kritéria pre určenie, do akej hĺbky máme scénu deliť, sa neustále vyvíjajú [MSHG92]. Jednou z najjednoduchších metód na rýchle určenie toho, či daný lúč pretína objekt, spočíva v používaní ohraničujúcich telies [WATT92]. Za ohraničujúce telesá sa najčastejšie volí guľa alebo kváder, lebo prienik lúča s nimi vieme vypočítať relatívne rýchlo. Kritérium výberu ohraničujúceho telesa závisí nielen na počte objektov v ohraničujúcom telese, ale aj na pravdepodobnosti, s akou sa lúč s ohraničujúcim telesom pretína. Od voľby ohraničujúceho telesa a stratégie závisí aj pamäťová náročnosť dodatočných dátových štruktúr.

14.3.2 Jednoduché zrýchlenie

Weghorst [WEGH84] navrhol zmiešaný algoritmus, v ktorom prienik **primárneho lúča** (ide o prvý lúč od pozorovateľa) vyhodnotíme v predbežnom spracovaní pomocou algoritmu odstraňovania zakrytých plôch. Tento spôsob je častokrát výkonnejší ako všeobecnejší prístup sledovania lúča. Napríklad, môžeme použiť modifikovaný z-bufer algoritmus.

14.3.3 Prispôsobenie hĺbky rekurzie

Intenzita lúčov prechádzajúcich cez scénu sa zoslabuje so vzdialenosťou. Odrazený lúč je tlmený koeficientom odrazu odpovedajúceho povrchu. Lomený lúč sa tlmý koeficientom prenosu zodpovedajúceho objektu, cez ktorý lúč prechádza a tiež od vzdialenosti, ktorú v priehľadnom materiáli prešiel. Tieto vlastnosti zahrnieme do koeficientov materiálu. Ak prínos intenzity daného bodu objektu k výslednému pixlu (daný súčinom

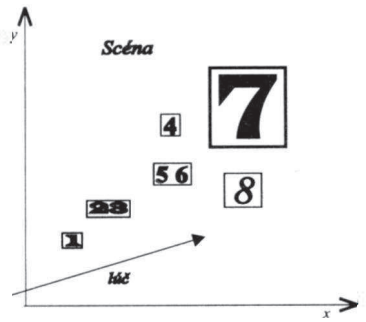
koeficientov odrazu a lomu pozdĺž cesty v strome lúčov) je menší ako prahová hodnota, tak skončíme s ďalším rekurzívnym sledovaním lúčov.

14.3.4 Triedenie objektov podľa obálok

Objekty usporiadáme podľa niektorej zo súradníc. Pri hľadaní najbližšieho objektu využívame to, že po ich usporiadaní nájdeme prvý priesečník skôr. Pre danú scénu vytvoríme zoznam obsahujúci objekty usporiadané podľa súradníc x, y, a z. Ak je lúč orientovaný v smere osi x, potom vyberieme zoznam objektov usporiadaných podľa x. Po nájdení prvého priesečníka sa testovanie ukončí.

Na nasledujúcom obr. 14.4 máme vypísané vytvorené zoznamy objektov usporiadaných podľa smeru súradníc x, $-x$, y a $-y$. Tieto zoznamy sa vytvoria na začiatku algoritmu a nepredstavujú priveľké zdržanie, pričom ich využitie môže podstatne urýchliť samotný beh algoritmu. Podobné predspracovanie sa využíva aj pri iných modifikáciách sledovania lúča. Všimnime si, že môžeme využiť tie vlastnosti, ktoré sme uviedli pri hĺbkovom algoritme.

smer	poradie objektov
$+x$	1, 23, 56, 4, 7, 8
$-x$	7, 8, 56, 4, 23, 1
$+y$	1, 23, 8, 56, 7, 4
$-y$	7, 4, 56, 8, 23, 1



Obr. 14.4 Usporiadaný zoznam objektov

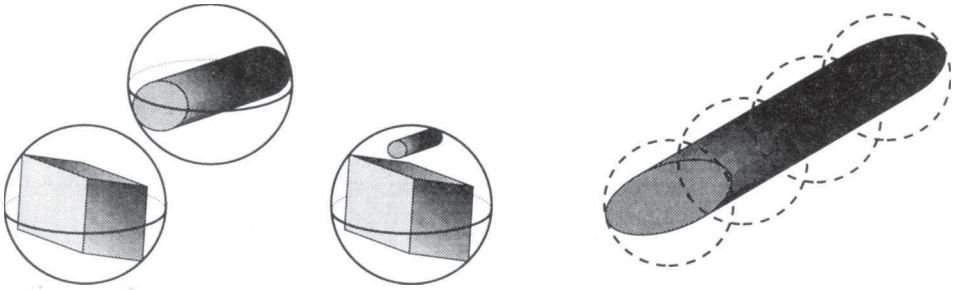
14.3.5 Využitie delenia priestoru a ohraničujúcich telies

Techniky delenia priestoru scény sú založené na vytvorení dodatočnej informácie o rozložení objektov v jednotlivých častiach scény. Existujú dva hlavné prístupy: rovnomerné a adaptívne delenie scény. Pri rovnomernom delení sa priestor scény obvykle postupne delí na kvádre alebo kocky, ktorých veľkosti na danej úrovni sú rovnaké a nezávislé od scény. Pri adaptívnom delení sa priestor scény delí na objemy rôznej veľkosti a tvaru podľa určitých kriterií, ktoré sú zavislé od scény. Kritériom delenia môže byť napríklad počet objektov v danom objeme.

Pre objekty navrhol Clark [CLAR76] ohraničujúce telesá, aby sa urýchlilo testovanie prieniku. Každý objekt alebo skupina objektov v scéne môže byť uzatvorená jednoduchým ohraničujúcim telesom, ako je napr. guľa (obr. 14.5). Sledovanie lúča sa testuje na prienik ohraničujúcich telies. Ak lúč nepretína ohraničujúce teleso, potom nemôže pretínať ani objekt.

Obrázok 14.5 ilustruje vytváranie ohraničujúcich telies. Prístup často volíme podľa danej scéne. Vo všeobecnosti môžeme rozlíšiť 4 prístupy :

- ohraničujúce teleso vytvoríme pre každý objekt,
- ohraničujúce teleso vytvoríme pre skupinu objektov,
- viac ohraničujúcich telies vytvoríme pre jeden objekt,
- vytváranie hierarchie ohraničujúcich telies.



Obr. 14.5 Použitie gule ako ohraničujúceho telesa

Nakoniec uvedieme príklad zistenia prieniku lúča s guľou a kvádrom, ktoré môžeme využiť ako ohraničujúce telesá.

Príklad 1. Test prieniku lúča s guľou. Nech je lúč zadaný začiatočným bodom $\boldsymbol{a} = (a, b, c)$ a vektorom $\boldsymbol{i} = (i, j, k)$, potom parametricky vyjadríme polpriamku:

$$\boldsymbol{x}(t) = \boldsymbol{a} + \boldsymbol{i}.t \qquad \text{pre } t \geq 0.$$

Guľovú plochu so stredom v bode $\boldsymbol{l} = (l, m, n)$ a polomerom r zapíšeme pomocou skalárneho súčinu:

$$(\boldsymbol{x} - \boldsymbol{l}) \,.\, (\boldsymbol{x} - \boldsymbol{l}) = r^2$$

Ak chceme vyjadriť najbližší bod priamky (lúča) k stredu gule, potom pre parameter t (určujúci tento bod) musí platiť, že vzdialenosť $(\boldsymbol{x}(t)-\boldsymbol{l}).(\boldsymbol{x}(t)-\boldsymbol{l})$ je minimálna :

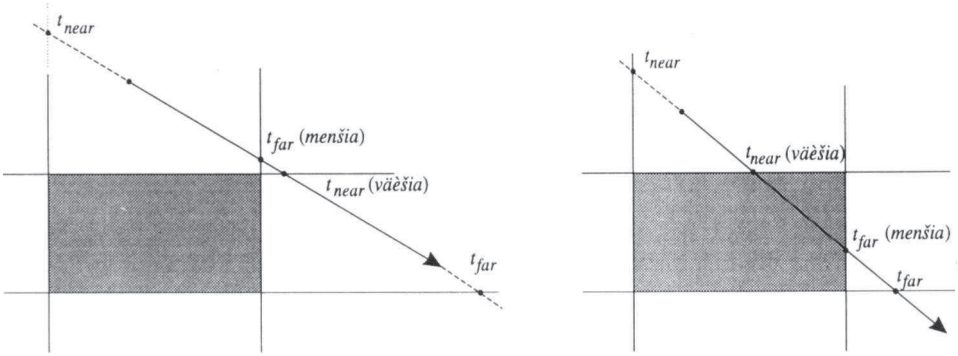
$$\frac{d\,(x(t)-l)^2}{dt} = 2 \cdot (x(t) - l) \cdot x'(t) = 0.$$

Odpovedajúca hodnota parametru t sa dá upraviť:

$$t_0 = \frac{i\cdot(l-a)+j\cdot(m-b)+k\cdot(n-c)}{i^2+j^2+k^2}.$$

Nech $d = x(t_0) - l$ je vzdialenosť bodu $x(t_0)$ od stredu gule. Ak parameter $d > r$, potom guľa nemá prienik s lúčom. Ak $0 \leq d \leq r$, potom guľová plocha sa pretína s lúčom v jednom alebo v dvoch bodoch. Tieto získame dosadením parametrického vyjadrenia polpriamky do rovnice guľovej plochy. Riešime získanú kvadratickú rovnicu pre parameter t. Dosadíme vypočítané hodnoty t do parametrického vyjadrenia polpriamky, a tým určíme súradnice bodov prieniku.

Príklad 2. Všeobecný kváder je vytvorený prienikom telies, určených dvojicami rovnobežných rovín. Ukážeme výpočet prieniku špecialne pre kváder, ktorého steny sú rovnobežné s odpovedajúcimi osami súradného systému. Z obr. 13.6 vidíme, že pre dvojicu rovnobežných rovín môžeme určiť dvojicu parametrov t_{near} a t_{far}, ktoré určujú body prieniku s lúčom ako priamkou. Ak najmešia hodnota z hodnôt t_{far} je menšia ako najväščia hodnota z hodnôt t_{near}, potom lúč nemá prienik s kvádrom, inak má prienik.

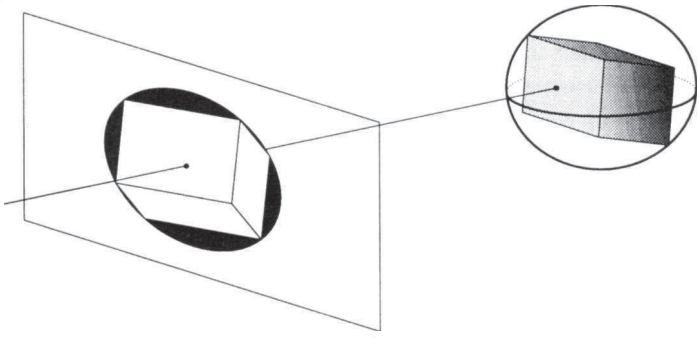


Obr. 13.6 Prienik lúča s kvádrom

14.4 Ohraničujúce telesá

Vrátime sa k problému ohraničujúcich telies. Aby sme sa vyhli zložitému výpočtu prieniku lúča s objektom, budeme najprv testovať lúč na prienik s ohraničujúcim telesom. Takýmto spôsobom vylúčime lúče prechádzajúce mimo objekt. Len v tom prípade, keď lúč ohraničujúce teleso pretína, musíme robiť ešte test na prienik s objektom. Aj keď to znamená teraz o jeden test naviac, celkovo ušetrených testov býva omnoho viac.

Weghorst a kol. [WEGH84] poukázali na to, že zložitosť testovania prieniku nemusí byť vždy najvhodnejšie kritérium pre výber ohraničujúceho telesa. Definovali **nevyužitý priestor** ohraničujúceho telesa v kolmej rovine na lúč. Tento je vytvorený rozdielom priemetu ohraničujúceho telesa a objektu ako je to znázornené na obr. 14.7.



Obr. 14.7 Efektivita ohraničujúceho telesa je funkciou nevyužitého priestoru

Ukázali, že tento nevyužitý priestor je funkciou objektu, ohraničujúceho telesa a smeru lúča. Definovali cenovú funkciu T pre test prieniku lúča s objektom:

$$T = b.B + i.I,$$

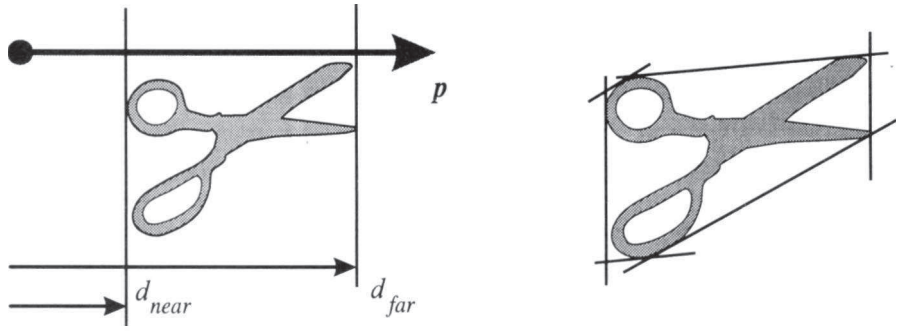
kde b je počet testovaní prieniku lúča s ohraničujúcimi telesami, B je cena testovania tohto prieniku, i je počet, koľkokrát sa testuje objekt na prienik s lúčom (kde $i \leq b$), I je cena testovania objektu na takýto prienik.

Z týchto výsledkov sa ukázalo, že je výhodnejšie vytvárať ohraničujúce teleso guľou ako kvádrom. Preto Kay a Kajiya [KAY86] zaviedli nový typ ohraničujúceho telesa, nazvali ho **zovšeobecnený hranol**, ktorý definovali ako prienik polpriestorov (pozri obr. 14.8). Tento postup zlepšuje riešenie problému nevyužitého priestoru.

Objekty sú uzatvorené do ohraničujúceho telesa vyrobeného pomocou dvojíc rovnobežných rovín. V trojrozmernom priestore potrebujeme minimálne tri nezávislé roviny k vytvoreniu uzatvoreného telesa. Dvojice rovín sú určené normálovým vektorom $\boldsymbol{p}$, ktorý reprezentuje koeficienty roviny:

$$ax + by + cz - d = \mathbf{0},$$

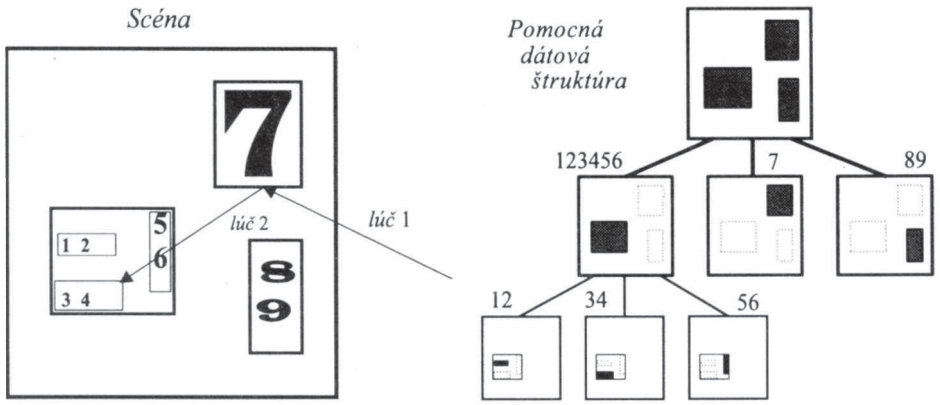
kde $\boldsymbol{p} = (a, b, c)$ a dvoma hodnotami d. Pre každý objekt ohraničený takým telesom je definovaná množina normál ohraničujúcich rovín spolu s hodnotami d_{near} , d_{far} pre každú normálu (pozri obr. 14.8).



Obr. 14.8 Vytvorenie zovšeobecneného kvádra pre objekt

Iný prístup na testovanie prieniku lúča s objektami je vytvorenie štruktúry pre ohraničujúce telesá. **Hierarchickú stromovú štruktúru** pre ohraničujúce telesá zaviedli Rubin a Whitted [RUBI80]. Hlavná myšlienka spočíva v tom, že blízke objekty združíme do podskupín, ktoré ohraničíme telesami na určitej hierarchickej štruktúre (pozri obr. 14.9). Z hľadiska vytvárania hierarchie je výhodnejšie uzatvárať kvádrami.

Lúč postupuje hierarchicky od ohraničujúcich telies až ku koncovým uzlom, kde sa ukončí testovanie na prienik objektov. Týmto spôsobom sa rýchlo priblížime k najbližšiemu objektu a znížime počet testovaní prieniku. Globálne hodnotíme výpočet logaritmicky v závislosti od vytvorenej stromovej štruktúry a od počtu objektov scény.



Obr. 14.9 Stromová štruktúra ohraničujúcich telies pomocou kvádrov

Napríklad na obrázku 14.9 vidíme, že prvý lúč testujeme na prienik s troma ohraničujúcimi telesami a s jedným objektom. Ak sledujeme druhý lúč, tento testujeme s 3 ohraničujúcimi telesami a na nižšej hierarchii opäť s 3 ohraničujúcimi telesami a nakoniec s 3 objektami. Celkovo tieto dva lúče testujeme na prienik s 9 ohraničujúcimi telesami a 4 objektami. Bez hierarchickej štruktúry by sme museli testovať až 18 objektov (2.9 =18) .

Hoci tento spôsob zníži lineárnu zložitosť výpočtov od počtu objektov v scéne, i tak nemusí byť výhodný, pretože každý lúč zostupuje cez hierarchickú štruktúru od koreňa k listom, pričom tento postup nemusí byť najvýhodnejší.

Pre nehierarchické štruktúry bolo najvýhodnejšie vyberať pre ohraničujúce teleso guľu, avšak pomocou nej sa ťažko vytvára hierarchia. Ako sme spomenuli Kay a Kajiya zaviedli nový typ ohraničujúceho telesa, ktoré si ponecháva výhody hierarchického ohraničujúceho telesa.

14.5 Priestorová koherencia

Základná myšlienka priestorovej koherencie je jednoduchá. Scénu rozdelíme na podoblasti. Teraz, skôr ako skontroluje lúč všetky objekty alebo skupinu ohraničených objektov, pokúsime sa odpovedať na otázku: sú v podoblasti, cez ktorú lúč prechádza, nejaké objekty? Ak áno, potom len skupinu objektov z podoblasti testujeme na prienik s lúčom.

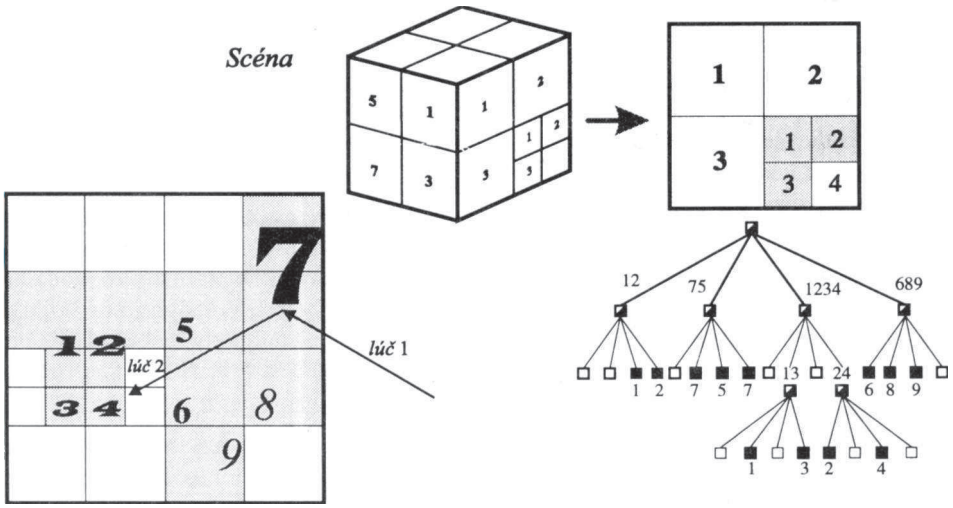
Prístupy *priestorová koherencia, priestorové podrozdelenie* zavedli nezávisle viacerí autori [GLAS84], [KAPL85] a [FUJI86]. Všetky tieto metódy vyžadujú predbežné spracovanie priestoru pomocnou dátovou štruktúrou, ktorá obsahuje informácie o rozdelení objektov v priestore. Lúče potom sledujeme využívajúc túto pomocnú dátovú štruktúru.

14.5.1 Štruktúra oktálneho stromu

Oktálny strom (*Octree*) slúži často ako doplnková hierarchická dátová štruktúra na reprezentáciu objektov v trojrozmernom priestore. Špecifikujeme kvádrové alebo kockové oblasti v priestore objektov. Tieto oblasti sa často nazývajú tiež *voxel-volume element*.

Predpokladajme, že scéna je ohraničená kockou. Najprv kocku rozdelíme na 8 častí. Často používame dva druhy kritérií pre rozdelenie objemovej kocky. Prvý jednoduchší spôsob spočíva v pravidelnom delení a druhý na nepravidelnom delení kocky. Na základe zvoleného kritéria (napr. počet objektov v kocke) budeme rekurzívne pokračovať v delení. Tento postup opakujeme dovtedy, kým nie sme na maximálnej povolenej hĺbke delenia. Ak kocka neobsahuje žiadne teleso, tak ju v strome reprezentujeme listom s hodnotou **prázdna** (*empty*). Ak kocka obsahuje nejaké teleso a ďalej sa nedelí na podkocky, tak reprezentujeme listom s označením **plná** (*full*) so smerníkom na zoznam objektov, ktoré obsahuje. Ak sa kocka delí, tak sa v strome reprezentuje vrcholom, ktorý je rodičom 8 vrcholov (kociek) a označíme ho ako **rozdelená** (*heterogenous*).

Na obr. 14.10 vidíme vytvorenú octree štruktúru, ktorá je kvôli názornosti znázornená v priemete ako quadtree štruktúra. Táto štruktúra nám umožňuje využiť priestorovú súvislosť pre objekty, ktoré sú reprezentované blízkymi vrcholmi v oktálnom strome.

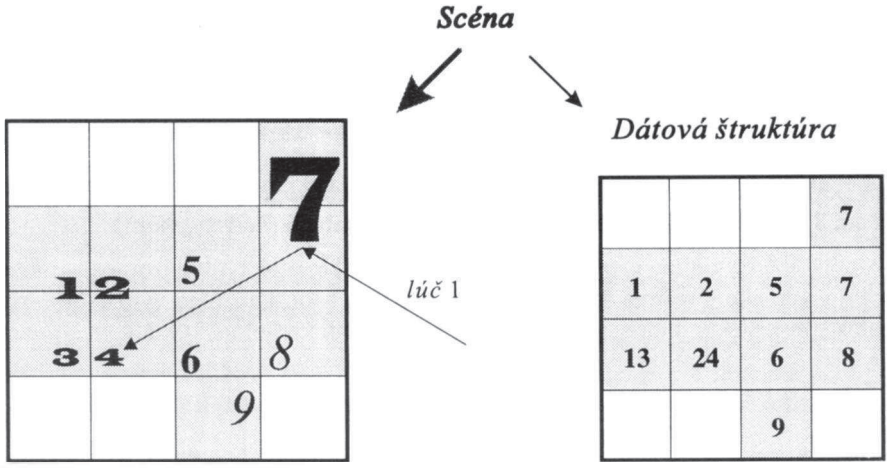


Obr. 14.10 Znázornenie octree (quadtree) hierarchickej dátovej štruktúry

Namiesto toho, aby sme pre sledovaný lúč počítali prienik s každým objektom v scéne, môžeme určiť **trasu lúča** vzhľadom na rozdelenie priestoru. Neprázdne listy na trase testujeme podrobnejšie a prázdnymi listami sa prechádza bez testovania na prienik s objektami. Pre každú oblasť, cez ktorú lúč prechádza, bude malý počet objektov (typicky jeden alebo dva), s ktorými by sa lúč mohol pretínať. Pri realizácii sledovania lúča cez octree štruktúru musíme vyriešiť dve úlohy. Za prvé, musíme nájsť kocku (*voxel*) zodpovedajúci prvku stromu (prípadne naopak), a po druhé, musíme vedieť nájsť *susedný voxel* pre daný smer lúča z *aktuálneho voxla.* Inač povedané hľadáme *voxle* "kocky - body digitálneho priestoru" na trase lúča, ktoré treba pretrasovať.

14.5.2 Uniformné delenie (SEADS)

Pri štruktúre oktálneho stromu sme delenie robili rekurzívne podľa počtu objektov v oblasti. Takto nám vznikajú rôzne veľké oblasti. Fujimoto [FUJI86] zaviedol voxle rovnakej veľkosti a tvaru. Takýto prístup, pomenovaný ako **SEADS** (*Spatially Enumerated Auxiliary Data Structure*), nám generuje oveľa viac voxlov ako pri neuniformnom oktálnom strome, ale na druhej strane trasu lúča určujeme ľahšie. Na obrázku 14.11 vidíme znázornenú pomocnú dátovú štruktúru.



Obr. 14.11 Uniformné delenie a dátová štruktúra SEADS

Štruktúra SEADS umožňuje veľmi rýchly prístup lúča k objektom. Pri prechádzaní octree štruktúrou v smere lúča sa často využíva technika podobná vytváraniu úsečky v 2D rastri, označovaná ako 3DDDA algoritmus (3-Dimensional Digital Differential Analyser), ktorý určuje za sebou nasledujúce kocky pre daný lúč. Prvé algoritmy 3DDDA vznikli pre tieto štruktúry, až neskôr boli zovšeobecnené aj pre všeobecný oktálny strom.

Nevýhodné môže byť použitie SEADS štruktúry (s pripravenými voxlovými podoblasťami) v prípade scény, pre ktorú vytvorená oblasť obsahuje len jeden veľmi malý objekt. V tom prípade budeme sledovať veľký počet lúčov, pričom však väčšina z nich pravdepodobne nebude mať prienik s objektom.

14.5.3 Neuniformné delenie

Pri nepravidelnom delení kocky na základe zvoleného kritéria (napr. počet objektov v kocke) budeme rekurzívne pokračovať v delení. Využitie takého oktálneho stromu je veľmi výhodné pri veľkom počte objektov s rôznymi veľkosťami. Pri použití musíme vedieť na začiatku vytvoriť štruktúru oktálneho stromu. Pri implementácii sa používajú smerníky na zoznamy príslušných objektov oblastí, čo vyžaduje pomerne veľkú pamäť. Prístup do tabuľky smerníkov sa môže robiť hašovaním. K. Sung [SUNG91] navrhuje použiť nasledujúcu tabuľku (*Leveled Voxel Space Index*), aby sa nemuseli odkladať smerníky.

Každá kocka je identifikovaná súradnicami (L, i, j, k), kde L je úroveň vnorenia, i, j, k - sú súradnice kocky z intervalu $\langle 0, 2^L - 1 \rangle$. Jednoducho môžeme prechádzať od rodiča k synom a naopak:

- rodič prvku (L, i, j, k) má súradnice (L-1, i div 2, j div 2, k div 2),

- synovia rodiča (L, i, j, k) majú súradnice (L+1, 2i+a, 2j+b, 2k+c).

Samotné vytvorenie oktálneho stromu vyžaduje geometrické výpočty určujúce, či objekt má prienik s kockou. Nie je však celkom jednoduché zistiť prienik kocky so zložitejšími objektami ako sú napr. kužeľ, toroid, a pod. Tento problém možeme riešiť rôznymi aproximačnými technikami. Jednoducho určíme prienik kocky s guľou a kvádrom. Preto často objekt buď ohraničíme takýmto telesom alebo povrch objektu pokryjeme množinou dostatočne malých kvádrov resp. trojuholníkov, ktoré objekt aproximujú. Ďalším problémom je stanovenie čo najvhodnejšieho kritéria maximálneho počtu objektov v liste stromu, pri ktorom už nedôjde deleniu kocky.

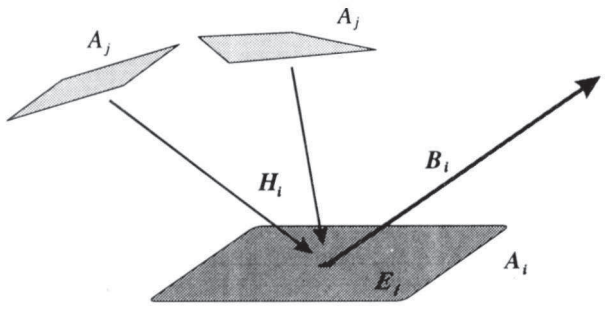
RADIAČNÁ METÓDA

Nedostatkom algoritmu sledovania lúča je, že negeneruje všetky lúče vychádzajúce zo svetelného zdroja, ale naopak sleduje len konečný počet lúčov vychádzajúcich z oka pozorovateľa. Ray-tracing preto nezohľadňuje fyzikálne zákony pre niektoré dôležité vizuálne efekty, napríklad zafarbenie tieňa vplyvom odrazu svetla od iného objektu.

Radiačná metóda (*radiosity method*) je založená na princípoch šírenia svetelnej energie a na energetickej rovnováhe. Jednotkou **intenzity vyžarovania** (*radiosity*) je $W.m^{-2}$. Na rýchle priblíženie si môžeme predstaviť, že nahradíme lokálny Phongov model presnejším globálnym svetelným modelom, zohľadňujúcim len difúzny odraz. Začiatky tejto metódy sa datujú od roku 1984, hlavne prispením autorov M. F. Cohena a D. P. Greenberga [COHE85] a [COHE86]. V tejto časti popíšeme základné vzťahy na výpočet a implementáciu radiačnej metódy.

14.6 Radiačná metóda

Pri realizácii radiačnej metódy musíme prejsť zo spojitého do diskrétneho modelu. Pri popise tejto metódy vychádzame z rozdelenia scény na malé **plôšky** (v origináli *patches*). Scéna pozostáva z plôšok, ktoré pôsobia zároveň ako svetelné zdroje aj ako odrážajúce plochy vytvárajúc uzavretý systém. Predpokladáme, že každá plôška môže ideálne difúzne odrážať alebo emitovať svetlo. Zaujíma nás, ako sa svetelná energia šíri zo zdroja na ostatné plôšky. Je výhodne predpokladať, že všetky plôšky sú druhotné zdroje svetla, t.j. majú priradené hodnoty pre vlastnú vyžiarenú energiu (označovanú ako E - **emitovanú**) a globálne **odrazenú energiu** B.



Obr. 14.12 Energetický vzťah na ploche

Na obr. 14.12 máme znázornenú situáciu pre danú plochu A_i s prichádzajúcou energiou H_i od plošiek A_j. Z dôvodu zachovania energie musí platiť vzťah:

$$B_i = E_i + p_i H_i , \qquad (1)$$

kde p_i je koeficient odrazu.

Prichádzajúca energia H_i závisí od počtu a veľkostí plôch A_j. Označíme $|A_i|$ pre veľkosť obsahu plôšky A_i. Celková energia plôšky A_i s jej obsahom je rovná súčtu príspevkov energií od všetkých viditeľných plôšok A_j s ich vyžarovanou energiou (radiositou) B_j a koeficientom F_{ji} (form-faktorom), čo môžeme zapísať:

$$H_i \cdot |A_i| = \sum_{j=1}^{n} F_{ji} \cdot B_j \cdot |A_j| . \qquad (2)$$

Je zrejmé, že len časť plochy A_j viditeľná z plochy A_i prispieva do prichádzajúcej energie H_i. V tomto prístupe musíme tiež riešiť problém viditeľnosti pre správny výpočet prichádzajúcej energie. Pre úplne zakryté plochy nadobúda **konfiguračný faktor** (*form-faktor*) hodnotu 0, inak nadobúda hodnotu z intervalu $\langle 0, 1 \rangle$. Pre dve plochy platí, že ich konfiguračné faktory sú v nasledujúcej závislosti:

$$F_{ij} \cdot |A_i| = F_{ji} \cdot |A_j| \quad \Rightarrow \quad F_{ij} = F_{ji} \cdot |A_j| / |A_i| .$$

Preto konfiguračný faktor vyjadruje podiel plochy A_j na ožiarení plochy A_i t.j. väčšia plocha A_j ovplyvňuje väčším form-faktorom F_{ij} plochu A_i. Po dosadení konfiguračného faktora do predchádzajúcej rovnice (2) upravíme výpočet H_i :

$$H_i = \sum_{j=1}^{n} F_{ji} \cdot B_j \cdot |A_j| / |A_i| = \sum_{j=1}^{n} F_{ij} \cdot B_j$$

Dosadením do rovnice (1) pre zachovanie energie dostaneme:

$$B_i - p_i \sum_{j=1}^{n} F_{ij} \cdot B_j = E_i .$$

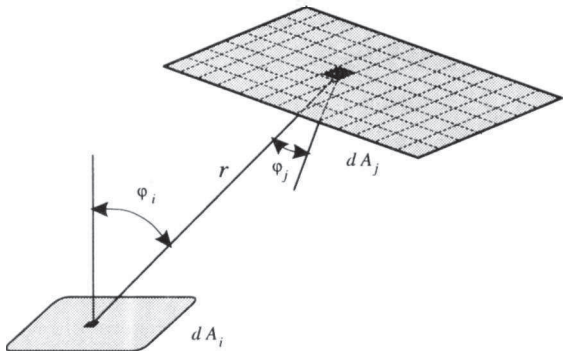
Pre n plôšok dostaneme sústavu n lineárnych rovníc s neznámymi B_i :

$$\begin{pmatrix} 1-p_1F_{11} & -p_1F_{12} & \dots & -p_1F_{1n} \\ -p_2F_{21} & 1-p_2F_{22} & \dots & -p_2F_{2n} \\ \dots & \dots & \dots & \dots \\ -p_nF_{n1} & -p_nF_{n2} & \dots & 1-p_nF_{nn} \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ \dots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \dots \\ E_n \end{pmatrix} \qquad (3)$$

Uvedená sústava lineárnych rovníc má parametre E_i a p_i, ktoré musia byť zadané. Form-faktory F_{ij} musíme vypočítať z geometrie scény. Hodnota emitovanej energie E_i je nenulová pre plochy, ktoré sú sami plošnými svetelnými zdrojmi. Všeobecne hodnoty E_i a p_i sú funkciami závislými od vlnovej dĺžky. Matica je diagonálne dominantná a na jej riešenie môžeme použiť Gauss-Seidelovu metódu. Vypočítané hodnoty intenzity vyžarovania B_i využívame na realistické vykreslenie plôch scény.

14.7 Výpočet form faktorov

Kľúčovým problémom je určenie konfiguračných faktorov F_{ij}, ktoré sú nezávislé od parametrov E_i a p_i. Form-faktory F_{ii} sú definované ako nulové. Nenulové by boli len v prípade, keby plôška žiarila sama na seba. Na obr. 14.13 máme znázornenú situáciu dvoch plôch.



Obr. 14.13 Určenie konfiguračného faktora pre plochy

Nech plôška dA_j žiari na dA_i, potom v diferenciálnom tvare vyjadríme form-faktor:

$$F_{dA_i dA_j} = \frac{\cos\varphi_i \cdot \cos\varphi_j}{\pi \cdot r^2} dA_j .$$

Integrovaním cez plochu A_j dostaneme nasledujúci integrál:

$$F_{dA_i A_j} = \int_{A_j} \frac{\cos\varphi_i \cdot \cos\varphi_j}{\pi \cdot r^2} dA_j \quad , \qquad (4)$$

čo je vlastne energetický príspevok dodaný celou plochou A_j. Pokiaľ teda nie sú plochy medzi sebou zakryté, potom určíme celkový konfiguračný faktor ako dvojný integrál:

$$F_{ij} = F_{A_iA_j} = \frac{1}{|A_i|} \int_{A_i} \int_{A_j} \frac{\cos\varphi_i \cdot \cos\varphi_j}{\pi \cdot r^2} dA_j dA_i.$$

Ak nastane čiastočné zakrytie, potom k podintegrálnej funkcii pribudne nová funkcia *hid*, ktorá nadobúda hodnoty 0 alebo 1 podľa toho, či elementárna plôška dA_j je viditeľná z plôšky dA_i. Z toho dôvodu analytický výpočet je často nemožný a dá sa realizovať len v špeciálnych prípadoch. V prípade, že vzdialenosť r je podstatne väčšia oproti rozmeru plôšok A_j a A_i, potom parametre φ_i, φ_j a r sú skoro konštantné v podintegrálnej funkcii, a preto často považujeme nasledujúci integrál za konštantný:

$$K = \int_{A_j} \frac{\cos\varphi_i \cdot \cos\varphi_j}{\pi \cdot r^2} dA_j,$$

čím sa výpočet výrazne zjednoduší.

Pokiaľ nedochádza k zakrytiu inými plochami medzi plôškami, potom môžeme približne počítať:

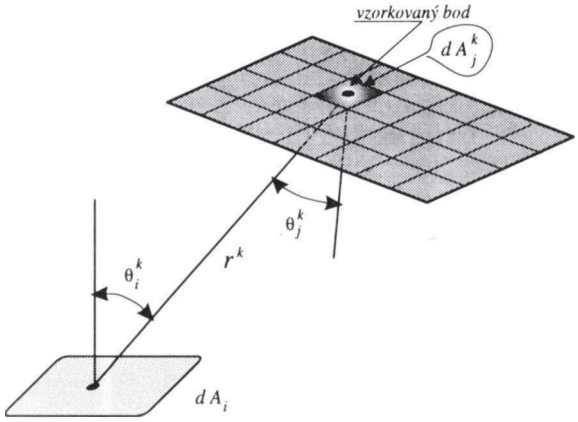
$$F_{ij} = F_{A_iA_j} = \frac{1}{|A_i|} \int_{A_i} K dA_i = K,$$

kde K je spomenutá konštantná funkcia. Tento vonkajší integrál je spriemerovanie konštantnej funkcie, a preto je rovný K.

14.7.1 Výpočet form faktoru pomocou sledovania lúča

Jedným z možných metód riešenia integrálu (4) je metóda Monte Carlo. Náhodne vyberáme body na plochách A_i a A_j, a príslušný integrál nahradíme súčtom form faktorov pre diferenciálne plôšky. Iný spôsob riešenia integrálu spočíva vo **vzorkovaní** (*area sampling*). Predpokladajme, že plochu A_j najprv rozložíme na menšie plôšky a pre každú z nich určíme vzorkovaný bod, ktorý bude zastupovať danú plôšku. Wallace navrhol pre obdĺžnikové plochy rozdelenie po "riadkoch a stĺpcoch" a vybral stred menšieho obdĺžnika za vzorkovaný bod (pozri obr. 14.14). Potom integrál nahradíme súčtom:

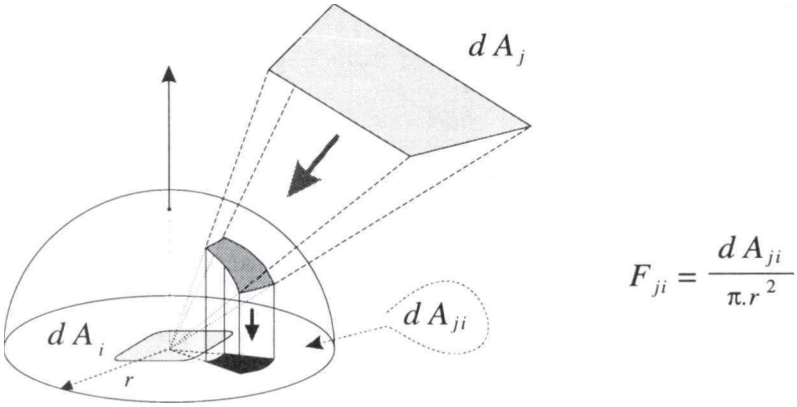
$$F_{dA_iA_j} = \int_{A_i} \frac{\cos\varphi_i \cdot \cos\varphi_j}{\pi \cdot r^2} hid(i,j)\, dA_j \approx \sum_{k=1}^{m} \frac{\cos\theta_i^k \cdot \cos\theta_j^k}{\pi \cdot (r^k)^2} hid\left(dA_i, dA_j^k\right) dA_j^k.$$



Obr. 14.14 Vzorkovanie plochy dA_j na výpočet form-faktoru

14.7.2 Výpočet form faktorov pomocou polgule a polkocky

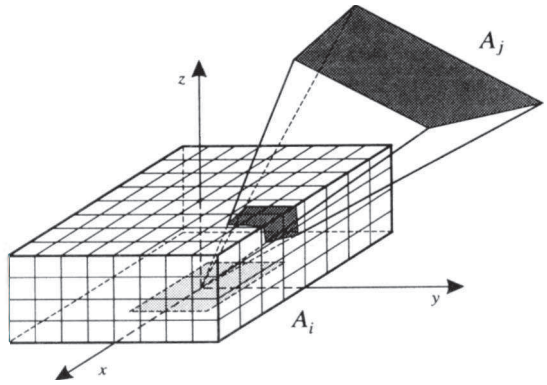
Iný spôsob výpočtu form-faktora je postavený na geometrickej analógii. Zostrojíme nad plôškou dA_i jednotkovú **polguľu** (*hemisphere*), ktorá vlastne vyjadruje priestorový uhol, cez ktorý plôška dA_i prijíma energiu. Priemet plochy dA_j na túto polguľu priamo určuje priestorový uhol, pod ktorým plôška dA_i vidí plochu dA_j. Kolmý priemet tejto časti polgule do základnej roviny dA_i je v priamom vzťahu s form-faktorom. Presnejšie je rovný tejto ploche delenej plochou kruhu πr^2. Obr. 14.15 znázorňuje túto situáciu.



Obr. 14.15 Určenie form-faktoru z priemetu časti polgule do roviny plochy

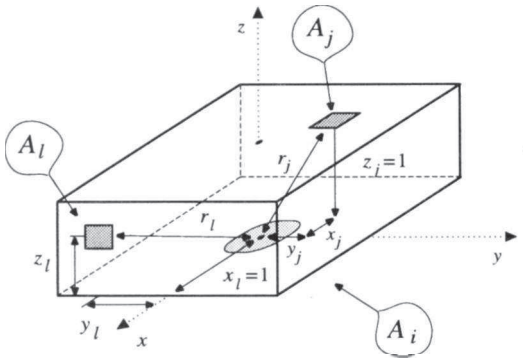
Guľová plocha pre výpočet priemetu nie je z praktického hľadiska vhodná, a preto sa kvôli jednoduchším výpočtom vyberá kocka. Zostrojíme nad plôškou dA_i **polkocku** (*hemicube*), znázornenú na obr. 14.16. Polkocku pokryjeme pravidelnou mriežkou a každý štvorec tohto dláždenia je charakterizovaný určitou hodnotou nazývanou delta faktor ΔF. Hodnota ΔF vyjadruje veľkosť energetického príspevku plochy premietnutej do tohto štvorca. Sčítaním delta faktorov zo štvorcov, na ktoré sa plocha dA_j premieta, získame konfiguračný faktor F_{ij}:

$$F_{ij} = \sum_{k=1}^{r} \Delta F_k .$$



Obr. 14.16 Použitie polkocky pre výpočet form-faktora

Nasledujúci obrázok 14.17 ilustruje výpočet delta faktorov ΔF na bočnej a hornej stene.



$$r_j^2 = x_j^2 + y_j^2 + 1$$

$$\cos\varphi_j = \frac{1}{r_j} \qquad \cos\varphi_i = \cos\varphi_j$$

$$r_l^2 = 1 + y_l^2 + z_l^2$$

$$\cos\varphi_l = \frac{z_l}{r_l} \qquad \cos\varphi_i = \cos\varphi_l$$

Obr. 14.17 Vyjadrenie delta form-faktoru na stenách polkocky

Jednoduchou úpravou dostaneme pre výpočet delta faktorov:

$$\Delta F_j = \frac{\cos\varphi_i \cos\varphi_j}{\pi r^2} = \frac{1}{\pi.\left(x_j^2+y_j^2+1\right)^2}\Delta A_j \text{ - horná stena,}$$

$$\Delta F_l = \frac{\cos\varphi_i \cos\varphi_l}{\pi r^2} = \frac{z_l}{\pi.\left(1+y_l^2+z_l^2\right)^2}\Delta A_l \text{ - bočná stena,}$$

kde ΔA_i, ΔA_j sú plochy dlaždíc. Určenie konfiguračných faktorov vyžaduje riešiť problém viditeľnosti, aby sme určili najbližšiu plochu, ktorá sa premietne na príslušný pixel. Na tieto účely sa využíva modifikovaný algoritmus z-bufer. Pre každú stenu riešime vlastný z-bufer. Scénu transformujeme tak, aby stred plochy ležal v rovine *xy*. Veľkosť pamäti pre z-bufer zodpovedá hustote dláždenia, v praktických úlohách sa pohybuje tento počet 50×50 až 100×100. Do druhého bufra, ktorý sa využíva pre farbu,

budeme ukladať poradové číslo plochy. Form-faktory dostaneme sčítaním delta faktorov všetkých štvorcov dláždenia, ktoré majú rovnaké poradové číslo plochy.

14.8 Implementácia intenzity vyžarovania

Pretože sústava lineárnych rovníc (3) tvorí diagonálne dominantná maticu, môžeme použiť Gauss-Seidelovu metódu. Označme neznámu radiositu ako vektor $\boldsymbol{B}$ a v iteračnom kroku vektor $\boldsymbol{B}^{(k)}$. Riešenie Gauss-Seidelovou iteráciou môžeme zapísať:

$$B_i^{(k+1)} = E_i - \sum_{j=1}^{i-1} K_{ij} \cdot \frac{B_j^{(k+1)}}{K_{ii}} - \sum_{j=i+1}^{n} K_{ij} \cdot \frac{B_j^{(k)}}{K_{ii}}, \qquad (5)$$

kde (K_{ij}) je matica sústavy, t.j. $\boldsymbol{K.B} = \boldsymbol{E}$. Symbolicky môžeme iteráciu zapísať pseudokódom.

14.8.1 Algoritmus Gauss-Seidela

Algoritmus Gauss-Seidel

```
function radiosity;
begin
    for i=1 to n do
        B_i = E(i);                          { začiatok iterácie vektora riešenia }
    while (not converged)
    begin
        for i=1 to n do
            B_i = E_i - Σ_{j=1}^{n} K_ij · B_j/K_ii ;     { iterácia, schematicky podľa (5) }
    end;
output B;
end
```

14.8.2 Progresívne zlepšenie

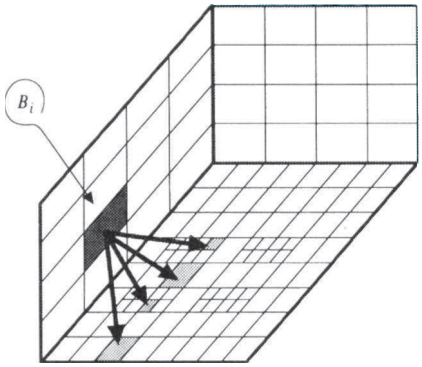
Cohen navrhol **progresívne zlepšenie** (*progressive refinement*). Cieľom tejto metódy je zobraziť výsledok po každej iterácii. Výpočet môžeme pomerne skoro prerušiť a pustiť odznova pri zistení prípadnej chyby. Algoritmus riešenia radiačnej metódy zapíšeme pseudokódom.

Algoritmus Progresive refinement

```
function progresive-refinement_radiosity;
begin
    for i=1 to n do
    begin
        B_i = E_i;                           { začiatok iterácie vektora riešenia }
        dB_i = E_i;
    end
    while (not converged)
    begin
        pick ( i );                          { vyber i tak, aby dB_i *A_i bolo maximálne }
            for j=1 to n do
```

```
            begin
            drad = dB_i*p_j*F_ji ;
            dB_j = dB_j + drad ;
            B_j = B_j + drad ;
            end;
        dB_i = 0;
        plot (scene)                    { zobraz scénu s aktuálnym B_i }
    end;
output B;
end
```

Predchádzajúci algoritmus má nasledujúcu fyzikálnu interpretáciu. Všetky plôšky majú okrem hodnôt intenzity vyžarovania B_i ešte nastavené hodnoty dB_i, ktoré znamenajú časť ešte nevyžiarenej energie. V každej iterácii sa vyberie plôška z najväčšou nevyžiarenou energiou, ktorá sa nechá vyžiariť (pozri obr. 14.18). Dôsledkom tejto akcie j-ta plôška dostane príspevok novej intenzity vyžarovania *drad*, ktorá sa pripočíta k B_j , ale aj k dB_j, pretože táto nová prijatá energia nie je vyžiarená. Prirodzene, že vyžiarená i-ta plôška, nemá čo viac vyžiariť, a preto $dB_i = 0$.



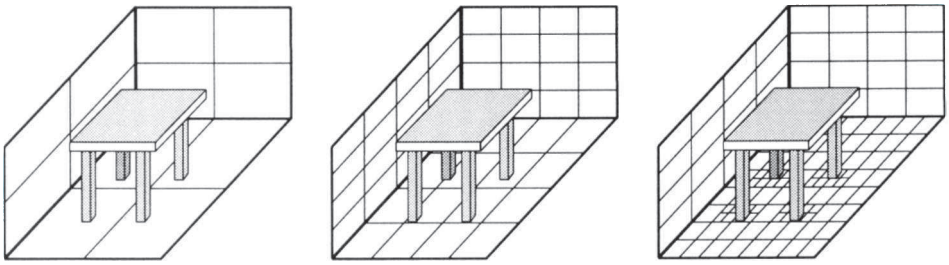
Obr. 14.18 Vyžarovanie z plôšky metódou progresívneho zlepšenia

Lineárnu sústavu (3) pre neznámu *radiositu* B_i riešime v prípade farebných modelov trikrát, t.j. pre každú základnú farbu RGB, pretože parametre p_i a E_i sú závislé od farby. Jedine konfiguračné faktory ostávajú rovnaké, pretože sú nezávislé od farby. Vypočítané hodnoty B_i používame na zobrazenie scény. Farby budú zodpovedať hodnotám intenzity vyžarovania B_i. Pri výpočte form-faktorov často zjednodušíme výpočet konštantými hodnotami na plôške. Preto na záver na zobrazenie využívame Gouraudovu interpoláciu intenzít, čím dosiahneme spojitý prechod farieb medzi susednými plôškami.

14.8.3 Adaptívne delenie plôch

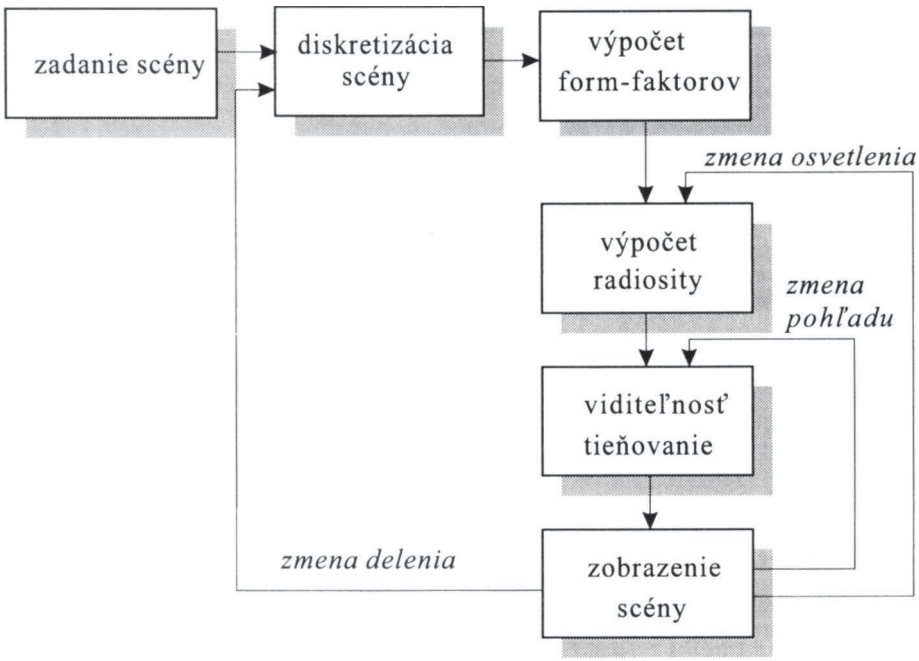
Keď scéna obsahuje veľké rozdiely B_i pre blízke plochy, potom je vhodné urobiť rozdelenie na menšie plochy. Ak zjemníme delenie, potom nemusíme počítať form-faktory pre všetky plochy, pretože celková energia jednej plochy sa iba rozdelí na ich súčet. Z jedného pôvodného riadka vznikne menšia podsústava so známymi

parametrami. V praxi často delíme plochy adaptívne podľa vypočítaných jasových úrovní. Táto situácia je zobrazená na obr. 14.19.



Obr. 14.19 Adaptívne delenie plôch

Pretože konfiguračné faktory vyjadrujú globálny geometrický vzťah prenosu energie medzi plochami, je prirodzené, že sú nezávislé na polohe pozorovateľa v scéne a môžu sa znova použiť pri zmene pohľadu. Pri zmene osvetlenia a farieb objektov sa musíme vrátiť k sústave lineárnych rovníc pre výpočet intenzita vyžarovania. Postup výpočtu pre radiačnú metódu vidíme na nasledujúcom obr. 14.20.



Obr. 14.20 Kroky pre radiačnú metódu

Radiačná metóda vychádza z fyzikálnej podstaty šírenia svetelnej energie, ale na rozdiel od metódy ray-tracing nedokáže zobrazovať zrkadlové javy. Nové prístupy spojenia radiačnej metódy a sledovania lúča sú teraz v štádiu výskumu.

15

Textúry a fraktály

15.1 Mapovanie textúry

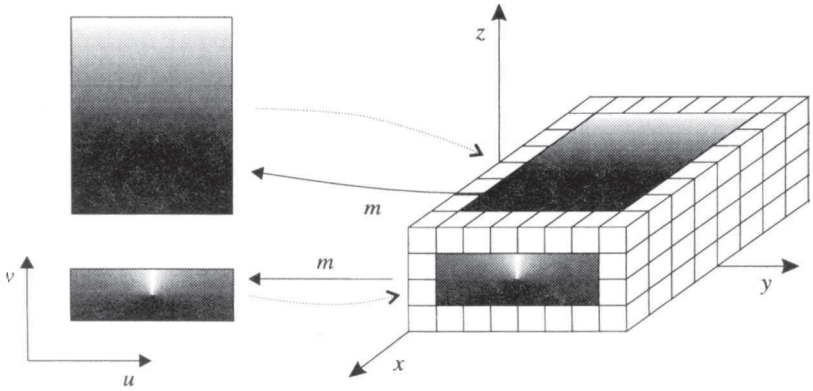
Tri základné vlastnosti objektu pri vizuálnom vnímaní sú tvar, farba a textúra. Objekty v skutočnosti nemajú v každom bode rovnakú farbu a dokonale hladký povrch. Ich opticko-fyzikálne vlastnosti sa líšia od bodu k bodu. Je to štruktúra povrchu materiálu, ktorá je ďalším zdrojom informácií o objekte a budeme ju nazývať **textúrou objektu**. Textúru si môžeme predstaviť ako malé značky na povrchu, ktoré vykazujú určitú štatistickú pravidelnosť. Najprv musíme mať definovaný tvar objektu až potom môžeme vytvárať vzhľad jeho povrchu (*texture mapping*). Na druhej strane je výhodné mať textúru uloženú nezávisle od tvaru objektu.

Z matematického hľadiska definujeme **všeobecnú textúru** ako zobrazenie rovinnej oblasti do modulovaného priestoru, ktorým môže byť priestor farieb alebo úrovní šedej

$t : D_t \rightarrow M$, kde $D_t \subseteq R^2$ a $M \subseteq R\,(R^3)$.

Ak máme zadaný tvar objektu, tak pomocou inverzného mapovania budeme zobrazovať pre každý bod povrchu bod z oblasti textúry

$m : D_m \rightarrow D_t$, kde D_m je oblasť na povrchu objektu.



Obr. 15.1 Mapovanie textúry (texture mapping)

Textúra na povrchu objektu je vlastne zloženie týchto dvoch zobrazení, čiže zobrazenie t je nezávislé na objekte a zodpovedá tapete, ktorú lepíme na teleso. Na obrázku 15.1 vidíme túto situáciu. Pomocou textúry získame nové možnosti pre zobrazovanie a spracovanie objektov.

Zobrazenie t sa často definuje tabuľkou s celočíselnými hodnotami. Môžu to byť obrázky zosnímané skenerom alebo vytvorené grafickým editorom, ktoré ukladajú informáciu v diskrétnej podobe. Inverzné zobrazenie mapuje do oblasti D_t vo všeobecnosti reálnymi hodnotami, preto musíme vedieť interpolovať chýbajúce hodnoty. Najčastejšie sa využíva **bilineárna interpolácia**.

Chceme získať hodnotu $t(x, y)$, preto označme najbližšie hodnoty nasledovne:

$\lfloor x \rfloor$ - zaokrúhlenie smerom dole na celočíselnú hodnotu a

$\lceil x \rceil$ - zaokrúhlenie smerom hore,

$t_{11} = t(\lfloor x \rfloor, \lfloor y \rfloor)$, $t_{12} = t(\lfloor x \rfloor, \lceil y \rceil)$,

$t_{21} = t(\lceil x \rceil, \lfloor y \rfloor)$, $t_{22} = t(\lceil x \rceil, \lceil y \rceil)$.

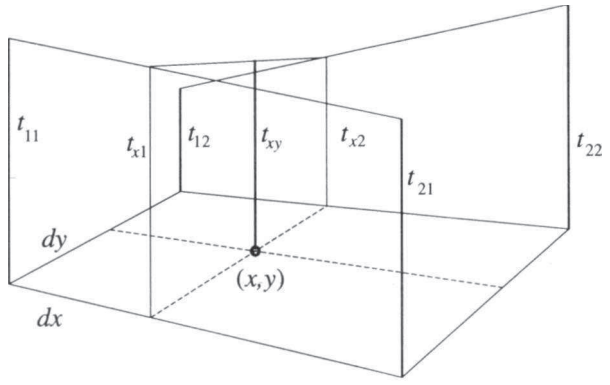
Z obrázku 15.2 vidíme ako vypočítať hodnotu $t(x, y)$ pomocou interpolácie:

$$t_{x1} = t_{11}(1 - dx) + t_{21}(dx), \quad t_{x2} = t_{12}(1 - dx) + t_{22}(dx),$$

$$t(x,y) = t_{x1}(1 - dy) + t_{x2}(dy).$$

Po úprave

$$t(x,y) = t_{11} + (t_{12} - t_{11})dy + [t_{21} - t_{11} + (t_{11} - t_{12} - t_{21} + t_{22})dy]dx$$

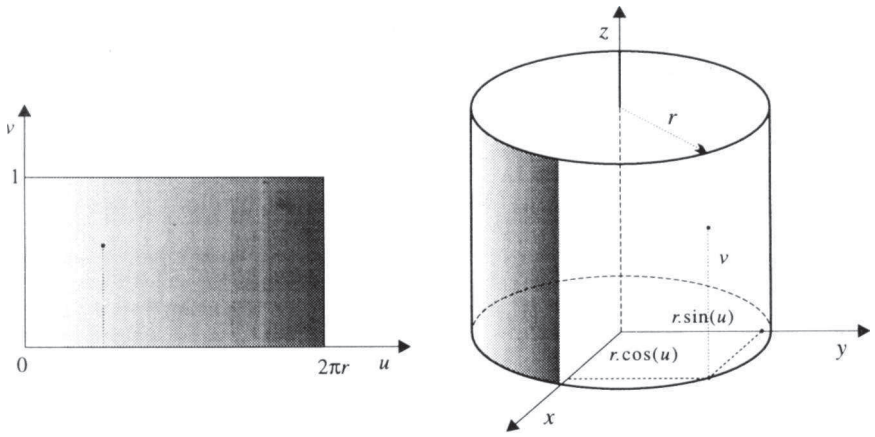


Obr. 15.2 Výpočet interpolovanej hodnoty t(x, y)

Uvedieme príklad zobrazenia textúry na valec. Pretože valec je rotačná plocha, možno ho vyjadriť :

$$x = r\cos(u), \quad y = r\sin(u), \quad z = v$$

Na obr. 15.3 máme zobrazený ohraničený valec pre hodnoty $v \in [0, 1]$.



Obr. 15.3 Inverzné mapovanie na valec

Pre ľubovoľný bod (x, y, z) na valci máme inverzné mapovanie odvodené z predchádzajúcich rovníc:

$$u = \begin{cases} \arccos(x/r), & \text{ak } y \le 0 \\ \arccos(x/r) + \pi, & \text{ak } y > 0 \end{cases}, \qquad v = z.$$

Ak je bod na podstave, tak inverzné mapovanie môžeme jednoducho zadať identitou:

$$u = x, \quad v = y.$$

Valec môžeme pokryť textúrou tak, aby nenastalo skreslenie, t.j. budú sa zachovávať vzdialenosti. Iným problémom je náväznosť troch rôznych mapovaní. Pri guľovej ploche je situácia zložitejšia, pretože neexistuje žiadné inverzné mapovanie, pri ktorom by sa zachovali vzdialenosti. Z toho dôvodu sa zaviedli aj priestorové textúry.

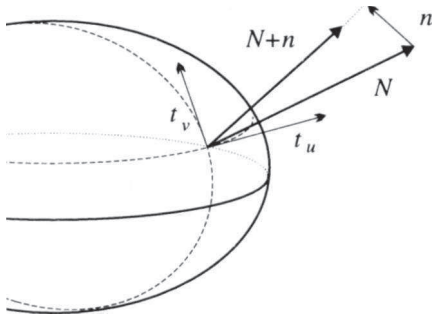
Priestorová textúra priraďuje každému bodu v priestore modulovanú hodnotu:

$$t : D_t \to M, \text{ kde } D_t \subseteq R^3 \text{a } M \subseteq R\ (R^3).$$

Rovinnú textúru sme prirovnali k lepeniu tapety, potom priestorová textúra odpovedá akoby vyrezávaniu z priestorového materiálu napr. dreva alebo mramoru. Pre priestorové textúry ostáva problém ukladania údajov do tabuľky, pretože sú príliš rozsiahle.

15.1.1 Modulovaná veličina

Doposiaľ sme predpokladali, že modulovaný priestor je buď 3D priestor farieb napr. RGB alebo len jednorozmerný priestor pre úrovne intenzity. Oborom hodnôt pre modulovaný priestor nemusí byť len priestor farieb, ale môže byť niektorý parameter osvetľovacieho modelu. Na nasledujúcom príklade si ukážeme takéto využitie.



Obr. 15.4 Modulácia normály

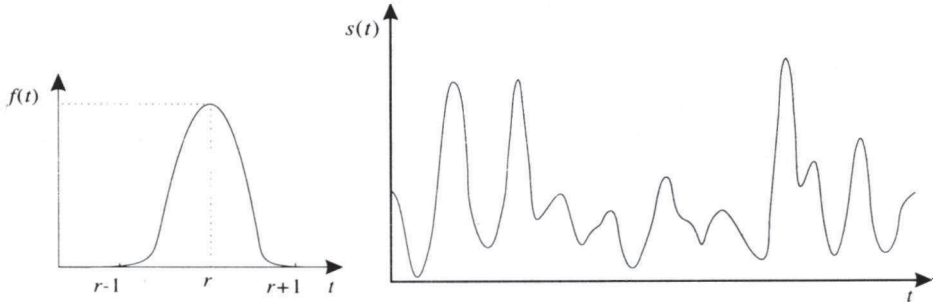
Modulácia normály plochy sa využíva pri zobrazovaní vodnej hladiny. Na obr. 15.4 máme zobrazenú skutočnú normálu ***N*** plochy v bode *P*. Určíme dodatočný vektor ***n***, ktorý bude určovať *hodnotu textúry*. Vo výpočtoch pre zobrazovanie objektu budeme namiesto normály ***N*** brať vektor ***N+n*** resp. s normovanou dĺžkou $(\boldsymbol{N}+\boldsymbol{n})/|\boldsymbol{N}+\boldsymbol{n}|$. Vektor ***n*** môžeme určovať z troch zložiek $\boldsymbol{n}(n_x, n_y, n_z)$ alebo vyjadriť pomocou dotykových vektorov daných parametrizáciou plochy:

$$n = u.t_u + v.t_v,$$

kde t_u, t_v sú dotykové vektory a u, v sú parametre nastavenia modulácie.

Na určenie modulovanej hodnoty sa často využíva **šumová funkcia** (*noise function*). Prirodzené požiadavky pre funkciu sú, aby bola spojitá a mala náhodné hodnoty t.j. vzdialené hodnoty od seba nezávislé. Pre tieto účely definujeme funkciu ako súčet väčšieho počtu náhodne posunutých funkcií. Pre praktické účely sa volí funkcia $f(t) = \exp(-7t^2)$, ktorá má hodnoty mimo intervalu $\langle -1, 1\rangle$ blízke nule a môžeme ich zanedbať, obr. 15.5. Potom pre súčet funkcií máme sumu len cez tie hodnoty r_k, pre ktoré je rozdiel $|r - r_k|<1$:

$$noise(r) = \sum_k v_k \cdot f(r - r_k).$$

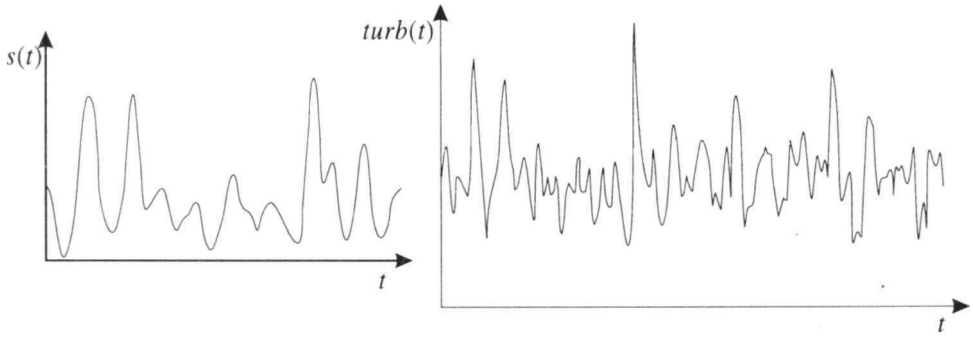


Obr. 15.5 Gaussova funkcia a pomocou nej vytvorená šumová funkcia

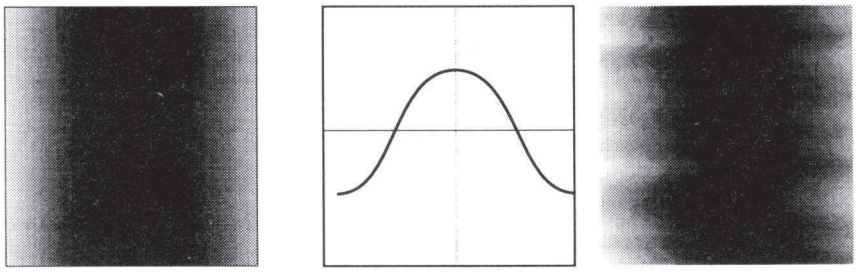
15.1.2 Turbulencia

Ak chceme dosiahnuť vyššie frekvencie, zavedieme podľa Perlina, turbulenciu nasledovným spôsobom (pozri obr. 15.6):

$$turb(x) = \sum_{i=0}^{k} noise(2^i x)/2^i$$



Obr. 15.6 Turbulencia ako šumová funkcia s vyššími frekvenciami



Obr. 15.7 Príklad využitia turbulencie pre textúru

Charakter šumu a turbulencie sa dá prakticky meniť voľbou funkcie *f*. Klasický príklad využitia turbulencie je generovanie textúry pomocou zmeny výberu farieb. Na obr. 15.7 máme neperturbovanú textúru, generovanú napríklad funkciou sin(x). Ak pridáme zmenu vo vertikálnom smere turbulenciou dostaneme nasledovnú zmenu (mramorovú textúru, *marble*):

$$marble(x) = \text{color}\,(\sin(x)),$$

$$marble(x) = \text{color}\,(\sin(x + turb(x))).$$

15.2 Fraktálne generovanie

Pri trojrozmernom generovaní scén sa často využíva fraktálne generovanie, ktoré pôvodne zaviedol B. Mandelbrot [MAND77]. Táto základná myšlienka je založená na poznatkoch generovania máp pobrežia pre rôzne zväčšenia. Prírodné útvary sú často nepravidelné (náhodné), ale na druhej strane sú si do určitej miery aj podobné. Otázkou podobnosti sa zaujímali vedci v šesťdesiatych rokoch a zaviedli pojem samopodobnosti, t.j. keď sa časť ponáša na celok.

Mandelbrot študoval jav samopodobnosti v rôznych súvislostiach, ale predovšetkým ho motivovala skutočnosť, že dĺžka morského pobrežia závisí od toho, v akom zväčšení ju meriame. Ak budeme merať stále vo väčšom zväčšení, potom sa nám bude dĺžka zväčšovať. V limitnom prípade bude dĺžka nekonečná. Preto dĺžka nie je vhodná informácia členitosti pobrežia a ukázal, že vhodným matematickým pojmom pre jej vyjadrenie je Hausdorffova dimenzia.

15.2.1 Hausdorffova dimenzia

Pojem Hausdorffovej dimenzie vysvetlíme na jednoduchom príklade. Ak máme krivku dĺžky jedna, potom ju môžeme merať s úsečkou dĺžky δ (rovnej $1/n$), ktorú pokryjeme počtom $N(\delta)$ (n) úsečiek. Pre dĺžku jednotkovej úsečky platí:

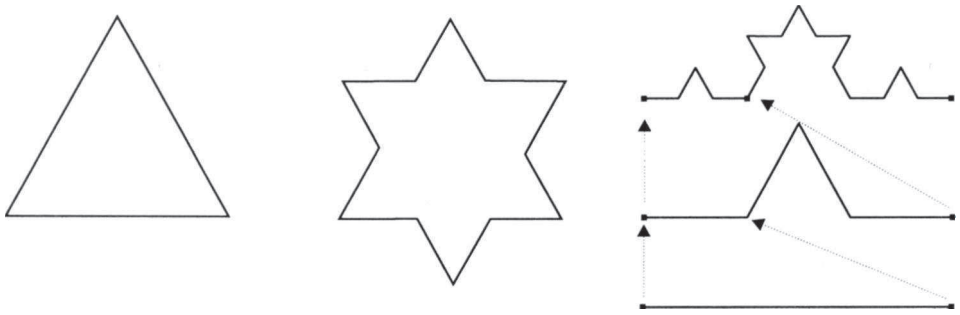
$$L = \lim N(\delta).\delta = 1.$$

Ak poslednú rovnosť zmeníme na nasledujúcu limitu pre δ idúce k nule:

$$\lim N(\delta).\delta^d < \infty,$$

potom má zmysel, len pre $d = 1$.

Tento iteratívny proces sa dá definovať rekurzívne a hodnotou d definovať Hausdorffova dimenzia. Na obrázku 15.8 máme zobrazenú časť Kochovej vločky.



Obr. 15.8 Konštrukcia Kochovej vločky

Hausdorffovu dimenziu Kochovej vločky vypočítame pomocou nasledujúcej limity:

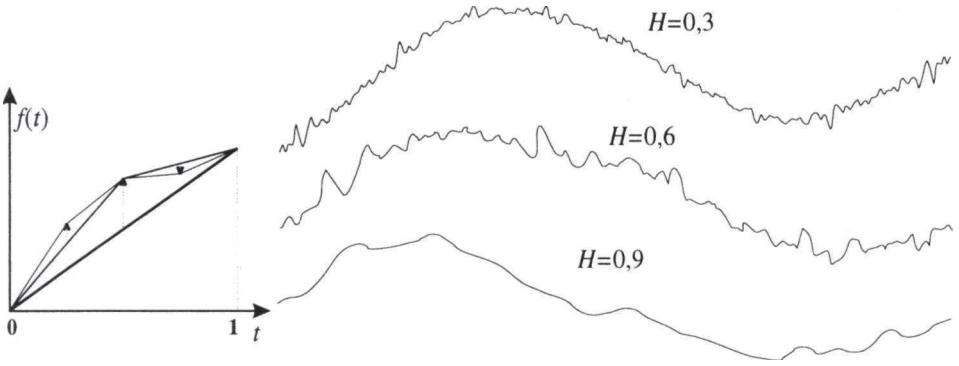
$$\lim N(\delta).\delta^d = \lim 4^i[(1/3)^i]^d < \infty,$$

má zmysel len pre $d = \log 4/\log 3$. Z toho vyplýva, že Hausdorffova dimenzia je reálna hodnota $\log 4/\log 3 = 1.2618$ medzi celými číslami 1 a 2.

Pod **fraktálom** budeme rozumieť taký geometrický útvar, ktorý bude mať Hausdorffovu dimenziu rôznu od topologickej celočíselnej dimenzie. Princip generovania fraktálov spočíva v tom, že sa opakovane používa určitá transformácia na body danej oblasti. Ako príklad uvedieme generovanie Kochovej vločky. Z obrázku 15.8 vidíme, že musíme rozdeliť vždy pôvodný útvar na tri časti a definovať 4 transformácie zložené z posunutia, škálovania a otočenia.

15.2.2 Generovanie náhodnou funkciou

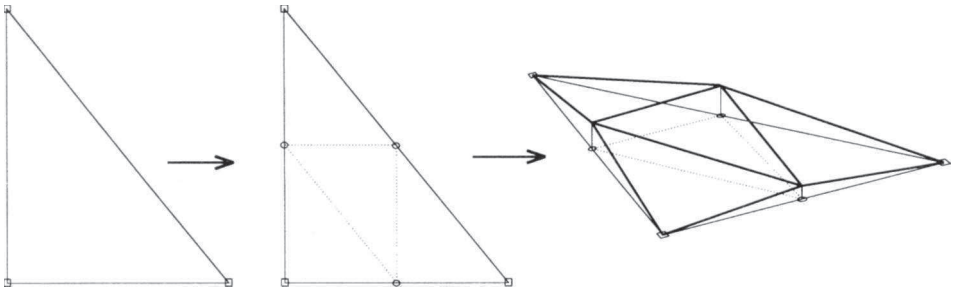
Pre výstižnejšie modelovanie prírodných javov je treba do transformácie zaviesť prvok náhodnosti. Ukážeme veľmi jednoduchý príklad generovania **Brownovho pohybu**, ktorý uviedol ešte Wiener. Algoritmus je založený na metóde náhodnej zmeny stredného bodu úsečky. Označme $f(t)$ polohu bodu v závislosti od parametra t ($0< t <1$). Nech máme dané hodnoty $f(0)$ a $f(1)$. Potom úsečku danú týmito bodmi rozdelíme v strede a aproximujeme lomenou čiarou vychýlenou v strede o náhodnú hodnotu s rozptylom $0.5^2.\sigma^2$, vo všeobecnosti s rozptylom $0.5^i.\sigma^2$.



Obr. 15.9 Krivka generovaná pre Brownov pohyb

Mandelbrot túto metódu zovšeobecnil. Fraktálne krivky popísané predchádzajúcim spôsobom môžeme upraviť tak, aby parametricky určovali rôznu Hausdorffovu dimenziu. Mandelbrot určil stredný bod tak, aby rozptyl v každom kroku bol rovný $(0.5^{2H})^t.\sigma^2$, kde parameter H je z intervalu (0, 1). Potom Hausdorffova dimenzia krivky je 2-H.

Rozšírením metódy na plochy dostaneme **fraktálne povrchy**, ktoré sa často využívajú v počítačovej grafike pri generovaní scén. Najčastejší spôsob generovania využíva techniku delenia uvedenú Fournierom [FOUR82]. Na obrázku 15.10 máme zobrazenú situáciu delenia trojuholníka.



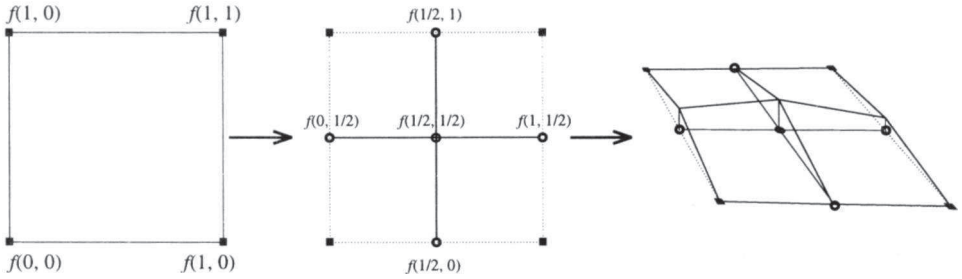
Obr. 15.10 Trojuholníkové delenie

Pre každý nový bod v strede strany generujeme výšku kolmo na rovinu trojuholníka určitým náhodným spôsobom uvedeným predtým pre Brownov pohyb. Týmto spôsobom rekurzívne pokračujeme pre každý nový trojuholník.

Podobne môžeme generovať **stochastické samopodobné fraktály** v priestore pre štvorcovú sieť. Tú inicializujeme podľa obr. 15.11 v rohových bodoch, ktoré označíme

$$f(0,0),\ f(0,1),\ f(1,0),\ f(1,1).$$

Nové body definujeme v stredných bodoch hrán a v strede štvorca. V týchto piatich bodoch $f(0,1/2), f(1/2,0), f(1,1/2), f(1/2,1), f(1/2,1/2)$ môžeme priradiť náhodnú hodnotu vo všetkých piatich bodoch, prípadne len v jednom smere (napr. v bodoch $f(0,1/2)$, $f(1/2,1/2), f(1,1/2)$) alebo len v strednom bode $f(1/2, 1/2)$.



Obr. 15.11 Štvorcové rekurzívne delenie v smere x

Náhodné hodnoty v nových bodoch počítame podľa predpisu:

$$f(0,1/2) = \tfrac{1}{2}(f(0,0)+f(0,1)) + \delta,\quad f(1,1/2) = \tfrac{1}{2}(f(1,0)+f(1,1)) + \delta,$$

$$f(1/2,0) = \tfrac{1}{2}(f(0,0)+f(1,0)) + \delta,\quad f(1/2,1) = \tfrac{1}{2}(f(0,1)+f(1,1)) + \delta$$

pričom pre stredný bod môžeme vybrať jednu z možností:

$$f(1/2,1/2) = 1/4.f_{01} + \delta,\quad \text{kde } f_{01} = f(0,0) + f(0,1) + f(1,0) + f(1,1)$$

$$f(1/2,1/2) = 1/4.f_{1/2} + \delta,\ \text{kde } f_{1/2} = f(0,1/2) + f(1/2,1) + f(1/2,0) + f(1,1/2)$$

alebo

$$f(1/2,1/2) = \tfrac{1}{8}(f_{01} + f_{1/2}) + \delta.$$

Týmto spôsobom je možné generovať obrázky, ktoré sa podobajú hmle alebo mrakom. Fraktály môžeme tiež použiť pre nanášanie textúry na objekty. Iný spôsob generovania fraktálov je cez L-systémy, ktoré sú pomenované podľa Lindenmayera.

15.2.3 L-systémy

L-systémom budeme rozumieť mechanizmus deterministických pravidiel, t.j. gramatikou na generovania objektov. Ukážeme si jednoduchý príklad gramatiky.

Nech $\mathcal{T} = \{a, b, c\}$ je množina terminálových symbolov a $\mathcal{N} = \{A, B\}$ je množina neterminálových symbolov. Definujeme štartovací symbol A a množinu pravidiel:

1. $A \Rightarrow aAb$ 2. $A \Rightarrow aB$ 3. $B \Rightarrow cB$ 4. $B \Rightarrow b$

Pomocou týchto pravidiel môžeme voľbou štartovacieho symbolu generovať nasledujúce reťazce:

ab, *aabb*, *acb*, ...

Teraz dáme do súvisu L-systémy s možnosťou generovať obrázky pomocou korytnačej geometrie (nazývanej LOGO). Ako množinu terminálových symbolov budeme brať:

f ... pohyb korytnačky dopredu

\+ ... otočenie korytnačky doprava

\- ... otočenie korytnačky doľava

(... uloženie stavu zásobníka

) ... vyzdvihnutie stavu zásobníka

Stačí definovať neterminálové symboly a množinu pravidiel a fraktály môžeme generovať ako obrázky, nazývané tiež graftály.

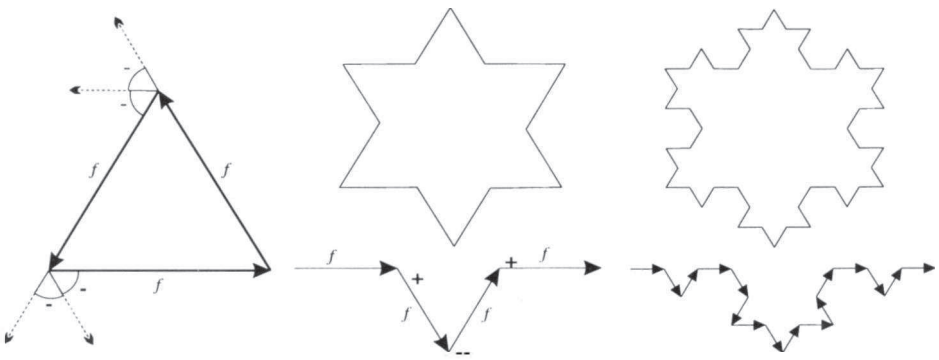
Príklad. Kochovu vločku definujeme pomocou neterminálového symbolu $\mathcal{N} = \{A\}$ a dvoch pravidiel:

1. $A \Rightarrow A+A--A+A$

2. $A \Rightarrow f$

Štartovací symbol je *A--A--A* a otočenie je o 60°. Tieto pravidlá generujú tieto reťazce:

f--f--f, *f+f--f+f--f+f--f+f--f+f--f+f*, ...



Obr. 15.12 Generovanie Kochovej vločky pomocou L-systému

Význam zátvoriek je nasledujúci. Ukážeme si pohyb korytnačky, ktorý je daný reťazcom *f*(+*f*)(-*f*)*f* na nasledujúcom obr. 15.13.

1. Pohyb dopredu

2. Uloženie stavu zásobníka a otočenie doprava

3. Krok dopredu

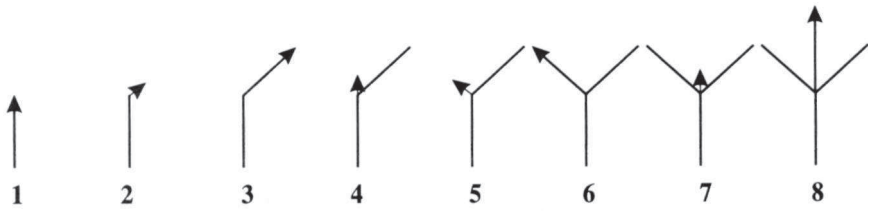
4. Výber stavu zásobníka t.j. návrat do stavu bodu 2

5. Uloženie stavu zásobníka a otočenie doľava

6. Pohyb dopredu

7. Vytiahnutie stavu zásobníka t.j. návrat do stavu z bodu 5

8. Krok dopredu



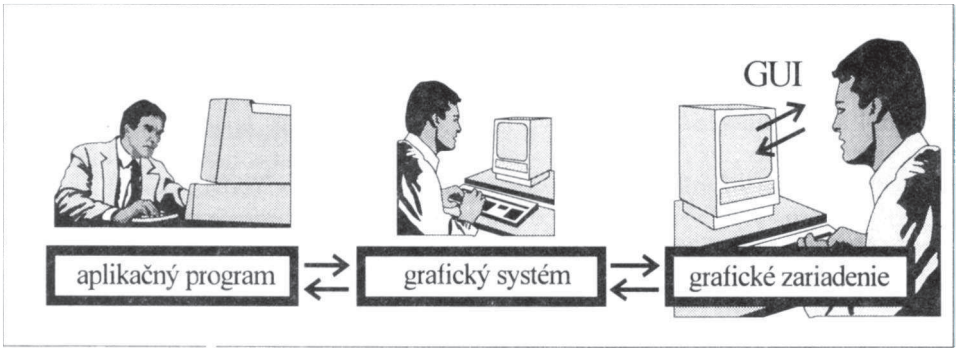
Obr. 15.13 Aplikácia reťazca so zátvorkami

16

Interakcia a oknové systémy

16.1 Grafické užívateľské rozhranie

V tejto kapitole radikálne zmeníme hľadisko i optiku a budeme si všímať najprv niektoré psychologické a potom hlavne informatické aspekty budovania grafického softveru. Namiesto doterajšieho riešenia algoritmických problémov budeme hľadať filozofiu zadávania týchto problémov - odkiaľ a prečo prichádzajú. Vodítkom nám bude **funkčnosť systému** (*functionality*), ktorou daný systém dosahuje cieľ - **efektívne spracovanie grafickej informácie**. Ako sme už spomenuli, na fyzikálne modely, matematické postupy a algoritmy počítačovej grafiky a spracovania obrazu z kapitol 2-15 sa možno pozerať buď ako na stavebné bloky, na myšlienkovom základe ktorých sa neskôr (zdola hore) buduje nejako špecifikovaný softver, alebo ako na prostriedky na dosiahnutie (zhora dole) premyslenej funkčnosti korektne špecifikovaného systému.



Obr. 16.1 Programátor, implementátor GS a operátor

V počítačovej grafike sa podľa funkčnosti striktne rozlišujú tri osoby: **aplikačný programátor, implementátor grafického systému** a **operátor**. Napr. aplikačný programátor rozhoduje o tom, že jeho program vykreslí hviezdičku, toto rozhodnutie však nezávislo od aplikácie vykoná grafický systém tak ako požadovanú akciu naprogramoval implementátor, no hviezdičku uvidí na obrazovke operátor, ktorý môže interaktívne riadiť ďalší beh aplikácie. Operátor sa často nazýva aj užívateľ (*user*). Rozhranie medzi operátorom a pracovnou stanicou nazývame **grafické užívateľské rozhranie, GUI**.

Rozhranie medzi pracovnou stanicou a aplikačným programom nazývame **grafický systém**. V ďalšom rozoberieme najmä funkčnosť týchto dvoch rozhraní.

Prenesením pozornosti na funkčnosť sa teda dostávame na hierarchicky inú úroveň uvažovania o počítačovej grafike a spracovaní obrazu. Nebude nás zaujímať, akým hardverom dosiahneme požadovaný odtieň farby, ani akým algoritmom a dátovou štruktúrou docielime vykreslenie úsečky či korektný vstup polohy kurzora. Bude nás zaujímať ako čo najpresnejšie špecifikovať naše požiadavky na hardver, grafický vstup/výstup a hladkú interakciu. Bude nás zaujímať **správne rozdelenie funkčnosti**, minimalizácia rozhraní a chybovosti, vstupy a výstupy funkcií, ošetrenie chýb - budeme hľadať projekt oknového a neskôr grafického systému. Budeme opakovať myšlienkový postup k funkčnej norme softverového systému, interaktívnej grafickej aplikácie s premysleným užívateľským rozhraním. Naším cieľom bude získať zdôvodnený odborný názor a nadhľad nad mnohými trhovými produktami. Budeme teda abstrahovať od aplikácie a jej modelu na jednej strane i od fyzického zariadenia na strane druhej. V tomto zámere nás bude tlmiť limitovaný rozsah - detailný popis by zabral tisíce strán, pokračovanie vetiev výkladu naznačíme odkazmi na ďalšiu literatúru. Pôjde nám teda o vystihnutie prístupu na riešenie problémov na úrovni funkčnosti grafického rozhrania a grafického systému.

Grafické užívateľské rozhranie (*Graphical User Interface,* GUI) z pohľadu operátora reprezentuje vstupno-výstupné aktivity aplikačného programu (grafický systém operátora nemusí zaujímať a operátor ho často ani nevníma). Motiváciou na výskum GUI je zvýšenie výkonnosti užívateľa. Jeden smer vývoja užívateľských rozhraní sa javí takým perspektívnym, že sa stal na softverovom trhu de facto normou. Sú to GUI na báze 2D **oknových systémov** (*window-management systems,* WMS), ktoré obhospodarujú obrazovku a vstupné zariadenia tak, že niekoľko aplikácií alebo viac záberov jednej aplikácie môže zdieľať obrazovku. GUI možno rozdeliť do troch hlavných častí: **1. okenný systém** (manipulácia s oknami a spracovanie užívateľovho vstupu), **2. zobrazovací model** (*imaging model,* podpora grafického výstupu, napr. kreslenie textov a 2D/3D obrázkov) a **3. aplikačné programové rozhranie a pravidlá štýlu** (*application programming interface, API and style guide*) na podporu prístupu aplikácie k WMS a definovanie **vzhľadu a funkčnosti resp. správania** (*look and feel*) GUI. Ako vidno, okenný systém úzko súvisí s GUI, ale nebolo by správne ich stotožňovať. GUI tiež speje k medzinárodnej normalizácii. Na trhu populárne GUI zahŕňajú Windows, OS Macintosh, OpenLook a OSF/Motif, založené na využití okien, menu, ikon a "ukazovacieho" zariadenia na riadenie polohy kurzora (myš). GUI využíva grafický systém a dopĺňa operačný systém s cieľom zvýšiť komfort a výkon operátora. Vývojové prostriedky resp. nástroje (*toolkit*) a pravidlá štýlu (*style guide*) dávajú metodiku, ako navrhnúť GUI čo najlepšie.

16.1.1 Techniky interakcie

Všeobecný problém komunikácie človek-stroj sa skúmal ešte pred príchodom osobných počítačov a široko dostupnej počítačovej grafiky. Výskumom, najmä psychologickým, sa zistili nasledujúce fakty. Človek vníma omnoho viac informácie, vydrží ju vnímať dlhšie a lepšie jej rozumie, ak je grafická a nie alfanumerická. Medzi jazykom aplikácie a užívateľom by malo byť byť čo najmenej formalizácie, tj. prekladu do iného jazyka. Napr. architekt sa nemá kvôli dialógu so strojom a architektonickej aplikácii učiť

myslieť inak ako architektonicky. Toto zabezpečuje **užívateľský model** (*user model*), v ktorom sa identifikujú objekty, ktoré môže užívateľ ovplyvniť. Vhodné je si pri návrhu uvedomiť, že dialóg sa skladá z dvoch jazykov - **výstupný jazyk** obrázkov na obrazovke a **vstupný jazyk** akcií operátora napr. cez myš a klávesnicu.

Uveďme kvôli úplnosti **metodiku matematického modelovania**, ktorou vzniká aplikačný program. Z každej nasledujúcej etapy sa možno v prípade potreby vrátiť do niektorej predchádzajúcej a spresniť model, algoritmus, program, výstupné údaje.

1. ***Analýza problému***, ktorej výsledkom je exaktný popis modelu

(napr. modelové rovnice).

2. ***Analýza modelu problému***, ktorej výsledkom je algoritmus na riešenie problému.

3. ***Analýza algoritmu***, ktorej výsledkom je program.

4. ***Výpočet zvolených úloh***, ktorých výstupom sú údaje-výsledky.

5. ***Analýza výsledkov***, ktorej syntézou by malo byť riešenie problému.

Metodika zobrazovania má štyri etapy. Autor aplikačného programu rozhodne, čo a ako zobraziť. Nie všetko z modelu sa vždy zobrazuje!

1. ***Určenie objektov*** (vrátane textov, ikon, apod.), ktoré sa budú zobrazovať.

2. ***Geometrický popis*** symbolov, ktoré budú dané objekty reprezentovať.

3. ***Vyjadrenie geometrického popisu*** symbolov v jazyku daného grafického systému.

4. Samotné ***zobrazenie***.

Jazyk dialógu má byť pohotový, úplny a má umožňovať spätnú väzbu. **Metodika návrhu dialógu** (interakcie) má štyri vrstvy:

1. ***pojmový návrh*** (užívateľský aplikačný model), ktoré objekty a pojmy sprístupniť interakcii, napr. pri editovaní súbor, riadok, slovo, znak.

2. ***sémantický návrh*** (činnosť a stav systému), napr. aké akcie možno zadať pre znak: vloženie, zmazanie, prepísanie...

3. ***syntaktický návrh*** (pravidlá dialógu), ktoré akcie sa vylučujú alebo podmieňujú v strome možností, napr. nedovoliť označenie bloku v prázdnom súbore.

4. ***lexikálny návrh*** (ako sformovať symbol jazyka interakcie z hardverových primitívov), napr. ktorá klávesa znamená HELP: F1 alebo ALT H.

Postupom v rámci uvedených metodík sa tvorí interaktívny grafický program na prácu s konkrétnym aplikačným modelom.

Všeobecne prijatým nástrojom interakcie je **menu** a **voľba** (*choice*). Operátorovi sa má zakaždým ponúknuť na voľbu aspoň z dvoch možností - ukončiť prácu resp. vrátiť sa a pokračovať. Jediná možnosť (napr. "Press Enter") nie je voľbou, ale degradáciou človeka na prídavné zariadenie. Strom možností, daný ponukami, nemá mať v jednom vrchole primálo ani priveľa vetiev, ideálny počet možností sa udáva dĺžkou telefónneho čísla (okolo 7), čo súvisí s kapacitou krátkodobej pamäti. Strom možností musí mať (okrem akcelerátorov, popísaných nižšie) práve toľko vetiev, koľko ich ponúka menu na

obrazovke. Aj viditeľne neponúknuté možnosti však musia figurovať v helpe. Foriem menu je niekoľko, napr. roletové či kaskádové.

Vstup čísla voľby je jednou zo **základných interaktívnych úloh** (*basic interaction tasks*). Ďalšími sú **výber časti obrázku** (napr. ikony označujúcej program), **zadanie čísla** (napr. 3.14), **vstup reťazca** (napr. mena súboru) a **vstup polohy** (napr. súradníc umiestenia kurzora v editovanom texte). Rozdiel medzi voľbou a výberom (*pick*) je v tom, že voľbou volíme jednu z (mála) vopred známeho počtu možností (napr. vetiev menu), kým výberom vyberáme z (mnohých) nie vždy vopred známych možností. Nad týmito piatimi vstupnými úlohami možno podľa potreby budovať nadstavby, kombinujúce viacero vstupných úloh, napr. sled viacerých polôh.

Vstup sa skladá z nasledujúcich akcií: program vyzve operátora **výzvou** (*prompt*), aby zadal vstupnú hodnotu. Operátor upraví priebežnú resp. implicitnú hodnotu vstupného zariadenia podľa potreby a odošle ju **spúšťou** (*trigger*). Program na **vstupnú hodnotu** reaguje **echom** (grafickou spätnou väzbou znázorňujúcou vstupnú hodnotu), aby čo najlepšie informoval operátora, ktorý niekedy môže spokojnosť so vstupnou hodnotou vyjadriť **potvrdením** (*acknowledgement*). Špecifikáciu požiadaviek na grafický vstup/výstup budeme v ďalšom podľa potreby štrukturovať, napr. pri vstupe budeme rozlišovať tri rôzne režimy vstupu - vyžiadanie (napr. čísla zvolenej položky menu), vzorkovanie (napr. polohy kurzora) a udalosť (napr. prekrytie okna). Vstupný model podrobne vyložíme v nasledujúcej kapitole.

Odporúča sa, aby aplikačný program počítal s dvojakou vyspelosťou užívateľa v používaní - **nováčik a skúsený operátor**. Pre nováčika treba ponúknuť čo najkomfortnejšie učenie a dostatok pomôcok (kontextovo riadený help, informácie o programe, príklady, demo), kým pre pokročilého treba poskytnúť **akcelerátory** (alebo aj *hotkeys*, napr. ALT-X v Turbo produktoch firmy Borland), ktoré nováčik na obrazovke nevidí, no ktorými pokročilý zrýchľuje dialóg, obchádza vetvenia v strome ponúk.

Program má ihneď po operátorovom vstupe poskytnúť **spätnú väzbu** (*feedback*). Bez spätnej väzby niet dialógu. Podľa rýchlosti vykonávanej činnosti sa rozlišujú tri druhy spätnej väzby - okamžitá, krátkodobá (niekoľko sekúnd) a dlhodobá (viac ako niekoľko sekúnd). Okamžite sa musí napr. stlačenie klávesy so znakom objaviť ako echo vypísaním písmena na obrazovke, kým napr. prekreslenie zložitého obrázku môže trvať dlhšie a pri dlhodobej odozve (napr. výpočet veľkej sústavy rovníc) treba operátora zároveň informovať o priebehu práce i zabávať (presýpacie hodiny, rastúca šípka alebo obdĺžnik, apod.).

Často používaným druhom grafickej spätnej väzby je zvýraznenie polohy na obrazovke **kurzorom**, textovým alebo grafickým. Kurzor v tvare krížika, šípky, ukazováka alebo inej značky sa združuje s polohou, na ktorú ukazujeme niektorým ukazovacím zariadením (myš, guľa (*trackball*)).

Špeciálne typy spätnej väzby sa vyvinuli pre interaktívne kreslenie: sledovanie aktuálnej polohy kurzora tzv. **pružnou čiarou** (*rubber band*), obdĺžnikom, kružnicou, **osovým krížom**. Osvedčilo sa aj zavedenie **obmedzení** (*constraints*), aby sa nekreslilo, kde netreba, napr. zaokrúhľovaním do povolených polôh na daný grid či ťahanie iba zvislej/vodorovnej čiary podľa jednej zo súradníc kurzora. Sem patrí aj zadávanie úsečky, obdĺžnika, kruhu a elipsy dvomi resp. tromi bodmi.

Konzistencia je ďalšou nevyhnutnou požiadavkou GUI. Súlad ponuky a funkčnosti programu, pravdivosť pomocných informácií či bezospornosť kontextu a ponúkanej akcie sú len niekoľkými z mnohých samozrejmostí kvalitnej interakcie. Operátor si má zvyknúť na rovnaký význam farby, postupnosti interaktívnych akcií, jednotného označenia, napr. návratu na predchádzajúcu úroveň stlačením klávesy ESC.

Dôležité je ďalej **nepreťažovať operátora** nadbytočnou informáciou, umožniť mu **ochranu informácie** (*backup, double-buffering*) a **včasné a presné odhalenie chýb** s možnosťou nápravy (*undo, cancel*). Ak však operátorovo rozhodnutie informáciu zničí, treba ho vopred poistiť **nutnosťou potvrdenia deštruktívneho kroku.**

Dobrá interaktívna aplikácia minimalizuje formalizmus a ponúka operátorovi "jeho" pojmy, jeho pracovný stôl, obohatený o využitie pamäti a rýchlosti počítača. Dosiaľ uvedené rozumné vlastnosti ergonomicky dobrej interakcie sa postupne vyvinuli do zaužívanej podoby oknového systému. Na samotný vzhľad a funkčnosť oknového systému (*look and feel*) existujú už tiež podrobne kanonizované odporúčania, napr. **OSF/Motif** či **OPEN LOOK**. Vznik oknovo koncipovaného prototypu GUI sa spája s vývojom užívateľského rozhrania pre objektovo orientovaný jazyk Smalltalk v laboratóriách firmy Xerox - Palo Alto Research Center (PARC) v roku 1975. Všetky základné položky okná, ikony, myš a roletové menu (*Window-Icon-Mouse-Pull-Down*, zaužívaná skratka *WIMP*) sa osvedčili aj v trhových produktoch pre osvedčené platformy, a to nepretržite od uvedenia na trh v roku 1983 (Macintosh, MS Windows, X Window System, NeWS...).

Jednotlivé objekty z aplikácie alebo rozhrania sa často úspešne idealizujú ako piktogramy či **ikony** (napr. program, súbor či odpadový kôš). Niektoré reprezentácie, napr. súbor ako štylizovaný fascikel, sú už všeobecne zaužívané. Vo väčšine aplikácií prevláda požiadavka **WYSIWYG** (*What you see is what you get* - na obrazovke vidno presne to, čo sa vytlačí na tlačiarni). Od komunikácie cez **príkazový jazyk** (*command language*) s príkazmi zadávanými cez textové reťazce sa prechádza k pozičnému výberu z vopred známych možností ukazovacím zariadením a **priamej manipulácii** s ikonami resp. obrazmi modelovaných objektov. Oblasť GUI presahujú hlbšie do vnímania operátora zasahujúce zážitky z fotorealisticky vykreslených scén, animácie a zdanlivého pobytu v priestoroch vytvorených metódami **virtuálnej reality** [AUBL94]. Všeobecne, projekcia operátora do sveta aplikácie opúšťa v maximálnej možnej miere pomalšiu komunikáciu cez druhú signálnu sústavu (cez akýkoľvek formalizmus či konvenciu) a presúva sa v maximálnej možnej miere k rýchlejšiemu a prirodzenejšiemu vnímaniu cez prvú signálnu sústavu, cez zmyslové vnemy. Toto platí aj pre ukazovacie zariadenie, napr. myš, čím sa odformalizovala orientácia a rozhodovanie priamo v obraze.

16.2 Oknový systém

Základnou metaforou oknového systému je **pracovný stôl** (*desktop metaphor*) - na obrazovke sa modelujú a zobrazujú napr. dokumenty, hodiny, diár, písací stroj, kalkulačka... (rôzne aplikácie) a operátor si svoj "pracovný stôl a papiere" podľa potreby upravuje (obvykle myšou a klávesnicou). Každá aplikácia má svoje okno, obdĺžnikovú časť obrazovky s menom, ponukou príkazov na riadenie okna a pracovnou (grafickou) plochou, prípadne hierarchiou podokien. V okne môže byť "jednorozmerný" text, 2D obrázok, trojrozmerná scéna, "4D" film, či vizualizácia mnohorozmerného fyzikálneho

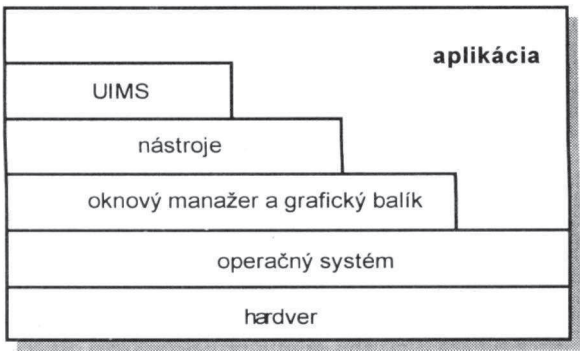
poľa. Dalo by sa povedať, že GUI a WMS vznikli hlavne **kvôli operátorovi** a skúmali **AKO** grafiku čo najlepšie použiť. Výskum GUI a WMS vyvíjal pojmy a metódy v oblasti užívateľského rozhrania (*user interface*), maximalizoval výkonnosť operátora, študovali sa jeho potreby a správanie. Samotná grafika je v GUI a WMS prostriedkom. Iný smer výskumu funkčnosti zas skúmal **ČO** je grafika - **normalizované grafické systémy (NGS)**, o ktorých bude reč neskôr - analyzovali a vyvíjali sa pojmy a metódy v oblasti hardveru, softveru, dát a algoritmov - optimalizovali zložitosť, čas a pamäť grafického pracoviska, minimalizovali rozhrania medzi aplikáciou, grafikou a hardverom. Niektoré oknové systémy obsahujú svoj grafický systém (GS), ktorý ponúka na zariadení nezávislé abstrakcie, kým iné len odovzdávajú volania grafických funkcií príslušnému grafickému softveru a hardveru. Oknový a grafický systém sa teda navzájom nevylučujú, ale možno hovoriť o ich spolupráci. Keďže medzinárodná norma pre WMS zatiaľ chýba, hoci už je jasné, že sa bude veľmi ponášať na X Window, popíšeme tie všeobecné vlastnosti oknového systému, ktoré v norme nepochybne budú.

Oknový systém predovšetkým spravuje prostriedky, takisto ako operačný systém. Rozdiel je v tom, že WMS prideľuje **plochu obrazovky** rôznym programom, ktoré použitie obrazovky požadujú a potom dbá, aby sa tieto pridelené časti obrazovky nemiešali s inými. WMS prideľuje programom aj **vstupné zariadenia** a zabezpečuje, aby sa tok vstupných dát dostal zo zariadenia do fronty udalostí príslušného programu.

Oknový systém sa skladá z dvoch dôležitých častí. Prvou je **window manager** (správca okien), s ktorým komunikuje aplikácia alebo operátor s požiadavkami na vytvorenie okna, zmenu veľkosti okna, pohyb okna, apod. Druhou - funkčne podriadenou - časťou je **window system** (obsluha okna), ktorý aktuálne okno vytvára, posúva, otvára, zatvára, apod. Window manager využíva jeho služby analogicky ako interpreter príkazových riadkov využíva služby jadra operačného systému. Hierarchicky nad oknovým systémom sú aplikačné programy, ktoré sa niekedy nazývajú klientmi, správca okien sa potom spravidla nazýva (špeciálnym) klientom. Oknové systémy sa líšia v ponímaní rozhrania medzi klientmi a serverom, vzťahom k operačnému systému, dokonca počtom správcov okien.

Smerom "hore" od hardveru sa nad oknovým systémom budujú **prostriedky na riadenie interakcie** (*interaction-technique toolkit*), tj. knižnice procedúr, ktoré sprístupňujú interaktívne techniky aplikačným programátorom. Typická zostava týchto prostriedkov (autori X Window System ich nazvali *widgets*) zahŕňa oi. dialógový box, box na výber súboru, help box, list box, message box, pevné menu, pop-up menu ("roletové" či "kaskádové" menu), textový vstup, tlačítka pre voľbu a aplikačné okno - všetky implementované ako okná. Objekty menšieho významu, neimplementované ako okná, napr. šípky v ráme okna, sa nazývajú *gadgets*. Štandardná zostava widgetov a gadgetov dosiaľ neexistuje. Obe slová sú nepreložiteľné (widget znamená čosi ako zariadeníčko, prístrojček a gadget vari caflík, dzindzík).

Nad WMS a jeho knižnicou môže byť ešte **systém správy užívateľského rozhrania** (*user interface management system, UIMS*), ktorý dovoľuje neprogramátorovi interaktívne navrhovať okná, menu, dialógové boxy a postupnosti. Napr. v programe AmiPro si myšou upravíme funkčnosť ponuky a usporiadanie ikon, dokonca si priradíme význam funkčných kľúčov F1, F2...



Obr. 16.2 Úrovne softveru pre GUI

Oproti členeniu aplikácia-grafika-hardver sa oknový systém zaraďuje do bohatšieho kontextu úrovní softveru pre užívateľské rozhranie (*user-interface software*). Aplikačný program má prístup k operačnému systému (cez ktorý siaha na hardver), oknovému systému, grafickému systému, programovým prostriedkom (*toolkit*) a systému správy užívateľského rozhrania (*user interface management system, UIMS*). Interaktívne návrhové prostriedky (WMS, GS, programové prostriedky a UIMS) dovoľujú neprogramátorovi navrhovať okná, menu, dialógové boxy a postupnosti: celok GUI.

16.3 Spracovanie výstupu v oknovom systéme

Oknový systém musí mať na spracovanie výstupu aspoň tieto základné funkcie:

Create Window (name) - vytvorí okno s daným menom

Set Position (xmin, ymin) - nastaví pozíciu aktuálneho okna

Set Size (height, width) - nastaví veľkosť aktuálneho okna

Select Window (name) - urči aktuálne okno

Show Window - zobraz aktuálne okno

Hide Window - skry aktuálne okno

Set Title (name) - nastav meno aktuálneho okna

Get Position (xmin, ymin) - zisti pozíciu aktuálneho okna

Get Size (height, width) - zisti veľkosť aktuálneho okna

Bring To Top - pošli aktuálne okno na vrch všetkých okien

Send To Bottom - pošli aktuálne okno na dno, za všetky okná

Delete Window - zruš aktuálne okno

Tým sme popísali minimálnu funkčnosť oknového systému pri spracovaní výstupu, pričom algoritmické riešenia tejto funkčnej špecifikácie nás na tejto úrovni nezaujímajú, hoci niektoré úvahy môžeme naznačiť. Napr. uvedené funkcie predpokladajú obdĺžnikové okno s menom, rozmermi a pozíciou na obrazovke, súbor takýchto okien s

vyznačením jedného z nich ako aktuálneho a operácie vytvorenia, zrušenia, určenia, premenovania, skrytia a zobrazenia okna. Okrem toho možno nastaviť a zistiť veľkosť a polohu okna. V súbore okien má byť ešte **dno** (*bottom*) a **vrch** (*top*), kam možno aktuálne okno poslať. V tejto špecifikácii "minimálnej" funkčnosti sa však napr. nehovorí, či takýto systém má nejaké menu, alebo ako sa má okno vykresliť a ako určiť aktuálne okno. Dôvodom na tieto "nedopovedané" časti funkčnosti takéhoto oknového systému je, že vykreslenie okna bude riešiť programátor, že určenie aktuálneho okna patrí do vstupu a obsluha menu patrí do grafického systému. Skúmaním horeuvedenej funkčnosti možno dospieť k celému radu ďalších otázok.

Oknový systém prideľuje plochu obrazovky rôznym klientom-programom, ktoré obrazovku vyžadujú, a potom dbá, aby sa ním pridelené časti obrazovky nemiešali s inými. Stratégie tohto alokovania obrazovky sa v rôznych oknových systémoch rôznia, ale dajú sa rozdeliť do troch širokých kategórií. Hlavný rozdiel spočíva v tom, ako sa pridelené okná zobrazujú:

1. Minimálny oknový systém ponecháva zobrazenie na klienta. Klient iba dostane správu, že okno sa vytvorilo, a môže sám spracovať túto udalosť, pričom musí dbať o vykresľovanie a efektívne odkladanie.

2. Pamäťovo bohatší oknový systém odkladá prekryté časti okien, takže klient o vykresľovanie nanovo odkrytých častí nemusí dbať. WMS určí, akú časť okna odložiť. Niektoré systémy (najmä kvôli rolovaniu) odkladajú mapu pixlov väčšiu ako celý displej, aby nestratili aj "odrolované" časti obrazu, čo môže byť pamäťovo neefektívne, no užívateľsky príťažlivé.

3. Oknový systém so **zobrazovacím zoznamom** (*display list*) pre každé okno pri každom prekreslení okna prejde zoznamom a zobrazí len to, čo treba, lebo do zoznamu sa pri vykreslení zapisujú identifikátory objektov, ktoré sa v danom okne zobrazili, a tak má systém priamo informáciu o štruktúre obrázku v okne.

Niektoré WMS používajú **hierarchické okná** - okno obsahuje podokná, ktoré ležia v ich rodičoch. Používajú sa napr. na dialógové boxy. Pri implementácii okien treba riešiť okrem odpamätávania aj niekoľko typických problémov: orezávanie do okna s prekrytými časťami; želá si klient zväčšujúci okno vidieť tú istú časť modelu s väčšou mierkou alebo zachovať mierku a vidieť väčšiu časť? (WMS musí dovoliť obe možnosti); ak sa zmenšuje okno-rodič, čo sa má stať s jeho podoknami? atď.

WMS prideľuje aj ďalšie výstupné prostriedky - položky tabuľky farieb či priamo farby. Ak má systém 8 bitov na pixel a 2 klientov, tak každý môže dostať buď polovicu tabuľky (128 položiek) alebo si WMS tabuľku ponechá a prideľuje im farby podľa ich požiadaviek. Ak však obaja súčasne žiadajú 100% modrej, ale každý z nich si ju predstavuje trocha inak a nemôže zmeniť tabuľku, vzniká neuspokojivý stav. Na túto dilemu opäť niet všeobecne vyhovujúceho riešenia - buď má klient primálo položiek alebo má všetky, ale bez možnosti nastavenia.

16.4 Spracovanie vstupu v oknovom systéme

Oknový systém prideľuje programom vstupné zariadenia a zabezpečuje, aby sa tok vstupných dát dostal zo zariadenia do fronty udalostí príslušného programu-klienta, ktorý prideľuje udalosti rôznym procedúram na spracovanie.

Spracovanie vstupu u klienta (typický pseudokód)

```
repeat
  WaitEvent (timeout, deviceClass, deviceId)
  case deviceClass of
     class1: case deviceId of
                device1: procedureA;
                device2: procedureB;
     class2: case deviceId of... atď.
...
  endcase
until (quit)
```

Typické druhy udalostí vo fronte sú jednak všeobecne grafické ako uvidíme neskôr, ale aj oknové: **KeyPress** (stlačená klávesa), **KeyRelease** (uvoľnená klávesa), **ButtonPress** (stlačená spúšť lokátora napr. myši), **ButtonRelease**, **Motion** (pohol sa kurzor), **LeaveNotify** (kurzor opustil okno), **EnterNotify** (kurzor vstúpil do okna), **WindowExpose** (okno sa čiastočne alebo celkom zobrazilo), **ResizeRequest** (žiadosť o zmenu veľkosti okna), **Timer** (vopred zadaný čas vypršal). Každá z týchto udalostí má **časovú pečiatku** (*timestamp*, čas vzniku), meno okna a ďalšie pre udalosť špecifické informácie, napr. nová veľkosť okna pre ResizeWindow. Samozrejme, fyzické stlačenie klávesy nemusí vo všeobecnosti byť oknovou udalosťou, ale stlačením potvrdený vstupný dátový záznam v danom kontexte už oknovou udalosťou môže byť. WMS musí mať na spracovanie vstupu aspoň nasledujúce procedúry:

EnableEvents (eventList) - dovoľ udalosti podľa ich zoznamu

WaitEvent (timeout, eventType, windowName, eventRecord)

- vezmi novú udalosť

SetInputFocus (window, eventList) - smeruj udalosti podľa zoznamu do okna

Zvláštnym typom udalosti pre WMS, ktoré nezaručujú odloženie okna, je *window-damage*, ktorá indikuje **poškodenie okna**, ktoré treba nanovo vykresliť. Okno sa poškodzuje zmenou veľkosti, rolovaním, odkrytím. Pri zmene kontextu často treba zmeniť tvar kurzora pomocou funkcie **CursorShape (pixmap, x, y)**, ktorá definuje tvar kurzora na pozícii (x, y). Problémom pri spracovaní vstupu je **pripojiť** (*to route*) udalosť k správnemu klientovi. Rieši sa dvoma prístupmi - **staticky** (*real-estate-based routing*) a **dynamicky** (*listener* alebo *click-to-type routing*). Prvý prístup zistí, v ktorom okne je kurzor, a všetky udalosti smeruje klientovi, ktorý toto okno vytvoril. Druhý prístup očakáva, že klient sám označí, aký typ udalosti pripojiť ktorému klientovi (nemusí to byť on sám).

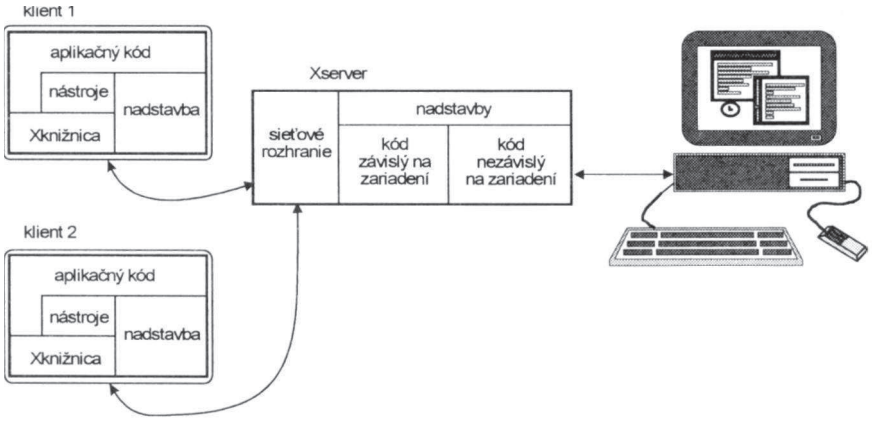
16.5 Príklady oknových systémov

16.5.1 MS Windows

MS Windows uvádzame ako príklad široko dostupného oknového systému. Už pri prvom operátorskom kontakte vidno prakticky všetky horeuvedené vlastnosti oknového systému: ovládanie prevažne myšou cez menu s ikonami, dialógové boxy na zadávanie užívateľských vstupov pre danú aplikáciu (reťazcom, voľbou zo zoznamu ponúk či svetelných tlačítok (*light buttons*)), možnosti škálovať okná, presúvať okná a ikony, zmenšiť okná do ikon (*shrinking windows to icons*), obnoviť veľkosť okna alebo ikony (*restore*). Oknový systém je súčasťou (užívateľským rozhraním) operačného systému.

16.5.2 X Window System

Oknový systém X (správne *X window system* alebo *X*, nesprávne *X windows*) verzia 11 (X11) je čoraz populárnejší "štandardný" oknový systém pre pracovné stanice s rastrovými displejmi. Verzia X11 Release 5 predstavuje de facto normu pre unixovské stanice.



Obr. 16.3 Architektúra X window system

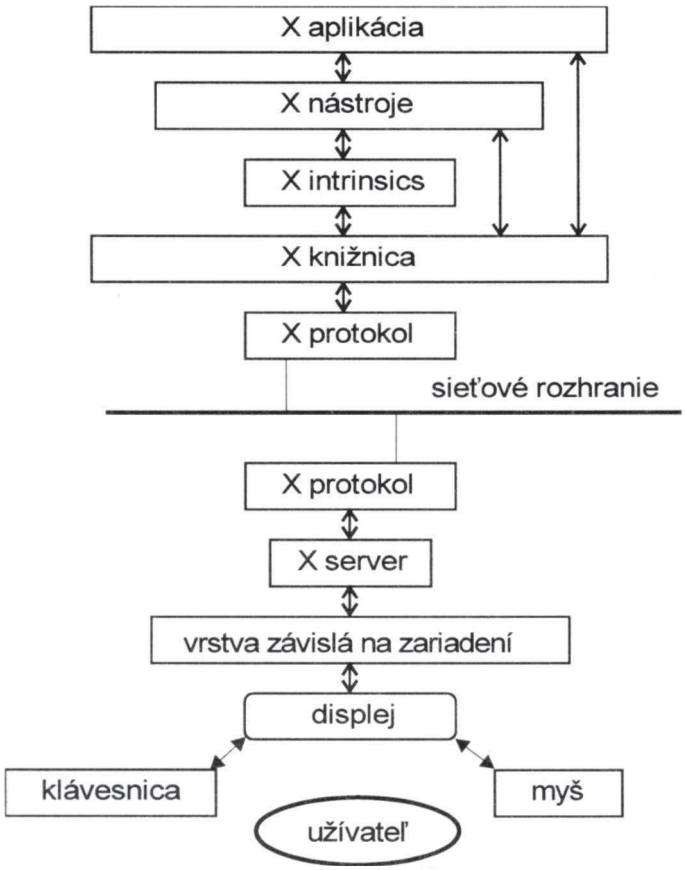
Vyvinul ho od r. 1984 malý tím v projekte Athena and Argus na Massachusets Institute of Technology (MIT), [SCGE86]. Uviedol do života niekoľko nových pojmov a za svoju popularitu vďačí aj distribúcii zdrojového kódu a vysokej prenositeľnosti. X poskytuje správu displeja a grafiku pre 2D aplikácie. Ďalší rozvoj X zabezpečuje od roku 1988 MIT spolu so skupinou špičkových výrobcov pracovných staníc X Consortium (členmi sú Apple, AT&T, DEC, HP/Apollo, Sun, IBM, Televideo a Tektronix). X nepredstavuje kompletný GUI, ale je bázou pre rozšírené GUI silných firiem: CXI (Hewlett Packard), DECwindows, OPEN LOOK (AT&T), resp. OSF/MOTIF (Open Software Foundation integrovala v tejto norme viacero GUI a podľa tejto normy potom vznikli HP-VUE, SCO, NeXTstep, IRIX na rôznych hardverových platformách a

samozrejme nad X). Neskôr na MIT ustanovili skupinu X3D, ktorá čoskoro [ROST89] vydala špecifikácie PEX (PHIGS Extension to X). PEX rozširuje X všeobecne pre 3D grafiku a konkrétne pre PHIGS a PHIGS PLUS. PEX je živou témou na popredných konferenciách o počítačovej grafike, napr. SIGGRAPH'94 venoval značnú časť vzdelávacieho programu (*tutorials*) programovaniu v PEX.

X poskytuje na vysokej úrovni veľmi výkonné rozhranie na riadenie rastrového displeja. Aplikácia komunikuje so serverom v pojmoch ako text, čiary a oblúky s atribútmi ako font, farba, šírka čiary. Server dbá, aby vykresľovanie v nezávislých oknách nekolidovalo a ošetruje alokovanie všetkých zdieľaných zdrojov (napr. plocha obrazovky, mapy farieb, fonty). Podobne server zabezpečuje spracovanie vstupov a smeruje každú vstupnú udalosť príslušnej aplikácii a oknu. Architektúra systému je založená na modeli **klient-server**. Server riadi displej, kým klient (aplikácia) žiada server o vykonanie operácií s grafikou a oknami. X je maximálne nezávislý na zariadení, čiže aplikačný programátor nemusí študovať rozdiely v hardveri, konkrétne hardverové detaily nemusí poznať. Významnou výhodou X oproti iným WMS je sieťová transparentnosť: aplikácia X nemusí bežať na tom počítači, kde je displej pripojený fyzicky. (Navyše, na jednom displeji môžu bežať aplikácie z rôznych počítačov s rôznymi operačnými systémami.) Komunikačný protokol X minimalizuje vplyv oneskorenia dát pri prenose, ale nemôže ho eliminovať úplne.

Vývoj X11 podliehal nasledujúcim požiadavkám a princípom: prenositeľnosť (vlastnosti unikátne na konkrétnom displeji sa buď eliminovali alebo zovšeobecňovali), nezávislosť aplikácie na zariadení (aplikácia pracuje v podstate rovnako na všetkých displejoch), sieťová transparentnosť, viacúlohový režim, podpora všetkých štýlov interakcie, vysoká výkonnosť, prekrývajúce sa okná, hierarchia okien a ich meniteľná veľkosť, kvalita textu i obrázkov. X obsahuje aj metódu na rozšírenie servera, napr. o 3D grafiku, video, zvuk alebo pridanie takých výstupných typov ako Bézierove krivky. Základné prvky a pojmy X zahŕňajú: **model klient-server**, **displej** (*display*) a **obrazovka** (*screen*), **okná**, **správa okien**, **grafika**. Internú prácu X zabezpečujú: dátové typy, prostriedky a **protokol X**, ktorý je základným jazykom (analógiou strojového kódu) pre X a programy v ňom sa v podstate nikdy nepíšu priamo. Analógiou assembleru je knižnica Xlib a analógiou jazyka vysokej úrovne je X Toolkit (nástroje).

Programovanie X pomocou 2D celočíselnej grafickej knižnice **Xlib** pozostáva zo zriadenia spojenia klient-server, nastavenia okien, odpovedí na udalosti, kreslenia obrázkov a komunikácie so správcom okien. Vyššia úroveň programovania využíva **XToolkit**, ktorý sa skladá z **Xt Intrinsics** (podporné funkcie) a **Widget set** (špecializované objekty-okná). Namiesto X Toolkitu však možno na tejto úrovni programovania GUI použiť aj inú nadstavbu, napr. čoraz rozšírenejší **OSF/Motif**, ktorý sa skladá z troch častí: Toolkit s widgetmi, Window Manager a User & System Management. Widgety sú aktívne objekty (OSF/Motif ich ponúka asi 40 tried, napr. aj posuvný potenciometer (*scrollbar*)), neoknové objekty - gadgety (napr. šípka hore na scrollbare). Programátor v jazyku UIL (User Interface Language) definuje statické hierarchie widgetov, aby komfortne a pritom efektívne navrhol čo najlepší GUI.



Obr. 16.4 Štruktúra X aplikácie

16.6 Perspektívy

Oknové systémy a GUI sa vyvíjajú ďalej. Samotné programovanie GUI a grafiky sa v súčasnosti obohacuje o metodiku objektovo orientovaného programovania. Iným smerom sa programovanie grafiky rozvíja do jazykov popisu kompozície stránok, napr. **PostScript** (produkt firmy Adobe). V uvedených smeroch (nadstavby, objekty, popis stránok) možno uvidieť tendenciu ďalej odformalizovať prácu s grafikou a interakciou. Vývoj X k 3D grafike pod názvom PEX naznačíme v kapitole 19. Oknový systém sa v dohľadnom čase stane tiež normou ISO, lebo problém neprenositeľnosti softveru sa už prejavil aj medzi rôznymi oknovými systémami. Ako neraz, v normalizácii pred svetom vedú Američania (American National Standardization Institute, ANSI - komisia X3H3). Všeobecne prijatá norma pre GUI bude súvisieť s dohodou **COSE** (*Common Operating System Environment*), ktorej GUI časťou bude **CDE** (*Common Desktop Environment*), kde sa definuje "look and feel", ktorý má nahradiť OSF/Motif a OPEN LOOK, [LIMP94].

17

Normalizované grafické systémy

17.1 Normalizácia grafických systémov

Kým oknový systém je súčasťou GUI, grafický systém poskytuje funkčnosť na konštrukciu okien, kreslenie do okna a spracovanie interakcie. V tejto kapitole sa budeme najmä z hľadiska aplikačného programátora zaoberať štruktúrou a funkčnosťou tých grafických systémov, nazývaných **normalizované grafické systémy** (NGS), ktoré predstavujú jeden zo smerov rozvoja počítačovej grafiky a sú výsledkom medzinárodného úsilia o vytvorenie jazykov na presné myslenie o grafike. Grafický systém možno vnímať ako rozhranie medzi aplikačným programom a grafickým zariadením. Aplikačný program objekt modeluje, grafický systém ho zobrazuje. Grafický systém nezávisí ani na aplikácii ani na konkrétnom zariadení. NGS vznikli predovšetkým kvôli prenositeľnosti grafického softveru a ich vplyv na softverovú výrobu je a bude dlhodobý. Prvým NGS bola norma GKS a trhovo najúspešnejšou z noriem sa zdá byť PHIGS, ktorý už firma DEC dodáva dokonca v hardverovej implementácii k procesoru Alfa. Formálne prijaté normy ISO bývajú trhovo úspešné a neúspešné. Napr. veľmi úspešnou normou je rozhranie RS-232-C, ale ANSI normu návrhu klávesnice celkom prekryl na trhu návrhu firmy IBM pre PC/XT a PC/AT.

Medzinárodné normy prijíma Medzinárodná organizácia pre štandardizáciu (*International Organization for Standardization,* ISO), existujú však aj európske a národné normy (v USA ANSI, v Nemecku DIN). Prvým NGS bol **Graphical Kernel System** (GKS, 1985), nasledovali **Computer Graphics Metafile** (CGM, 1987, 1992 - norma nie NGS ale spôsobu ukladania grafických dát, čo s NGS úzko súvisí), **Graphical Kernel System for Three Dimensions** (GKS-3D, 1987), **Programmer's Hierarchical Interactive Graphics System** (PHIGS, 1989), **Computer Graphics Interface** (CGI, 1992) a **PHIGS Plus Lumiérre und Shading** (PHIGS PLUS, PHIGS+, 1992). Pripravuje sa **Windows Management System**, norma pre užívateľské rozhranie na princípe oknového systému. V roku 1992 vydali meta-normu **Computer Graphics Reference Model** (CGRM), ktorá by mala byť počiatkom tzv. druhej generácie grafických noriem. Vyše tisícstranová norma o spracovaní obrazu sa nazýva **Image Processing and Interchange** (IPI, 1993). Hlavnou výhodou programov, využívajúcich NGS, je medzinárodne garantovaná prenositeľnosť. Hlavnou nevýhodou písania o NGS je fixovanie pohľadu na pohyblivý cieľ - medzinárodná normotvorba v oblasti spracovania grafickej informácie priam exploduje. Od roku 1977 ju koordinuje ISO, ktorej členmi sú národné

organizácie - za SR Úrad pre normalizáciu, meranie a skúšobníctvo SR. Grafické normy ako technické normy patria pod Slovenský ústav technickej normalizácie. V roku 1987 ISO podpísala dohodu s IEC (*International Electrotechnical Committee*) o spolupráci (*double standards*) v Spoločnej technickej komisii 1 (*Joint Technical Committee 1, JTC1 Information Processing*), ktorá má viacero podkomisií (*Subcommitte, SC*). Grafické normy dnes pripravuje podkomisia **ISO/IEC JTC1/SC24 COMPUTER GRAPHICS AND IMAGE PROCESSING,** ktorá má 4 pracovné skupiny: **Multimedia Presentation and Interchange, Image Processing and Interchange, Language Bindings and Registration** a **Architecture**, ktoré vyvíjajú normy v nasledujúcich projektoch:

24.1 Graphical Kernel System

24.2 GKS Language Bindings

24.3 PHIGS

24.4 PHIGS Language Bindings

24.5 Computer Graphics Metafile

24.6 CGI

24.7 Conformance Testing

24.8 Reference Model

24.9 Registration (procedúry registrácie normovaných produktov)

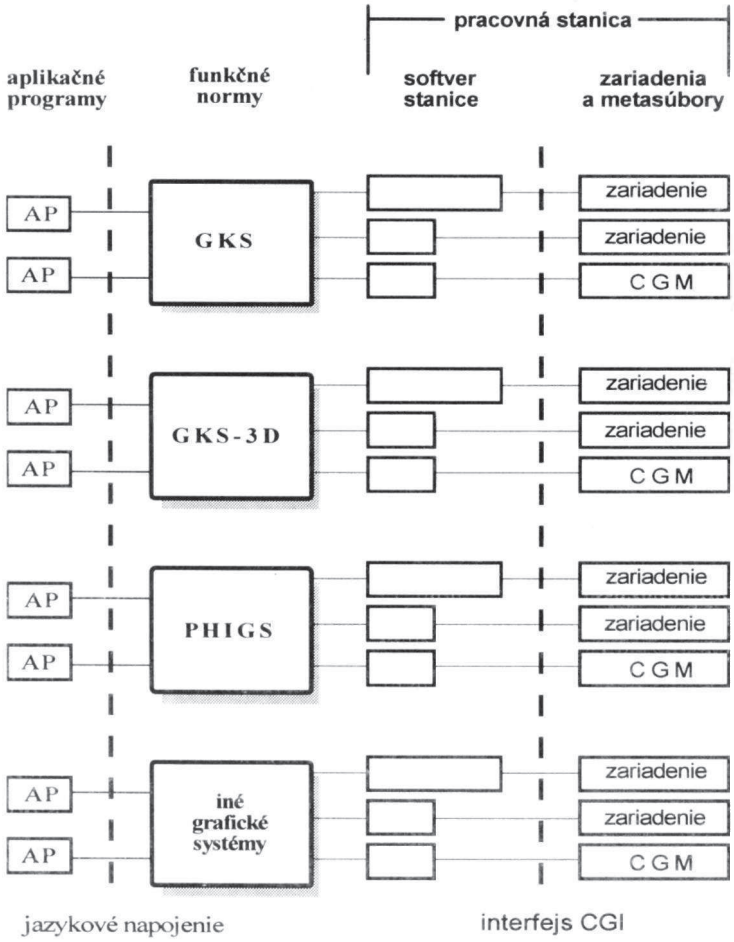
24.10 Image Processing and Interchange

24.11 PREMO

Hlavný dôraz kladieme na normu PHIGS, ktorá zahŕňa 2D i 3D interaktívnu grafiku a hierarchiu objektov. Uvažovanie o funkčnosti grafiky v pojmoch normy PHIGS sa javí globálne prijatým terminologickým štandardom aj v najnovších učebniciach [HEAR94], [FOLE94]. V príkladoch pre mená funkcií a parametrov NGS používame ich celé plnovýznamové znenia namiesto normou predpísaných skratiek. Pripúšťame miešanie typov hodnôt, napr. real a integer. Predpokladáme, že potrebné deklarácie mien funkcií a parametrov sa už vykonali inde. Prípravou na výklad normy PHIGS bude výklad normy GKS, ktorý použijeme ako nástroj na definovanie potrebných pojmov.

Pri vývoji niektorých noriem (GKS, PHIGS) vysvitlo, že konceptuálny model základnej štruktúry aplikačného grafického systému (aplikácia - grafický systém - zariadenie) nedostačuje. Vznikol preto podrobnejší model, ktorý sa nazýva **referenčný model grafických noriem,** obr. 17.1. Rozhranie na grafický systém sa tu špecifikuje v 2 krokoch. V 1. kroku sa určí **funkčná norma**, napr. PHIGS, ktorá ale nezávisí na programovacom jazyku, a preto treba medzi program a PHIGS vložiť tzv. **jazykové napojenie** (*language binding*), ktoré v konkrétnom jazyku stanovuje reprezentáciu každej funkcie PHIGS. Jazykové napojenie možno považovať za rozhranie aplikačného programu (*aplication interface*). Pri projektovaní programu teda uvažujeme v pojmoch funkčnej normy, kým pri programovaní v pojmoch jej jazykového napojenia. Grafický systém realizuje požiadavky aplikačného programu na danom zariadení, ale tiež by nemal závisieť od konkrétneho fyzického zariadenia. Konceptuálna obdoba fyzického zariadenia sa nazýva **pracovná stanica** (*workstation*). Grafické systémy umožňujú prácu s viacerými stanicami. Stanica je abstrakciou fyzického zariadenia, ktoré spĺňa všetky

požiadavky grafického systému. Aby sa konkrétne fyzické zariadenie zúplnilo na takú úroveň, treba ho podporiť softverom, ktorý sa nazýva **softver stanice**. Medzi softverom stanice a vlastným zariadením teda existuje ďalšie rozhranie. Normovaním tohto rozhrania sa zaoberá norma CGI (*Computer Graphics Interface*). Nazýva sa aj **rozhraním medzi na zariadení závislou a nezávislou časťou grafického systému.**



Obr. 17.1 Referenčný model grafických noriem

17.2 Grafický výstup podľa normy GKS

Medzinárodná norma Graphical Kernel System (GKS) [IS7942] špecifikuje množinu funkcií na programovanie aplikačných programov, využívajúcich interaktívnu počítačovú 2D grafiku. Táto norma odpovedá na potrebu štandardizovanej metódy na vývoj grafických programov a reprezentuje štandardné grafické rozhranie s konzistentnou syntaxou. Funkčnosť GKS sa špecifikuje nezávislo od programovacieho jazyka i od fyzického zariadenia, no možno ho považovať za programovací jazyk v tom zmysle, že

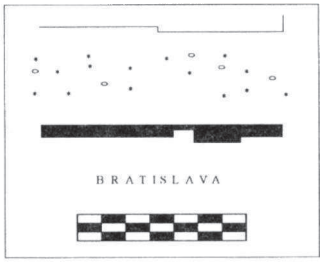
ponúka programátorovi konzistentnú množinu rezervovaných slov, ktoré možno vyjadriť v konkrétnom programovacom jazyku. Vyjadrenie pojmov GKS v programovacom jazyku (napr. C alebo Fortran 77) presne definuje jazykové napojenie. V jazyku C sa podľa jazykového napojenia body zadávajú typom Gpoint (dvojica súradníc) v poli s menom napr. CIARA. Skutočné volanie v C je teda gpolyline(&ciara).

Štruktúru GKS tvoria nasledujúce skupiny prvkov. **Grafický výstup** popisujú **grafické výstupné prvky**, ich **parametre** a **atribúty** a **súradnicové systémy**, v ktorých sa prvky špecifikujú. Obrázky možno rozdeliť do **segmentov** a so segmentami a ich atribútmi ďalej pracovať. Interakciu špecifikujú **logické vstupné zariadenia** a **vstupné režimy**. Ústredným pojmom GKS je **pracovná stanica**, abstrakcia fyzického zariadenia. Funkčnosť dopĺňa **prostredie** GKS, **metasúbor** na ukladanie a prenos obrázkov, **zisťovacie funkcie** a **spracovanie chýb**.

Obrázky zostrojíme z častí, ktoré nazývame **grafické výstupné prvky**:

1. **lomená čiara**: kreslí postupnosť spojených úsečiek
2. **sled značiek**: označí postupnosť bodov značkou
3. **výplňová oblasť**: kreslí plochu danú bodmi hranice
4. **text**: vykreslí znakový reťazec
5. **pole buniek**: vytvorí obraz z buniek rôznych farieb
6. **zovšeobecnený grafický výstupný prvok**:

   vykreslí napr. elipsu, ak to vie zariadenie.



Obr. 17.2 Výstupné prvky

Grafický prvok má množinu **parametrov**, ktoré určujú jeho konkrétny tvar. Napr. TEXT má dva parametre - pozíciu a reťazec, príkaz TEXT (*WHERE, "ABC"*) vykreslí znaky *ABC* na pozícii *WHERE*. Kým parametrami riadime tvar grafických prvkov, ich **aspekty vzhľadu** (napr. farbu, font) riadime **atribútmi**. Atribút môže mať rovnakú hodnotu pre viac prvkov, platí **modálne**. Napr. s daným fontom možno vykresliť niekoľko reťazcov.

Atribúty v GKS patria do 2 skupín - **1. globálne**, ktoré možno použiť na všetkých staniciach, a **2. závislé na stanici**. Globálne atribúty možno nastaviť **individuálne**, napr. priamo výšku znaku. "Lokálne" závislé atribúty združujeme do tzv. **zväzkov** (*bundles*) a riadime indexom v tabuľke zväzku pre danú stanicu. Index určuje **reprezentáciu prvku na stanici.**

Lomenú čiaru vykreslíme volaním funkcie

POLYLINE (*N, POINTS*),

kde *POINTS* je zoznam *N* bodov - od bodu *POINTS(1)* po bod *POINTS(N)*. Lomená čiara sa skladá z N-1 úsečiek, spájajúcich po sebe nasledujúce body. *POLYLINE* (*5, CIARA*) vykreslí 4 úsečky, ktoré spoja 5 bodov *CIARA(1)* až *CIARA(5)*. Ak chceme rozlíšiť dve lomené čiary, nastavíme im rozdielne indexy. Presná reprezentácia lomenej čiary (**farba, hrúbka a typ čiary**) bude závisieť na konkrétnych možnostiach stanice. **Index lomenej čiary** nastavíme funkciou SET POLYLINE INDEX (*N*). Atribút index lomenej čiary bude mať hodnotu *N* pre všetky lomené čiary, kým ho znova nezmeníme.

Množinu bodov znázorníme použitím prvku **sled značiek**.

POLYMARKER (*N, POINTS*),

má také isté parametre ako lomená čiara, ale umiestni v každom z bodov značku, charakterizovanú podrobne **typom, farbou a veľkosťou**, napr. najmenšia modrá bodka. Sledy značiek možno tiež rozdielne reprezentovať, a to rozdielnymi indexami sledu značiek, nastavenými funkciou SET POLYMARKER INDEX (*N*), kde *N* je požadovaná hodnota indexu sledu značiek.

Výplňovú oblasť zadávame zoznamom bodov jej hranice. Ak prvý bod nesplýva s posledným, GKS ich spojí, aby sa hranica uzavrela. Inak je význam parametrov ten istý ako pre predchádzajúce funkcie:

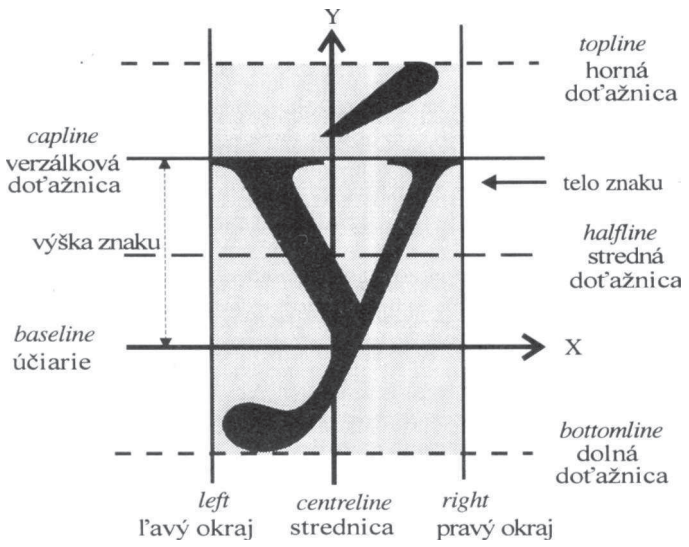
FILL AREA (*N, POINTS*).

Reprezentáciu výplňovej oblasti takisto riadime atribútom index výplňovej oblasti. Nastavujeme ho tiež modálne, funkciou SET FILL AREA INDEX (*N*). Výplňové oblasti možno rozlišovať spôsobom vyplnenia (v GKS sa nazýva **typ vnútra**: prázdne, plné, šrafované) a **farbou**. Pre prázdne vnútro (*HOLLOW*) sa vykreslí iba hranica výplňovej oblasti. Ak chceme výplňovú oblasť s hranicou, treba vykresliť okolo nej lomenú čiaru. Hranica oblasti sa však môže pretínať. Ktoré plochy sa v takomto prípade vyplnia, ak zvolíme typ vnútra plný? Čo je vnútrom danej oblasti? GKS rieši tento problém podľa známeho pravidla: ak polpriamka z daného bodu, neprechádzajúca vrcholom hranice oblasti, pretína zadanú hranicu v párnom počte bodov (vrátane nuly), tak bod je vonku, inak je vnútri.

Grafický prvok **text** sme už spomenuli:

TEXT (*POSITION, STRING*).

Rozdielne požiadavky kvality a rýchlosti zobrazovania textu rieši GKS troma presnosťami textu. Každý z doteraz uvedených grafických prvkov mal jeden atribút (index), ktorý (prostredníctvom reprezentácie) riadil jednotlivé aspekty vzhľadu prvku. Keďže text je zložitejší, ovplyvňuje jeho vzhľad viac aspektov. Niektoré z nich sa neriadia indexom (reprezentáciou), a pritom sa nemenia pri každom texte (ako sa menia práve parametre textu: pozícia a znakový reťazec). Tieto dôležité aspekty textu sú výška znaku, sklon znaku, smer textu a zarovnanie textu. Aspekty vzhľadu textu riadime atribútmi a každému atribútu textu môžeme priradiť individuálnu hodnotu. Samozrejme, aj reprezentáciu textu riadime indexom textu. Najprv však uveďme **globálne atribúty textu: výška znaku, vertikálny vektor znaku, smer textu a zarovnanie textu**. Pozícia textu sa v GKS neznačkuje a nemusí byť vždy rovnaká vzhľadom k vykreslenému reťazcu. Atribútom zarovnanie textu môžeme zarovnávať text samostatne pozdĺž horizontálnej i vertikálnej osi, pričom vertikálna os je orientovaná v smere vertikálneho vektora znaku (definujeme ho nižšie). Zarovnávať môžeme horizontálne na pravý i ľavý okraj a na stred reťazca, vertikálne dokonca piatimi spôsobmi. Ozrejmia to niektoré typografické pojmy:



Obr. 17.3 Súradnicový systém na popis druhu písma

Funkciou na nastavenie **zarovnania textu** SET TEXT ALIGNMENT (*HORIZ, VERT*) riadime horizontálne zarovnanie *HORIZ* i vertikálne zarovnanie *VERT*. Každý z nich môže nadobudnúť aj hodnotu *NORMAL*. Atribút **výška znaku** určuje výšku znakov. Pretože definícia znaku obsahuje pre každý druh písma aj pomer strán tela znaku, určuje výška znaku zároveň aj jeho šírku. Zmenou výšky znaku pomocou funkcie SET CHARACTER HEIGHT (*H*) sa teda zároveň zmení aj šírka znaku.

Atribútom textu **vertikálny vektor znaku** riadime predovšetkým orientáciu znakov, ale nastavíme aj referenčný smer, ktorý sa používa pri stanovení smeru textu a pri zarovnaní. Vertikálny vektor znaku určí aj sklon jednotlivých znakov a orientáciu znakov v reťazci, pretože znaky sa implicitne umiestňujú na priamku kolmú na vertikálny vektor znaku. SET CHARACTER UP VECTOR (*X, Y*) nastaví *X* a *Y* zložky vertikálneho vektora znaku, čím zadáme odklon znaku od zvislého smeru a tým aj uhol odklonu textu. Veľkosť vektora sa nevyužíva, SET CHARACTER UP VECTOR (*-1, 1*) má ten istý účinok ako SET CHARACTER UP VECTOR (*-15, 15*). Oba spôsobia, že sa nasledujúce reťazce vykreslia v 45-stupňovom sklone ku horizontu (o implicitnom smere textu budeme hovoriť neskôr). Nie každým písmom sa píše zľava doprava a ani my vždy nemusíme vyžadovať, aby bol smer textu kolmo na vertikálny vektor znaku. Napr. v grafoch sa os Y často označuje "hotelovým" spôsobom písania písmen pod sebou. Atribút **smer textu** riadime funkciou SET TEXT PATH (*PATH*), kde *PATH* nadobúda jednu zo štyroch hodnôt: *RIGHT, LEFT, UP* a *DOWN*. Všetky sa vzťahujú ku smeru, definovanému vertikálnym vektorom znaku. Implicitná hodnota *RIGHT* (vpravo) spôsobí, že sa znaky vykresľujú jeden za druhým na priamke, kolmej (v smere chodu hodinových ručičiek) na vertikálny vektor znaku. *LEFT* (vľavo) spôsobí opačné umiestňovanie znakov, vhodné pre systémy písma, písané sprava doľava. *DOWN* (dole) spôsobí umiestňovanie znakov jeden pod druhým, kým *UP* (hore) jeden nad druhým. Napr. šikmý nápis *HOTEL PISA*, písaný zhora dole:

SET CHARACTER UP VECTOR (*1, 6*)

SET TEXT PATH (DOWN)

TEXT (*WHERE, "HOTEL PISA"*)

Ako sme už uviedli, text má tiež **reprezentáciu**, dostupnú prostredníctvom **indexu textu**. SET TEXT INDEX (*N*) nastaví požadovanú hodnotu. Nech reprezentácia textu 1 je románsky druh písma so štandardným tvarom znakov, s nulovým rozostupom znakov a štandardnej farby a s presnosťou na časť znaku, **ťah písma** (*STROKE*, text najvyššej kvality). Nech reprezentácia textu 2 je taká istá, iba s presnosťou **na znak** (*CHAR*, stredná kvalita) a reprezentácia 3 má najnižšiu presnosť, **na reťazec** (*STRING*). Potom príklad:

SET CHARACTER UP VECTOR (*-1, 1.414*)

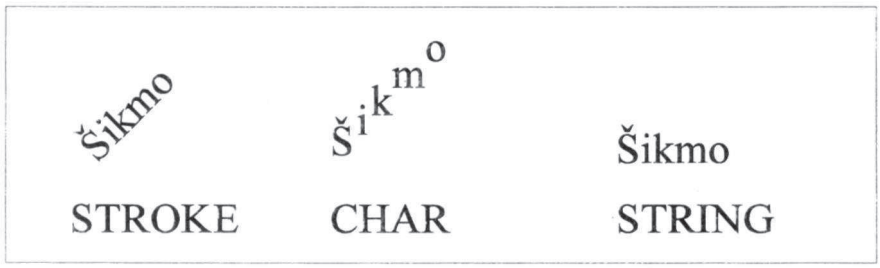
SET TEXT PATH (*RIGHT*)

SET TEXT ALIGNMENT (*LEFT, BASE*)

SET TEXT INDEX (*1*)

TEXT (*WHERE, "Šikmo"*)

vykreslí presne reprezentovaný textu, kým použitím indexu 3 získame na pozícii WHERE vodorovný reťazec (najnižšia presnosť textu STRING). Iný druh písma, napr. kurzívu možno získať ďalším indexom textu a inou príslušnou reprezentáciou textu.

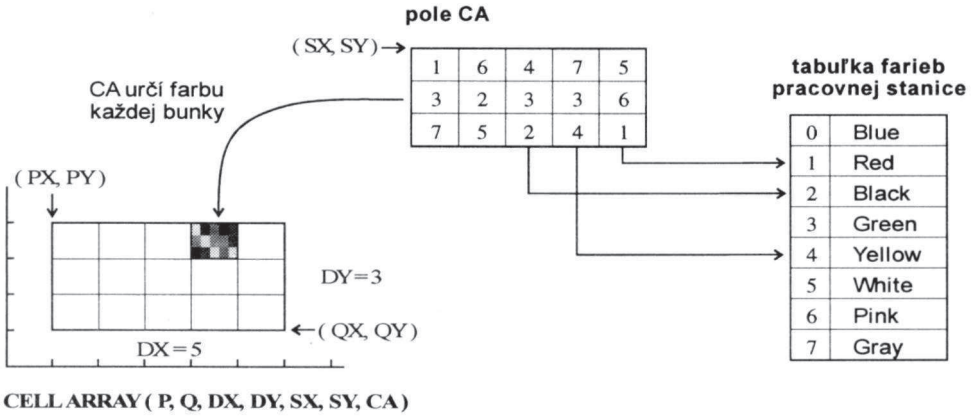


Obr. 17.4 Šikmé texty podľa presnosti

Na rastrových obrazovkách sa často vyskytujú obrázky s opakovaným vzorkovaním farieb alebo úrovní šedej, virtuálne pole pixlov - **pole buniek**.

CELL ARRAY (*P, Q, DX, DY, SX, SY, CA*),

kde body *P* a *Q* určujú protiľahlé vrcholy (osovo zarovnanej) obdĺžnikovej oblasti, ktorá je konceptuálne rozdelená do *DX* buniek v smere *X* a do *DY* buniek v smere *Y*. S každou bunkou (*I,J*) je združená farba, stanovená v položke poľa indexov farieb *CA(I, J)*, kde I nadobúda hodnoty od *SX* po (*SX+DX-1*) a *J* prechádza od *SY* po (*SY+DY-1*). Bunke *P* (*PX,PY*) teda prislúcha farba *CA*(*SX,SY*) a bunke *Q* farba *CA*(*SX+DX-1, SY+DY-1*). Prvok matice *CA(I,J)* obsahuje v skutočnosti pozíciu v tabuľke farieb na stanici, ale zatiaľ to považujme priamo za farbu. Pole buniek sa špecifikuje v priestore modelu a podlieha všetkým transformáciám tak ako iné grafické výstupné prvky. Použitie poľa buniek na nerastrovom zariadení má dať jeho čo najlepšiu aproximáciu.



Obr. 17.5 Pole buniek

Trh ponúka pracovné stanice, ktoré majú hardverovo implementované kreslenie elíps či splajnov. Aby sa tieto normou nepovolené schopnosti dali využiť, ponúka GKS **zovšeobecnený grafický výstupný prvok** (GDP)

GENERALIZED DRAWING PRIMITIVE (*N, POINTS, ID, LDR, DR*),

kde *ID* špecifikuje typ *GDP*. *POINTS* určujú súradnice *N* bodov, ktoré sa použijú na definovanie *GDP*. Keďže *GDP* môže mať aj ďalšie vlastnosti, nevyjadriteľné súradnicami, dovoľuje pole *DR* (dátový záznam, *data record*) dĺžky *LDR* popísať aj tieto vlastnosti. Napr. *GDP* môže byť kruhový oblúk, *N* sa nastaví na 3 a tri body určujú stred kružnice a krajné body oblúka. Parameter *LDR* možno nastaviť na indikovanie jedinej hodnoty *DR(1)*, ktorá určuje, či sa má oblúk kresliť v smere chodu hodinových ručičiek alebo opačne. Pre GDP možno využívať atribúty naraz z viacerých výstupných prvkov. Napr. kruhový oblúk môže použiť index lomenej čiary na stanovenie typu čiary oblúka. GDP umožňuje na jednej strane bohatšiu inštaláciu GKS, ale zároveň oslabuje prenositeľnosť.

17.2.1 Riadenie atribútov

Pred popisom funkcie POLYLINE sme spomenuli dve skupiny atribútov - **globálne a závislé na stanici**. Pri každom grafickom výstupnom prvku sme mali index, ktorým sme dosahovali rozdielnu reprezentáciu vzhľadu. Neuviedli sme však (s výnimkou niektorých atribútov textu), ako sa reprezentácia pre daný index nastaví. Na každej stanici je **tabuľka zväzkov** atribútov, do ktorej ukazuje index príslušného výstupného prvku. Položky tejto tabuľky nastavujeme napr. na stanici *WS* pre POLYLINE INDEX 1 funkciou SET POLYLINE REPRESENTATION (*WS, 1, LINETYPE, LINEWIDTH, COLOUR INDEX*), kde *LINETYPE* nastaví typ čiary, *LINEWIDTH* jej hrúbku a (*POLYLINE*) *COLOUR INDEX* farbu čiary. Potom už možno použiť SET POLYLINE INDEX (*1*). Na tej istej stanici (alebo na všetkých) však možno žiadať aj aby sa všetky čiary kreslili napr. biele. Vtedy treba riadiť atribút farby čiary individuálne použitím funkcie SET POLYLINE COLOUR INDEX (index do tabuľky farieb na bielu). Je jasné, že tieto dva spôsoby riadenia atribútov si odporujú - globálne nastavená biela farba sa nemusí zhodovať s

farbou pre index 1 na stanici *WS*. Aby sa správne určilo, ktorá možnosť práve platí, zaviedol sa mechanizmus **príznaku pôvodu** (*aspect source flag, ASF*). Ku každému atribútu teda patrí príznak, ktorý určuje, či sa hodnota atribútu vezme z individuálneho nastavenia alebo zo zväzku. Funkciou SET ASPECT SOURCE FLAGS (*LIST*) nastavíme všetkých 13 položiek zoznamu *LIST* do príslušných príznakov pôvodu atribútov. Hodnota položky zoznamu *LIST* i príslušného *ASF* je buď *BUNDLED* alebo *INDIVIDUAL*.

17.2.2Príklad použitia grafického výstupu

Nasledujúci príklad je z jazykového napojenia C, [IS8651]. Ponechali sme z neho len časť komentárov. Celý text možno nájsť v [ERTL93].

Program STAR

```
/* Tento program kreslí žltú hviezdu na modrom pozadí a vypíše biely nápis "STAR" pod
hviezdou. Úroveň: GKS 0a
Vypočítaj súradnice bodov hviezdy
Otvor GKS a otvor/ a aktivizuj pracovnú stanicu
Vycentruj okno na počiatok
Definuj použité farby
Nastav vnútro výplňovej oblasti na plnú a jej farbu na žltú
Vykresli hviezdu
Vyber veľké biele písmená centrované na stred
Vypíš nápis
Uzavri všetko: deaktivuj a zavri stanicu i GKS */
```

17.3 Súradnicové systémy v GKS

Vzhľad výstupu závisí aj od súradníc, v ktorých ho popisujeme, a preto v GKS možno definovať počiatok súradnicovej sústavy i mierky na oboch osiach podľa potreby. Tieto karteziánske súradnice v priestore modelu nazývame **svetové súradnice** (*World Coordinates,* WC). Pretože sa rozsahy súradníc rôznych zariadení výrazne líšia, zavádza sa v GKS **normalizované zariadenie**, ktorého zobrazovacou plochou je jednotkový štvorec, súradnice X aj Y sú z uzavretého intervalu [0, 1]. Tieto súradnice budeme nazývať **normalizované súradnice** (*Normalized Device Coordinates*, NDC). Výstup vo WC treba teda zobraziť na časť priestoru NDC. Oblasť nášho záujmu v modeli zadáme ako **okno** (obdĺžnik v priestore WC) a vymedzíme, na ktorú časť zobrazovacieho zariadenia ho mienime mapovať, aký bude **záber** (obdĺžnik v priestore NDC). Voláme dvojicu funkcií

SET WINDOW (*N, XWMIN, YWMIN, XWMAX, YWMAX*),

SET VIEWPORT (*N, XVMIN, YVMIN, XVMAX, YVMAX*),

kde *N* je číslo transformácie. Zvyšné parametre definujú hranice okna a záberu, čiže transformáciu WC do NDC, ktorú nazývame **normalizujúca transformácia**. Môžeme definovať viac normalizujúcich transformácií. Okná a zábery danej aplikácie je vhodné definovať v záhlaví programu. (Poznamenajme, že okno v oknovom systéme a GUI zodpovedá záberu!) Na výber normalizujúcej transformácie slúži funkcia SELECT NORMALIZATION TRANSFORMATION (*N*). Implicitná hodnota N je nula.

Normalizujúcu transformáciu 0 nemožno predefinovať. Implicitná definícia SET WINDOW (*0, 0, 1, 0, 1*) a SET VIEWPORT (*0, 0, 1, 0, 1*) zaručí možnosť definovať výstup priamo v NDC namiesto vo WC (a že neexistuje vstup mimo záberu).

Orezávanie na záber riadime funkciou SET CLIPPING INDICATOR (*IND*), kde *IND* môže nadobudnúť hodnoty *CLIP* (implicitne) a *NOCLIP*. Indikátor orezávania súvisí vždy s práve platnou normalizujúcou transformáciou. Samostatný indikátor orezávania pre každú normalizujúcu transformáciu neexistuje. Ak sme nastavili orezávanie, orežú sa lomené čiary vždy na hranicu záberu. Pre sled značiek môže časť značky niekedy presiahnuť hranicu záberu. Orezávanie textu závisí od presnosti textu. Najnižšia presnosť *STRING* môže spôsobiť, že sa nezobrazí celý text, ak je jeho pozícia mimo záberu. Text s presnosťou *CHAR* sa orezáva po znakoch. Ak je časť znaku mimo záber, vynechá sa celý znak. Presnosť na ťah písma *STROKE* vyžaduje orezať presne na hranicu záberu.

17.4 Segmenty a ich atribúty

Často treba obrázky alebo ich časti uložiť a neskôr znova použiť. Na tento účel má GKS pojem **segment**, časť pamäti označenú menom segmentu. Segment má mnoho atribútov, ktorými môžeme riadiť vzhľad a výskyt segmentov v čase ich využitia. Segment vytvoríme funkciou CREATE SEGMENT (*ID*), *ID* je meno segmentu. Tým vznikne **otvorený segment**, zároveň s vykreslením sa výstupné prvky vkladajú do otvoreného segmentu, až kým segment neuzavrieme funkciou CLOSE SEGMENT. Otvoriť možno vždy iba jeden segment. Do uzavretého segmentu sa už nedajú vkladať ďalšie výstupné prvky. Segment **zrušíme** volaním funkcie DELETE SEGMENT (*ID*) a meno *ID* možno použiť znova. Segment možno **premenovať**: RENAME SEGMENT (*OLD, NEW*), kde *OLD* a *NEW* sú staré a nové meno segmentu, pričom už nesmie existovať segment s menom *NEW*. Premenovať možno aj otvorený segment.

Atribút **transformácia segmentu** umožní efektívne naraz manipulovať so všetkými súradnicami v definícii segmentu. Pri otvorení segmentu je jeho transformácia identická, no dá sa postupne meniť:

SET SEGMENT TRANSFORMATION (*ID, MATRIX*),

kde *ID* je meno segmentu (môže byť aj otvorený), *MATRIX* je transformačná matica rozmeru 2*3. Zostrojuje sa funkciami: EVALUATE TRANSFORMATION MATRIX(*FX, FY, TX, TY, R, SX, SY, SWITCH, MATRIX*) a ACCUMULATE TRANSFORMATION MATRIX (*MATIN, FX, FY, TX, TY, R, SX, SY, SWITCH, MATOUT*), kde *FX* a *FY* špecifikujú **pevný bod**, *TX* a *TY* definujú **vektor posunu**, *R* je **uhol rotácie** v radiánoch a SX a SY sú **škálovacie faktory**. Vypočítaná matica transformácie sa vráti v parametri *MATRIX*. Mierka daná škálovacími faktormi a rotácia sa vzťahujú na zadaný pevný bod. **Prepínač súradníc** *SWITCH* môže nadobudnúť hodnoty WC a NDC. V prvom prípade sa hodnoty pre pevný bod a vektor posunu berú vo WC, inak v NDC. Pre hodnotu *SWITCH* = WC sa pevný bod (*FX, FY*) a vektor posunu (*TX, TY*) transformujú do NDC podľa platnej normalizujúcej transformácie. Prvky výslednej matice sa vyjadrujú v NDC. Matica sa počíta v poradí transformácií zmena mierky, rotácia a posunutie. Transformácia určená týmito parametrami sa v druhej funkcii zloží s transformáciou zadanou v matici *MATIN* a výsledná transformácia sa navráti v matici *MATOUT*.

Poradie skladania je vstupná matica, zmena mierky, rotácia a posun. Uveďme príklad použitia EVALUATE TRANSFORMATION MATRIX. Majme v segmente nejakú lomenú čiaru, začínajúcu sa v bode *(0, 8.8)*. Ak ju chceme posunúť, aby sa začínala v bode *(5, 13.8)*, musíme zostrojiť maticu posunutia o *(5, 5)*. Na to netreba ani zmenu mierky ani rotáciu. Na pevnom bode pri posunutí nezáleží - zadáme hoci *(0, 0)*:

Príklad transformácie segmentu

```
...
CREATE SEGMENT (1)
POLYLINE (44, BODY)
CLOSE SEGMENT
TX = TY = 5
R = 0
SX = SY = 1
FX = FY = 0
SWITCH = WC
EVALUATE TRANSFORMATION MATRIX (FX,FY,TX,TY,R,SX,SY,SWITCH,MAT)
SET SEGMENT TRANSFORMATION (1, MAT)
...
```

Veľkosť zobrazeného segmentu riadime škálovacími faktormi. Dvojnásobné zmenšenie dosiahneme zadaním *SX = SY = 0.5*. Rotáciu proti smeru hodinových ručičiek o 30 stupňov (PI/6 radiánov) by sme získali zadaním *R = PI/6*. Treba rešpektovať poradie skladania transformácií (vstupná matica, zmena mierky, rotácia a posun), lebo posunom či rotáciou pred zmenou mierky môžu vzniknúť rozdielne výsledky, ak sa *SX* líši od *SY*.

Účinok **orezania** transformovaného segmentu závisí od času vykonania transformácie segmentu. Táto sa vykonáva po normalizujúcej transformácii, ale pred akýmkoľvek orezávaním. Súradnice v segmentoch sú v NDC a takisto aj prvky matice transformácie. Grafický výstupný prvok sa v segmente transformuje normalizujúcou transformáciou a transformáciou segmentu. Ak bol nastavený indikátor orezania na *CLIP* v čase, keď sa grafický prvok zaraďoval do segmentu, vykoná sa orezanie na záber (normalizujúcej transformácie), ktorý bol aktívny v tom čase. V každom segmente sa skutočne ukladajú orezávacie obdĺžniky pre každý grafický prvok. Tieto obdĺžniky sa však transformáciami segmentu netransformujú. Ďalšie atribúty segmentu sú **viditeľnosť, zvýraznenie, a priorita segmentu,** ktorou riadime poradie, v akom sa segmenty prekresľujú pri zmenách obrazu. Atribút **detekovateľnosť** budeme môcť popísať neskôr.

17.5 Logické vstupné zariadenia a režimy

Hardver umožňuje vstupné dáta zadávať v rozličných formách, ktoré však aplikačný program musí spracovať nezávislo od zariadenia. Abstrahujeme teda od fyzických zariadení a podľa typu vstupných dát identifikujeme 6 tried logických vstupných zariadení:

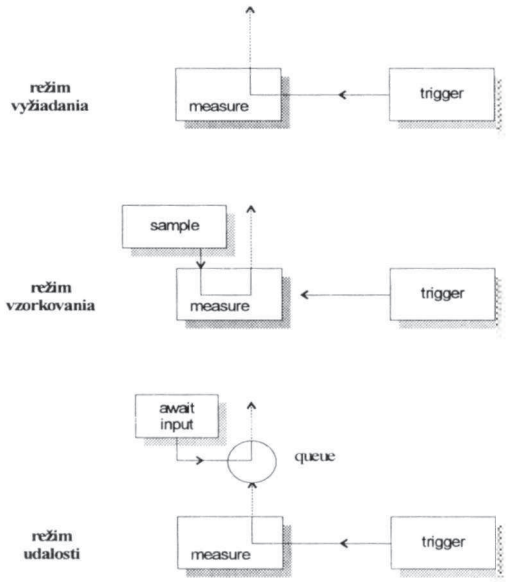
1. **lokátor** - umožňuje zadať pozíciu - hodnotu *(X, Y)*
2. **výber** (*pick*) - umožňuje identifikáciu niektorého segmentu menom
3. **voľba** (*choice*) - umožňuje voľbu z menu číslom voľby
4. **valuátor** - umožňuje zadanie hodnoty

5. **reťazec** (*string*) - umožňuje zadanie reťazca znakov

6. **sled polôh** (*stroke*) - umožňuje zadanie postupnosti pozícií

Lokátor, výber a sled polôh dávajú vstup priamo súvisiaci s obrazom, ostatné vstupné triedy slúžia na komfortnejšiu interakciu. Vstupné hodnoty dostane aplikačný program v NDC - v jednotkovom štvorci - po spätnej transformácii zo súradníc fyzického zariadenia (*Device Coordinates*, DC). Popíšme stručne vstupný model GKS. Aplikačný program potrebuje od operátora, aby zadal **vstupnú hodnotu** (*input value*). Logické vstupné zariadenie sa inicializuje na nejakú hodnotu, ktorej hovoríme **priebežná hodnota** (*measure*), napr. prázdny reťazec pre zariadenie triedy reťazec. Operátorovi sa adresuje **výzva** (*prompt*), napr. blikajúci kurzor, aby zadal vstup. Operátor upravuje priebežnú hodnotu, ktorú vhodne zobrazuje **echo,** napr. zobrazuje sa zadávaný reťazec. Keď je operátor spokojný, stlačí **spúšť** (*trigger*), napr. klávesu Enter, čím zmení priebežnú hodnotu na vstupnú, ktorú už nemôže meniť. Príslušné logické vstupné zariadenie poskytne aplikačnému programu vstupnú hodnotu. Aplikačný program môže operátorovi oznámiť, že vstupnú hodnotu prijal. Podľa spôsobu poskytnutia vstupnej hodnoty rozoznávame **režim vyžiadania**, **vzorkovania** a **udalosti**.

Každé logické vstupné zariadenie môžeme použiť v jednom z troch vstupných režimov. V operačných režimoch logického vstupného zariadenia sa špecifikuje, kto (operátor alebo aplikačný program) má iniciatívu:



Obr. 17.6 Vstupné režimy

1. **vstup vyžiadaním** zadá operátor po výzve od programu. Vstupný proces sa odštartuje, na požiadanie (funkciou REQUEST) doručí vstupné dáta aplikačnému programu a vráti sa do stavu čakania. Aktívny je iba jeden proces, nikdy nie oba naraz.

2. **vstup vzorkovaním** si vyžiada priamo aplikačný program. Vstupný proces pracuje na pozadí tak, že poskytuje zo zariadenia vždy najnovšie vstupné dáta, ktoré aplikačný program môže (funkciou SAMPLE) alebo nemusí využiť.

3. vstup udalosťou generuje asynchrónne operátor. Vstupný proces dáta uloží do fronty, odkiaľ ich môže získať aplikačný program, a očakáva, že aplikačný program bude konať v závislosti od získaných dát. Operátor riadi prístupnosť vstupných dát, a tak efektívne riadi aj interakciu.

Všetky logické vstupné zariadenia môžu pracovať v každom z týchto režimov. V režime vyžiadania môže v danom čase byť aktívne najviac jedno zariadenie. Zároveň môže byť niekoľko zariadení aktívnych v režime vzorkovania a ostatné môžu byť aktívne v režime udalosti, čo umožňuje programovanie čo najvhodnejšej interakcie.

Každé zariadenie môže byť v danom okamihu iba v jednom režime, ktorý volíme volaním funkcie typu SET XXX MODE (*WS, DV, MODE, EC*), kde XXX, *WS* a *DV* určujú zariadenie, *MODE* definuje jeden z troch režimov REQUEST, SAMPLE a EVENT. Posledný parameter *EC* môže nadobudnúť jednu z dvoch hodnôt *ECHO, NOECHO*. *ECHO* znamená, že sa napr. na obrazovke má indikovať, že do programu vstupujú dáta. Konkrétne zariadenie reťazca by malo echovať stlačenie klávesy na klávesnici vypísaním príslušného znaku. Implicitný operačný režim pre všetky zariadenia je režim vyžiadania. Ďalšia implicitná hodnota je stav *ECHO*.

Režim vyžiadania takmer identicky zodpovedá príkazu READ v bežnom programovacom jazyku. Program čaká, kým mu vstup dodá požadovanú hodnotu

REQUEST XXX (*WS, DV, ST, ...*),

kde XXX je trieda logického vstupného zariadenia (lokátor, výber, ...). *WS* je pracovná stanica, ktorá vstupné zariadenie obsahuje. *DV* špecifikuje, o ktoré zo zariadení danej triedy ide. Stavový indikátor *ST* vracia príznak platnosti vstupu. Tri bodky znamenajú parametre špecifické pre každé zariadenie. Napr. lokátor vyžaduje aj číslo normalizujúcej transformácie, kým reťazec od súradníc nezávisí. So zariadením výberu (segmentu) zas súvisí **detekovateľnosť segmentu**. Riadime ho funkciou SET DETECTABILITY (*ID, DET*), kde *ID* je identifikátor segmentu a *DET* je požadovaná detekovateľnosť - *DETECTABLE* alebo (implicitná) *UNDETECTABLE*. Segment nemožno vybrať, keď nie je detekovateľný, ani vtedy, keď je neviditeľný.

Na získanie hodnoty **v režime vzorkovania** slúži funkcia

SAMPLE XXX (*WS, DV, ...*).

Pre všetky zariadenia **v režime udalosti** existuje jediná **vstupná fronta**. Je však možné, aby bolo v režime udalosti aktívnych niekoľko zariadení súčasne. Vstupné dáta sa do fronty pridávajú za sebou v tom poradí, v akom sa generujú, spolu s identifikáciou zariadenia. Funkcia na riadenie fronty je

AWAIT EVENT (*TIMEOUT, WS, DVCLASS, DV*),

ktorá zistí, či je fronta prázdna. Ak áno, aplikačný program sa pozastaví, kým nevznikne udalosť - negenerujú sa vstupné dáta - alebo kým nevyprší čas, zadaný v sekundách v TIMEOUT. Ak čas vyprší, udalosť nevznikne, DVCLASS vráti hodnotu NONE. Ak je TIMEOUT nulový, program sa nepozastavuje a platí hodnota NONE. Ak fronta nie je prázdna, AWAIT EVENT vráti informáciu o prvej položke vo fronte. WS a DV špecifikujú vstupné zariadenie, DVCLASS jeho triedu (napr. lokátor, výber). Dáta sa z fronty vyberú a prenesú do položky **aktuálna správa o udalosti**. Keď zistíme, z ktorej triedy zariadenia udalosť pochádza, vstupné dáta z aktuálnej správy o udalosti prečítame, napr. GET LOCATOR (*NT, POS*), kde *NT* je číslo normalizujúcej transformácie a *POS* je

pozícia. Vo funkcii GET treba špecifikovať len triedu zariadenia, ktorú poznáme z informácie od funkcie AWAIT EVENT. Sila režimu udalosti sa prejaví najmä vtedy, keď operátor riadi viacero zariadení alebo keď program robí medzi prijatím udalostí aj iné veci. Potrebnou funkciou na údržbu fronty je

FLUSH DEVICE EVENTS (*WS, DVCLASS, DV*),

ktorá vyberie z fronty všetky vstupy zo zariadenia, určeného parametrami *WS, DVCLASS* a *DV*.

17.6 Pracovné stanice

Aby sme nemuseli programovať pre indviduálne fyzické vstupné alebo výstupné zariadenia, pracujeme s ich abstrakciou, na logickej úrovni. Ústredným pojmom GKS je **pracovná stanica**, ktorá pre jednoduché prostredie obsahuje obrazovku a s ňou združené vstupné periférie. Pracovná stanica však nemusí mať ani vstup či výstup (napr. ploter či digitizér). Takéto stanice sa nazývajú vstupné (INPUT) resp. výstupné (OUTPUT) pracovné stanice. Stanica so vstupom aj výstupom sa nazýva pracovná stanica OUTIN. INPUT, OUTPUT a OUTIN sú **kategórie pracovných staníc**. Každé prostredie pre GKS potom možno popísať ako niekoľko pracovných staníc, ktoré používa jeden operátor a jeden aplikačný program. Preto možno mať niekoľko pracovných staníc, ktoré používame naraz. Definícia pracovných staníc závisí na rozsahu zariadení, prístupných v určitej inštalácii. Inštalátor preto v príručke danej inštalácie GKS označí typy, kategórie, miesto prístupných pracovných staníc a výskyt logických vstupných zariadení. Napr.

1 OUTIN	pamäťová obrazovka	pri sedadle	všetky
2 OUTIN	vektorová obrazovka	pri sedadle	všetky
3 OUTPUT	ploter	na chodbe	žiadne

Namiesto zohľadňovania individuálnych vstupných alebo výstupných zariadení zavádza GKS koncept pracovnej stanice, ktorá pre jednoduché prostredie obsahuje obrazovku a s ňou združené vstupné periférie, pripojené k počítaču. Pracovná stanica však nemusí mať ani vstup či výstup (napr. plotter či digitizér). Takéto stanice sa nazývajú vstupné (INPUT) resp. výstupné (OUTPUT). Stanica so vstupom aj výstupom sa nazýva pracovná stanica OUTIN. INPUT, OUTPUT a OUTIN sú kategórie pracovných staníc GKS. Ďalšie sú MI (METAFILE INPUT) a MO (METAFILE OUTPUT).

Stanicu pre náš program najprv identifikujeme, tj. popíšeme jej typ, spôsob pripojenia k počítaču a čas použitia volaním OPEN WORKSTATION (*WS, CONNECTID, WSTYP*), kde *WS* identifikuje stanicu, *CONNECTID* kanál a *WSTYP* typ stanice. Napr. OPEN WORKSTATION (1, 6, 3) spojí stanicu 1 cez kanál 6 s plotrom (typ 3) na chodbe. Takto stanicu **otvoríme**. Potom ju môžeme **aktivovať** funkciou ACTIVATE WORKSTATION (*WS*). Výstup sa zobrazuje na všetkých aktívnych staniciach OUTPUT a OUTIN, resp. ukladá do metasúboru cez stanicu MO. Počas behu programu možno stanice aktivovať a **deaktivovať** volaním DEACTIVATE WORKSTATION (*WS*) alebo vyčistiť použitím CLEAR WORKSTATION (*WS, FLAG*), kde *FLAG* je buď *ALWAYS* (vždy) alebo *CONDITIONALLY* (stanica sa vyčistí, ak sa na nej niečo

vykreslilo). Ak boli na stanici zapamätané segmenty, má táto funkcia vedľajší účinok, ktorý popíšeme neskôr.

17.6.1 Transformácia na stanicu

Obrázok zostavujeme v priestore NDC a zobrazujeme na aktívnych staniciach. **Transformáciou na stanicu** nazývame zobrazenie z okna stanice (NDC) na záber stanice (DC). Ktorú časť NDC a kde na stanici zobraziť určíme dvojicou funkcií:

SET WORKSTATION WINDOW (*WS, XNMIN, XNMAX, YNMIN, YNMAX*)

SET WORKSTATION VIEWPORT (*WS, XDMIN, XDMAX, YDMIN, YDMAX*)

Záber zadávame v súradniciach zariadenia (napr. počtom pixlov). Transformácia na stanicu je na rozdiel od normalizujúcej transformácie pre každú stanicu jediná a použije sa pre celý obrázok. Ak ju zmeníme, zmení sa príslušne aj celý obrázok. (Vedľajšie účinky popíšeme neskôr.) Transformácia na stanicu povoľuje iba izotropické zobrazenie. Ak sa pomer strán okna stanice nerovná pomeru strán záberu stanice, zobrazí sa celé okno na zariadení v správnom pomere strán, pričom sa použije obdĺžnik s ľavým dolným rohom rovnakým ako je ľavý dolný roh špecifikovaného záberu stanice s maximálnou možnou veľkosťou. Aby sa predišlo chybám, odporúča sa definovať okno a záber stanice vždy s rovnakým pomerom strán. Pre každú stanicu platí implicitná hodnota transformácie, ktorá zobrazí jednotkový štvorec NDC na najväčší možný štvorec na zariadení, pričom ľavý dolný roh priestoru NDC sa zobrazí na ľavý dolný roh obrazovky.

17.6.2 Reprezentácia

Zväzky atribútov na stanici nastavíme ďalej uvedenými funkciami:

SET POLYLINE REPRESENTATION (*WS, PLI, LT, LW, PLCI*),

kde *WS* je identifikátor stanice, *PLI* index lomenej čiary, *LT* typ čiary, *LW* je koeficient hrúbky čiary a *PLCI* je index farby lomenej čiary. Všeobecne platí, že ak stanica má menší rozsah funkčnosti pre niektorý atribút, použije ten (napr. hrúbku čiary), ktorú zariadenie pripúšťa a je najbližšie k špecifikovanej hodnote. Na farebnom displeji možno hodnoty v tabuľke farieb meniť. Každá položka tabuľky obsahuje index farby, intenzitu červenej (R), zelenej (G) a modrej (B) zložky požadovanej farby z uzavretého intervalu [0, 1]. Hodnotu farby pre daný index zmení funkcia

SET COLOUR REPRESENTATION (*WS, CI, RED, GREEN, BLUE*),

kde *CI* je index farby na stanici *WS*. Pre každú stanicu existuje jediná tabuľka farieb, čiže rovnaký index farby znamená na stanici rovnakú farbu.

SET POLYMARKER REPRESENTATION (*WS, PMI, MT, MS, PMCI*)

má parametre *PMI* index sledu značiek, *PMCI* index farby sledu značiek, *MT* je typ značky (štandardne znamená 1 bodku, 2 plus, 3 hviezdičku, 4 krúžok, 5 krížik). *MS* (*marker size*) je koeficient veľkosti značky.

Reprezentáciu výplňovej oblasti nastavíme funkciou:

SET FILL AREA REPRESENTATION (*WS, FAI, IS, SI, FACI*),

kde *FAI* je index výplňovej oblasti a *FACI* index farby výplňovej oblasti. *IS* znamená typ vnútra (*interior style*) a SI je index typu vnútra (*style index*), ktorý má význam len pri type vnútra šrafovanie (*HATCH*).

Reprezentáciu textu s indexom TI riadime funkciou:

SET TEXT REPRESENTATION (*WS, TI, TF, TP, CEF, CS, TCI*).

Druh písma *TF* (*text font*) reprezentuje požadovaný druh písma (románske, gotické, apod.). Presnosť *TP* nadobúda jednu z hodnôt *STRING, CHAR, STROKE*. Koeficient rozšírenia znaku *CEF* (*character expansion factor*) mení pôvodnú šírku znaku vzhľadom k výške. Napr. 2 znamená dvojnásobok šírky znaku. Implicitný rozostup znakov *CS* (*character spacing*) je nulový, lebo potrebný priestor okolo znaku je zahrnutý v tele znaku v každom druhu písma. Rozostup sa definuje v jednotkách veľkosti znakov, napr. 0.5 znamená, že sa medzi telá znakov vložia medzery o veľkosti 0.5 veľkosti znaku. Index farby textu *TCI* ukazuje do tabuľky farieb.

17.6.3 Ukladanie segmentov

Zatiaľ sme neuviedli, kde sa ukladajú segmenty. Konceptuálne na staniciach, aktívnych v čase vytvorenia segmentu. Dôsledná implementácia by mala ukladať segmenty centrálne, ale výkonnosť sa zvýši, ak sa segmenty zapamätajú na inteligentných staniciach s lokálnou pamäťou. Aj pri centrálnom ukladaní segmentov je vhodná predstava, že segmenty sú zapamätané na stanici. Segment *ID* na stanici *WS* možno zrušiť funkciou

DELETE SEGMENT FROM WORKSTATION (*WS, ID*).

Funkcia CLEAR WORKSTATION zruší všetky segmenty na špecifikovanej stanici. Funkcia REDRAW ALL SEGMENTS ON WORKSTATION sa používa na prekreslenie zobrazovacej plochy bez toho, aby sa zrušili segmenty. Uzavretím stanice sa zrušia všetky na nej zapamätané segmenty. Pri otvorení sa stanica inicializuje s prázdnou pamäťou segmentov. Ak sa segment zruší na každej stanici, tak je zrušený celkom (ako po funkcii DELETE SEGMENT).

17.6.4 Niektoré ďalšie vlastnosti pracovnej stanice

Niekedy je nevyhnutné **zdržanie zmien obrazu.** Ak sa ako stanica používa inteligentný displej s obnovovaním, očakáva operátor bezprostredné zmeny obrazu, no nie vždy. Napr. odoslanie výstupu po častiach na pamäťovú obrazovku spojenú s počítačom cez sieť dáva nižší výkon ako zhromaždenie informácie a odoslanie vcelku. Preto možno nastaviť funkciou SET DEFERRAL STATE niektorý režim zdržania:

1. ASAP (As Soon As Possible) - okamžitá zmena obrazu

2. BNIG (Before the Next Interaction Globally) - zmena musí nastať vtedy, keď na niektorej stanici vznikne požiadavka na vstup

3. BNIL (Before the Next Interaction Locally) - ako **2.** ale iba na tejto stanici

4. ASTI (At Some Time) - zmena nastane niekedy.

Ďalšou príčinou malej výkonnosti môže byť, že isté funkcie spôsobia **automatickú regeneráciu** celého obrazu (napr. zmena reprezentácie lomenej čiary zapríčiní, že všetky lomené čiary s daným indexom sa prekreslia) - na plotri sa teda musí vyčistiť zobrazovacia plocha a prekreslia sa všetky segmenty, čím možno stratiť časti obrazu,

nezapamätané v segmentoch. Automatickú regeneráciu môžeme preto oneskoriť - stanica pracuje v jednom z režimov *SUPRESSED* (potlačená) a *ALLOWED* (povolená). Režim zdržania a automatickej regenerácie nastavíme funkciou SET DEFERRAL STATE (*WS, DM, IGM*), kde *DM* je režim zdržania (*ASAP, BNIG, BNIL, ASTI*), *IGM* je režim automatickej regenerácie (*SUPRESSED, ALLOWED*) na stanici *WS*. Ak režim zdržania nie je *ASAP* alebo režim automatickej regenerácie je *SUPRESSED*, môžeme vyžiadať vykonanie zdržaných akcií pomocou UPDATE WORKSTATION (*WS, RF*), kde regeneračný príznak *RF* môže byť *PERFORM* (vykonaj) a *POSTPONE* (odlož). Automatická regenerácia sa však vykoná iba vtedy, keď je nevyhnutná a keď príznak regenerácie *RF* má hodnotu *PERFORM*. Naopak funkcia REDRAW ALL SEGMENTS ON WORKSTATION (*WS*) vykoná všetky zdržané akcie, vyčistí obrazovku a bezpodmienečne prekreslí všetky segmenty. Predpokladá sa, že obrazovky s obnovovaním pracujú zvyčajne v režime (*ASAP, ALLOWED*), lokálne pamäťové obrazovky v režime (*BNIL, SUPRESSED*) a off-line ploter v režime (*ASTI, SUPRESSED*).

Logické vstupné zariadenia GKS definuje virtuálne, takže aplikačný program dostáva vstup prostredníctvom zariadení LOCATOR, STROKE, VALUATOR, CHOICE, PICK a STRING. V inštalácii musí byť aspoň jedno zariadenie každej triedy, realizované na konkrétnych fyzických zariadeniach. V režime REQUEST je naraz aktívne iba jedno zariadenie a netreba ho identifikovať. Ak však inštalácia podporuje všetky vstupné režimy (REQUEST, SAMPLE a EVENT), treba so vstupnou hodnotou aj identifikovať zariadenie, lebo naraz ich môže byť aktívnych viac. Logické vstupné zariadenia napr. na pamäťovej obrazovke s klávesnicou a potenciometrom na riadenie nitkového kríža sa potom implementujú takto: LOCATOR: aktivácia lokátora vykreslí nitkový kríž, ak už nie je vykreslený. Hodnotu lokátora odosiela operátor klávesou L (tj. L znamená spúšť). STROKE: aktivácia sledu polôh ako lokátor. Jednotlivá pozícia sa odosiela klávesou S a sled polôh ukončuje stlačenie F, apod. Samozrejme, na bohatšom fyzickom zariadení je implementácia logických vstupných zariadení jednoduchšia.

17.7 Ďalšia funkčnosť GKS

Inicializáciu GKS žiadame funkciou OPEN GKS (*EF, SU*), ktorú možno počas práce s GKS zavolať len raz. *EF* (*error file*) zadáva meno súboru na návrat chybových informácií aplikačnému programu (spracovanie chýb popíšeme neskôr). Parameter *SU* (*storage used*) slúži na alokovanie pamäti. GKS inicializuje mnoho premenných na ich implicitné hodnoty, z ktorých časť závisí od implementácie (napr. maximálny počet súčasne otvorených staníc) a ostatné definuje pre všetky implementácie norma. Napr. aktuálny POLYLINE INDEX sa vždy nastaví na 1 (tak ako ostatné podobné indexy), číslo aktuálnej normalizujúcej transformácie na 0, všetky normalizujúce transformácie sa inicializujú na okno i záber ako jednotkové štvorce. Po skončení práce GKS **uzavrieme** funkciou CLOSE GKS. Vtedy sa vykonajú niektoré implementáciou požadované vnútorné funkcie a uzavrie sa chybový súbor.

Inicializácia vstupu (náznak, echo a spúšť). Logické vstupné zariadenie sa inicializujú zadaním počiatočnej hodnoty, typu náznaku a echa spolu s obdĺžnikovou oblasťou echa.

Ďalšiu funkčnosť grafického systému predstavujú **operačné stavy.** GKS je vždy práve v jednom z piatich operačných stavov, ktoré uľahčujú najmä **spracovanie chýb**.

Pred otvorením je GKS v stave **GKS CLOSED**. V každom inom stave možno volať len časť funkcií. Hlavné stavy sú:

1. **GKS OPEN**
2. **WORKSTATION OPEN**: otvorená aspoň jedna stanica
3. **WORKSTATION ACTIVE**: aktívna aspoň jedna stanica
4. **SEGMENT OPEN**: otvorený segment

Pri zmene stavu možno prechádzať len do niektorého zo susedných stavov. Preto napr. nemožno otvoriť segment, kým nie je aktívna aspoň jedna stanica. Norma presne stanovuje, v ktorom stave možno ktoré funkcie volať a ktoré funkcie a ako menia stav.

Tabuľky popisu a stavové zoznamy GKS. Tabuľka popisu závisí od implementácie a popisuje GKS resp. stanicu pred začatím práce. Stavový zoznam GKS resp. stanice obsahuje informácie o aktuálnom stave za behu aplikácie. Aplikačný program všetky tieto údaje smie iba čítať prostredníctvom zisťovacích funkcií. Hlavné tabuľky popisu a stavové zoznamy sú: **1. Operačný stav**: jediná hodnota, udáva stav GKS. **2. Tabuľka popisu GKS**: popisuje danú implementáciu. **3. Stavový zoznam GKS**: informácie o aktuálnych atribútoch, normalizujúcich transformáciách, otvorenom segmente, vstupnej fronte, ai. **4. Tabuľka popisu stanice**: charakteristiky stanice. **5. Stavový zoznam stanice**: informácie o otvorenej stanici.

Zisťovacie funkcie umožňujú programu čítať informácie z tabuliek popisu a stavových zoznamov. Používame ich napr. **1.** aby sa na určitej stanici dosiahol presný výsledok, napr. či možno na stanici kresliť v presnej mierke, **2.** aby sme aplikáciu prispôsobili špeciálnemu prostrediu, napr. aké sú maximálne rozmery displeja, **3.** aby sme mohli vyvíjať knižničné funkcie, napr. zistiť na začiatku funkcie atribúty, ktoré funkcia môže zmeniť a pri návrate z funkcie ich obnoviť, **4.** aby sa program zotavil z chyby.

Typická zisťovacia funkcia má tvar: INQUIRE NAME OF OPEN SEGMENT (*IND, SEGNAME*). Pri volaní zisťovacích funkcií **nemôže nastať chyba**, a preto možno po vzniku chyby bezpečne zisťovať jej príčinu. Každá zisťovacia funkcia má výstupný parameter *IND* na indikáciu, či sú ďalšie vrátené informácie platné. Nulový *IND* tu znamená, že parameter *SEGNAME* obsahuje meno otvoreného segmentu, kým nenulový *IND* oznamuje chybu, napr. že nie je otvorený segment a jeho meno neexistuje.

Spracovanie chýb. GKS má dobre definovaný **súbor chýb**, ktoré možno oznámiť aplikačnému programu prostredníctvom záznamu do chybového súboru, ktorý sa špecifikuje v čase otvorenia GKS. Typický záznam v ňom obsahuje: číslo chyby a funkciu GKS, počas práce ktorej sa chyba zistila. Činnosť po rozpoznaní chyby závisí od programu. Ak tento nepožaduje nejakú špeciálnu akciu, vyvolá GKS procedúru na spracovanie chýb ERROR HANDLING (*NMBR, FCTID, EF*), ktorá závisí od inštalácie a má parametre číslo chyby *NMBR*, meno funkcie *FCTID* a meno chybového súboru *EF*. Táto procedúra zaznamená chybu do chybového súboru volaním procedúry ERROR LOGGING (*NMBR, FCTID, EF*) a vráti riadenie do funkcie, v ktorej sa chyba zistila. Potom buď sa dá obnoviť stav GKS pred vyvolaním tejto funkcie alebo nie. V prvom prípade bude stav GKS, akoby sa funkcia nevyvolala. Inak následky nemožno predvídať.

Tento trocha ťažkopádny spôsob spracovania chýb sa v GKS prijal preto, aby dovolil aplikačnému programu vlastné spracovanie chýb. Treba poznamenať, že aj pre užívateľské spracovanie chýb je pred návratom záväzné volanie procedúry ERROR

LOGGING. V procedúre na spracovanie chýb možno volať iba nasledujúce funkcie GKS: **1.** ERROR LOGGING, **2.** zisťovacie funkcie, **3.** EMERGENCY CLOSE GKS. Posledná funkcia sa pridá pri takej chybe, z ktorej systém nie je schopný sa zotaviť a treba podľa možností zachrániť čo najviac grafických informácií (segmentov). Pre niektoré triedy chýb túto funkciu volá priamo GKS.

Úrovne GKS. Všestranný grafický systém vhodný pre široké spektrum 2D aplikácií môže znamenať pre jednoduché aplikácie aj mnoho nadbytočného. Použitie iba časti schopností GKS, dovoľuje mechanizmus **úrovní** (*level*). GKS má 9 platných úrovní, definovaných tromi rozličnými voľbami na dvoch osiach, ktoré možno približne nazvať **vstup** a **ostatné funkcie**: **a.** bez vstupných funkcií, **b.** vstup len v režime vyžiadania (REQUEST), **c.** vstupy vo všetkých režimoch (REQUEST, SAMPLE a EVENT). Výber funkčnosti na druhej osi sa definuje takto: **0.** Minimálny výstup - možno používať všetky výstupné prvky, ale nemožno meniť význam ich indexov, musia sa používať implicitné hodnoty, nastavené inštaláciou, v danom okamihu je povolená iba jedna výstupná stanica a musí byť aspoň jedna nastaviteľná normalizujúca transformácia. **1.** Ako 0., ale navyše hlavne schopnosť špecifikovať indexy reprezentácie výstupných prvkov, mať súčasne viac aktívnych výstupných staníc, segmentáciu a viacero normalizujúcich transformácií. **2.** Ako 1., ale navyše schopnosť na stanici nezávislého ukladania segmentov. Najjednoduchšia implementácia GKS má preto úroveň **0a** - iba výstup na jedinú stanicu v danom okamihu.

Na stanici nezávislé ukladanie segmentov (WISS). Dosiaľ popísané ukladanie segmentov nazývame **na stanici závislé ukladanie segmentov**, čo spôsobuje vysokú spotrebu pamäti i potrebu vykonávať všetky zmeny segmentov na staniciach. Na presúvanie segmentov zo stanice na stanicu slúži mechanizmus **na stanici nezávislého ukladania segmentov** (*Workstation Independent Segment Storage,* WISS). Implementácia GKS môže poskytnúť najviac jednu takúto pamäť segmentov. Aplikačný program môže segmenty z WISS presunúť na ľubovoľnú pracovnú stanicu. Segmenty uložené vo WISS majú presne tie isté atribúty ako segmenty vo WDSS a manipulujú s nimi tie isté funkcie GKS. WISS je v GKS jednou pracovnou stanicou, na ktorej sa funkciou CREATE SEGMENT segmenty vytvárajú, ak bola aktivizovaná. Takisto možno z nej segmenty rušiť pomocou DELETE SEGMENT FROM WORKSTATION. Ďalšie užitočné funkcie na prácu s WISS sú: ASSOCIATE SEGMENT WITH WORKSTATION, COPY SEGMENT TO WORKSTATION a INSERT SEGMENT. ASSOCIATE kopíruje segment z WISS na stanicu. To isté spraví aj COPY. Líšia sa tým, že výsledkom ASSOCIATE je segment na stanici, kým výsledkom COPY sú na stanici iba grafické výstupné prvky segmentu. Obe funkcie nemožno volať pre otvorený segment. Funkcia INSERT SEGMENT(*ID, MAT*) uplatní na segment *ID* (ktorý musí existovať vo WISS) po transformácii segmentu transformáciu danú maticou *MAT* a okopíruje segment *ID* na všetky aktívne stanice.

Po uzavretí GKS sa stratia všetky informácie, uložené v segmentoch. Na dlhodobú úschovu grafických informácií slúži **GKS metasúbor** (GKSM). Formát a funkčnosť metasúborov všeobecne stanovuje norma **CGM** [IS8632], popísaná v kapitole 18. GKSM sa líši od CGM. GKS považuje metasúbor za zvláštnu kategóriu pracovnej stanice, možno mať súčasne aktívnych viac staníc typu výstup do alebo vstup z metasúboru. Ak je aktívna stanica typu **výstup do metasúboru**, ukladajú sa výstupné funkcie, nastavenie atribútov a segmenty, geometrické dáta sa ukladajú v súradniciach NDC a

mimografické dáta možno zapísať zvláštnou funkciou WRITE ITEM TO GKSM. GKSM je postupnosťou trojčasťových položiek: typ položky, dĺžka dátového záznamu položky, dátový záznam položky. Pri volaní každej funkcie GKS, ktorá sa ukladá do metasúboru, sa *generuje* aspoň jedna položka GKSM.

Vstup z metasúboru sa nazýva **interpretácia**. GKS má na získanie vstupu z metasúboru tri funkcie: GET ITEM FROM GKSM, READ ITEM FROM GKSM a INTERPRET ITEM. Interpretácia položky môže spôsobiť implicitnú regeneráciu obrazu. Stanicu vstup z metasúboru netreba aktivizovať. Keby sme do GKSM zahrnuli aj mimografické dáta, mal by ich spracovať náš program a nepripustiť ich grafickú interpretáciu funkciou INTERPRET ITEM.

Vlastnosti zariadenia nedostupné cez štandardné funkcie GKS ako zazvonenie zvončeka môžeme riadiť **rozširujúcou funkciou** ESCAPE.

17.8 GKS-94

Takmer všetko z doteraz uvedeného platí aj pre normu PHIGS. Z hľadiska výkladu normy PHIGS sme GKS použili ako nástroj na definovanie a objasnenie pojmov v jednoduchšom 2D kontexte.

Na záver tejto kapitoly treba skonštatovať, že norma GKS priniesla v roku 1985 do počítačovej grafiky pojmy a spôsoby uvažovania, ktoré zostanú klasickými. Jej nová verzia GKS-94 poňatie grafického systému v mnohom zmodernizovala, najmä podľa nového referenčného modelu grafických noriem **CGRM** (*Computer Graphics Reference Model*), z ktorého v kapitole 20 priblížime prostredia.

Norma GKS-94, nazývaná aj NewGKS, označuje revidovanú podobu 2D funkčnej normy GKS. Vyšla v roku 1994 ako prvá norma 2. generácie, na báze nového referenčného modelu z roku 1992. Z inovácií treba spomenúť zovšeobecnenie výstupu na skupiny výstupných prvkov, pridanie splajnových kriviek, zovšeobecnenie znaku na glyf, rozlíšenie pojmu logického obrázku a realizovaného obrázku (*reified picture*), pridanie štvrtého súradnicového systému na popis logického obrázku, kontrolný "playback" (*audit trail*), kontrolné zobrazenie výstupu bez vplyvu negeometrických atribútov (*backdrop*).

Cvičenia.

1. Navrhnite implementáciu funkcií pre prácu so segmentami.
2. Implementujte grafické výstupné prvky GKS.
3. Simulujte niektorým logickým vstupným zariadením všetky ostatné.
4. Napíšte v GKS program na interaktívne kreslenie.
5. Realizujte v GKS jednoduchý oknový systém.
6. Špecifikujte funkcie, zavedené v GKS-94.

18

Kódovanie grafickej informácie

18.1 Kódovanie informácie

Touto časťou textu všeobecnejšie dopĺňame problematiku **metasúboru** v súvise s grafickým systémom, čoho sme sa pri výklade GKS dotkli len letmo a čo celkom vynecháme v kapitole 19 o norme PHIGS. Mechanizmus metasúboru má však každý grafický systém i každá kvalitnejšia aplikácia. Pod **kódovaním** obrázku alebo obrazu tu rozumieme jeho reprezentáciu konkrétnym formátom organizácie dát kvôli uloženiu, spracovaniu alebo prenosu. Pri spracovaní grafickej informácie ide často o obrovské objemy dát, a preto sa s kódovaním neraz spája **kompresia**, zníženie objemu pri zachovaní pôvodného obsahu (bezstratová kompresia) alebo kompresia s určitou stratou (*lossy compression*). Medzinárodné normy na kódovanie grafickej informácie vyvíjajú pracovné skupiny podkomisie **ISO/IEC JTC1/SC29 CODING OF AUDIO, PICTURE, MULTIMEDIA AND HYPERMEDIA INFORMATION.**

Na prenos a ukladanie grafických dát sa používa mnoho formátov. Možno ich rozlišovať podľa dimenzie. Pre 2D sú to **CGM, JBIG, JPEG, MPEG, GIF, TIFF, TGA, HPGL, PIC, PCX, WMF, MPS, FIF, BMP, CDR, DIB, IFF, IMG, DRW, WPG, EPS,** ai. Na kódovanie 3D grafickej informácie sa často používajú formáty **RIB, DXF, NFF, OFF, IGES,** ai. Často sa používajú aj všeobecné dátové kompresie **ZIP, ARJ**, a pod., no problematike kompresií sa tu nebudeme vôbec venovať. Treba rozlišovať kódovanie **neštrukturovaného obrazu** (*image*), napr. JPEG, GIF, PCX, a **štrukturovaného obrázku** (*picture*), napr. CGM, EPS, WMF, kde sa ukladá aj štruktúra obrázku, jeho prvky a logika výstavby obrázku. Štrukturovanému obrázku sa niekedy hovorí, že je vo **vektorovej forme,** pre neštrukturované sa používajú pojmy **bitová mapa, bitmapa, pixmapa,** a pod. Štrukturovanosť (viac informácie o obrázku) umožňuje úsporu pamäti ale aj presné škálovanie.

Informácie možno rozdeliť na alfanumerické, grafické, hudobné a iné (napr. genetické). Základnou normou pre kódovanie alfanumerickej informácie je 15-stránkový dokument **IS 646** - *7-bit coded character set for information interchange*, ISO 1983, ktorá stanovuje základnú tabuľku 7-bitového kódovania znakov, napr. (binárne) 100 0001 znamená A, 011 0001 znamená 1, a 111 1111 znamená DEL. Celá tabuľka popisuje konvenciu známu ako **ASCII kód** (128 znakov). ASCII je skracuje názov pôvodnej americkej normy *American Standard Code for Information Interchange*. Kódovanie takmer všetkých týchto znakov je záväzné minimálne v 23 krajinách, vrátane SR. Norma **IS 2022** - *ISO 7- bit and 8-bit coded character sets - code extension techniques*, ISO

1983 rozširuje kódovanie 7-bitové na 8-bitové a uvádza všeobecný mechanizmus kódovania (rozširujúce [ESC] postupnosti) aj pre rozsiahlejšie súbory znakov, napr. pre kódovanie japonskej abecedy. S rozširovaním 16-bitových dátových zberníc prichádza aj 16-bitový kód **UNICODE**, v ktorom by každý globálne používaný **glyf** (písmeno, číslica, značka, symbol, nota, atď.) mal mať svoju bitovú kombináciu, svoj kód. Normované kódovanie znakov si často neuvedomujeme. Uvedomenie nastáva napr. s príchodom počítačových sietí, videotexu a ďalších z hľadiska kódovania medzinárodne dohodnutých spôsobov resp. kontextov, ktoré sú popísané v množstve ďalších noriem. Z nich len spomeňme normy ISO na kódovanie **1. monochromatických obrazov** (*bilevel images*) **JBIG**, **2. statických obrázkov** (*still pictures*) **JPEG**, **3. pohyblivých obrázkov** (*moving pictures*, film, animácia) **MPEG**, **4. multimediálnej informácie** (video, audio, ai.) **HyTime** a **MHEG**, **5. hyperdokumentov HyperODA**, **6. hudobnej informácie** (audio) **MIDI**. (Skratky objasníme nižšie.) Okrem medzinárodných noriem sa používa množstvo grafických formátov resp. kompresií, ktoré sú de facto normami pre veľké komunity užívateľov. Na danej ploche možno na celú rozsiahlu oblasť medzinárodnej normotvorby v kódovaní informácie len poukázať. Pri ďalšom štúdiu kódovania informácie by bolo dobré mať na zreteli 3 skutočnosti. **1.** Silné firmy (napr. Tektronix, AutoDesk (výrobca známeho systému AutoCAD)) zotrvávajú ***v určitej opozícii voči medzinárodným normám***, čím do istej miery chránia svoj trh. **2.** Význam medzinárodných konvencií však vzrastá s poznaním, že ideálnym partnerom v mnohých aplikáciách sa pre človeka stáva **multimediálny systém,** tj. systém, integrujúci spracovanie (analógovo i číslicovo reprezentovanej) textovej, obrazovej, hudobnej i filmovej (video) informácie. Vývoj takýchto systémov sa opätovne stretá - na úrovni kódovania informácie - s problémom **portability**, ktorého systémovým riešením bude prijatie noriem. **3.** Pre ČSFR boli a pre SR sú záväzné normy ČSN, odvodené od noriem ISO. To bude platiť aj pre normu CGM.

18.2 Prehľad normy CGM

Medzinárodná norma **CGM** (*Computer Graphics Metafile*) [IS8632] definuje funkčnosť a kódovanie metasúboru na ukladanie a prenos grafických 2D informácií. CGM nie je normou grafického systému, ale spôsobu ukladania grafických dát, čo s grafickými systémami úzko súvisí. Význam CGM vzrastá v súvislosti s rozvojom počítačových sietí, kde sa CGM popri **EPS** (*Encapsulated PostScript*) ai. stáva jedným z formátov pre také moderné komunikačné projekty ako je **WWW** (*World-Wide Web*), **MIME** (*Multipurpose Internet Mail Extensions*), apod.

Význam CGM spočíva v tom, že poskytuje formát súboru vhodného na uloženie a znovuzískanie grafických informácií. Formát pozostáva z usporiadanej množiny prvkov, ktorými možno popísať obrázky spôsobom prijateľným medzi grafickými systémami s rôznou architektúrou a zariadeniami s rôznymi schopnosťami a využitím. Norma podporuje úplny sekvenčný prístup a umožňuje aj nesekvenčný prístup k dátam. Konkrétne generovanie a interpretácia metasúboru sa v norme nepopisuje, hoci výsledok interpretácie áno. Základný súbor prvkov CGM zahŕňa aj možnosť pridať aplikačné dáta, ktoré nemajú grafický význam a výsledok ich interpretácie norma nestanovuje. Norma má tri diely - **1. Funkčná špecifikácia**, **2. Kódovanie znakmi** (*Character Encoding*) a **3. Binárne kódovanie** (*Binary Encoding*). V staršej verzii CGM bolo aj **textové kódovanie**, ktoré vyjadrovalo čitateľné a editovateľné metasúbory - za cenu enormného rozsahu.

Zdôraznime rozdiel medzi formálnou špecifikáciou a kódovaním - funkčnosť metasúboru sa **oddeľuje** od špecifikácie každého konkrétneho formátu. Norma pripúšťa aj súkromné kódovanie. Každé z kódovaní dovoľuje reprezentovať úplnu funkčnosť metasúboru. Norma CGM je založená na modeli grafického systému podľa GKS; CGM však možno využiť aj mimo GKS, napr. v dokumente podľa normy **SGML** (IS 8879 *Standard Generalized Markup Language*, jazyk na reprezentáciu textových dokumentov).

18.3 Funkčná špecifikácia CGM

Cieľom CGM je poskytnúť mechanizmus na popis, ukladanie a komunikáciu grafických informácií od zariadenia nezávislým spôsobom. Preto definuje syntax a sémantiku súboru prvkov CGM, ktorých je 9 tried. CGM je súbor prvkov z týchto tried:

1. **oddeľovače**: oddeľujú význačné štruktúry v CGM
2. **popisovače metasúboru**: popisujú funkčný obsah CGM
3. **popisovače obrázku**: popisujú atribúty vzhľadu obrázkov
4. **riadiace prvky**: modifikujú hranice a súradnicové systémy
5. **grafické výstupné prvky**: popisujú obrázok
6. **atribútové prvky**: popisujú vzhľad grafických prvkov
7. **rozširujúci prvok**: popisuje nenormalizované prvky
8. **externé prvky**: pre mimografické informácie
9. **segmentové prvky**: popisujú segmenty a ich atribúty

Ilustrácia štruktúry CGM

```
BEGIN METAFILE meno metasúboru
    (popisovače metasúboru)
BEGIN PICTURE meno obrázka
    (popisovače obrázka)
BEGIN PICTURE BODY
    (riadiace prvky)
    (grafické výstupné prvky)
    (atribútové prvky)
    (externé prvky)
END PICTURE
...
END METAFILE
```

Prvky BEGIN METAFILE a END METAFILE sa v CGM vyskytnú práve raz, kým ostatné prvky podľa potreby. Aby sa grafický obsah CGM zobrazil na zariadení, musí sa **interpretovať**. Interpretáciu riadia popisovače a riadiace prvky.

18.4 Kódovanie CGM znakmi a binárne

V pôvodnej verzii CGM (1987) bolo kódovanie trojaké. **Kódovanie znakmi** na minimalizáciu veľkosti metasúboru, **binárne kódovanie** na minimalizáciu času interpretácie a **kódovanie textom** na podporu čitateľnosti metasúboru.

Kódovanie CGM znakmi poskytuje reprezentáciu syntaxe metasúboru pre situácie, v ktorých treba minimalizovať veľkosť metasúboru. Každý prvok metasúboru má jednoduchú regulárnu syntax a v budúcich vydaniach normy možno pridávať nové prvky i nové operandy. Takisto každý operand má jednoduchú regulárnu syntax. Operandy sú rozličnej dĺžky, čo dovoľuje malé hodnoty reprezentovať čo najmenším počtom bitov. Kódovanie zodpovedá pravidlám pre rozšírenie kódu podľa normy IS 2022 v kategórii úplneho kódovacieho systému. Pre každý prvok CGM je tu špecifikované kódovanie. Základným pojmom v tomto kódovaní je 7- a 8-bitová **tabuľka kódu**. V 7-bitovom prostredí má tabuľka kódu 8 stĺpcov a 16 riadkov. Znak má 7 bitov, označených b1 až b7 od najnižšieho rádu k najvyššiemu. Bity b7, b6 a b5 adresujú stĺpec, bity b4, b3, b2 a b1 riadok. Používa sa notácia **stĺpec/riadok.** Napr. 1/11 znamená znak v stĺpci 1 a riadku 11, binárne 0011011. K tabuľke kódu možno označiť súbor znakov, ktoré majú byť reprezentované prvkami tabuľky. Týchto súborov môže byť viac a platný súbor znakov možno tzv. vyvolať. Ak je požadovaný súbor znakov rozsiahly, treba ho reprezentovať viacerými tabuľkami kódu resp. jeden znak viacerými bajtami. Takto možno reprezentovať hoci japonskú obrázkovú abecedu s jej 6802 grafickými znakmi.

Do prostredia kódu CGM možno vstúpiť z iného, napr. z kódu IS 2022 postupnosťou ESC 2/5 F, kde znak F určia pravidlá normy 2022. Takisto sa možno do prostredia predchádzajúceho kódu z CGM vrátiť: ESC 2/5 4/0. Vtedy sa obnoví aj predchádzajúce označenie a vyvolanie znakového súboru ku tabuľke kódu. Konkrétne kódy prvkov možno nájsť v norme; napr. BEGIN METAFILE má kód 3/0 2/0 a TEXT 2/3, súradnice sa ukladajú v tzv. základnom formáte. Označme X nevyužitý bit, e pokračovací bit, b dátový bit. Potom súradnicu zapíšeme v tvare X 1 e s b b b b (prvý bajt so znamienkom s) a X 1 e b b b b b (ďalšie bajty). Bit b7 je príznak parametra, keby bol nulový, išlo by o kódy operandov.

Binárne kódovanie metasúboru optimalizuje rýchlosť interpretácie a generovania. Tento spôsob kódovania je blízky reprezentácii údajov v rôznych počítačových systémoch (napr. kódovanie čísel sa zhoduje s bežným kódovaním v norme IEEE P754, ktoré používajú mnohé počítačové systémy). Základnou jednotkou organizácie binárne kódovaného metasúboru je osmica - 8 bitov. Dve osmice tvoria logické slovo metasúboru. Každý prvok sa musí začínať na začiatku slova, kvôli rýchlosti spracovania.

18.5 Niektoré ďalšie kódovania

Pri tvorbe názvov noriem ISO pre kódovanie sa zaužívali pre niektoré normy skratky utvorené z názvov komisií, ktoré ich vytvárali, nie z názvu normy.

JBIG (*Progressive Bi/level image compression standard*) je norma ISO na kódovanie monochromatického (napr. čiernobieleho) obrazu.

JPEG (*Joint Photographic Experts Group*, ISO/IEC 10918 *Digital compression and coding of continuous-tone still image*) je norma ISO a UTI (predtým CCITT) na ukladanie jednotlivého statického rastrového obrazu, kde možno stanoviť vzťah kompresného pomeru a kvality obrazu. JPEG rozoznáva 4 spôsoby kódovania, väčšina z nich využíva DCT (diskrétnu kosínusovú transformáciu). JPEG bol implementovaný hardverovo i softverovo.

MPEG (*Moving Picture Experts Group*) je norma ISO a IEC na komprimované ukladanie a prenos videosekvencií. MPEG má tri časti: MPEG-Video, MPEG-Audio a MPEG-System. Pre video sa komprimujú postupnosti rastrových obrázkov, pre audio sa komprimuje súbor tónov a MPEG-Systém integruje komprimované obrazové a zvukové dáta. MPEG kombinuje viaceré kompresné metódy a dosahuje kódovanie 1.5 Mbit/s v kvalite VHS s kompresným pomerom až 100:1. V súčasnosti sa na trh dostávajú prvé dekompresné karty MPEG, čím sa sprístupňuje využitie PC ako videoprehrávača. Vyššie typy MPEG II a MPEG III podporujú prenos dát 10Mbit/s a 40 MBit/s a sú ešte vo vývoji. MPEG IV na nízke rýchlosti prenosu do 64 Kbit/s má byť základom pre videokonferencie a očakáva sa po roku 2000.

Spomeňme ďalej niektoré rozšírené 2D formáty.

GIF (*Graphics Interchange Format*) je príkladom de facto normy na ukladanie rastrových dát s použitím kompresie LZW (Lempel-Ziv-Welch), podobne zameraný je **PCX** (*CompuServe Graphics Metafile*), používaný napr. v programe PaintBrush, a **BMP** (Microsoft Windows bitová mapa).

TIFF (*Tag Image File Format*) slúži najmä pre DTP aplikácie, má však mnoho obmien - s kompresiou a bez, obrazy podľa farieb: čiernobiele, v škále sivej, vo farebnom modeli RGB a s paletou.

EPS (*Encapsulated PostScript*) je **jazyk popisu stránok** (*page description language*), v mnohom príbuzný CGM, používaný najmä na riadenie laserových tlačiarní.

Vhodnosť výberu "jediného správneho" formátu sa relativizuje konvertormi, ako napr. Graphics Workshop.

V sieťovom prostredí Internet sú na transport obrazu vo WWW registrované 4 typy médií **CGM, GIF, JPEG** a **IEF** (*Image Exchange Format*).

Scénu definuje ISO ako **skutočné rozmiestnenie objektov.** Na kódovanie 3D grafickej informácie, tj. najmä objektov a charakteristík scény, slúžia oi. nasledujúce formáty:

IGES (*Initial Graphics Exchange Specification*) je od leta 1981 ANSI norma Y14.26M, široko používaná na prenos 2D a 3D databáz CAD/CAM.

OFF (*Object File Format*) je všeobecný a rozširovateľný formát na kódovanie 3D objektov, vyvinutý v Digital Equipment Corporation.

NFF (*Neutral File Format*) je navrhnutý ako minimálny jazyk na popis scén, tj. geometrie a základných charakteristík povrchu objektov, pozorovateľa a osvetlenia scény.

DWG (*DraWinG*), interný formát rozšíreného návrhového systému AutoCAD firmy AutoDesk, ktorý sa môže meniť podľa verzie AutoCAD, a preto nie je uverejnená jeho

dokumentácia. Z týchto dôvodov nie je vhodný na priame použitie, na to sa odporúča formát **DXF**.

Formát **RIB** (*RenderMan Interface Bytestream*) vyvinuli vo firme RenderMan.

Niektoré z týchto formátov neobsahujú už iba popis 3D objektov, ale aj ďalšiu technologickú informáciu, napr. odkazy do databázy súčiastok, dielov, apod. Takéto formáty už napomáhajú pri spracovaní v systémoch automatizovaného návrhu **CAD** (*Computer Aided Design*) a/alebo počítačovo riadenej výroby **CAM** (*Computer Aided Manufacture*). Už dnes je v oblasti návrhových systémov CAD/CAM známych niekoľko "cadovských" noriem na kódovanie informácie (PDES, SET, VDA-FS, PDDI, XBF, STEP), z ktorých viacero súvisí s už spomenutým formátom IGES.

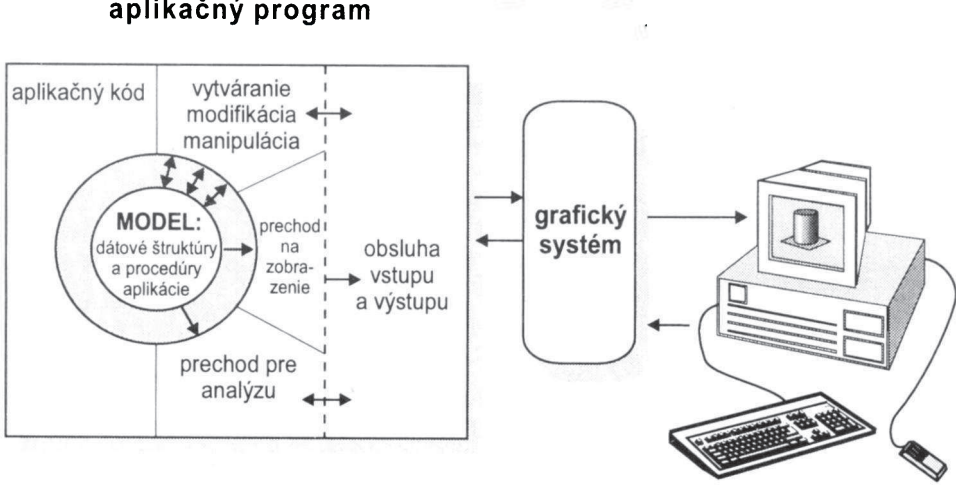
Kódovanie multimediálnej informácie normujú dve konvencie - MHEG a HyTime. **MHEG** (*Multimedia/Hypermedia Experts Group*) je návrh normy ISO (ISO/IEC CD 13522 *Coded Representation of Multimedia and Hypermedia Information Objects*). Normu **HyTime** (ISO/IEC 10744) prijali v roku 1992, jej oficiálny názov je *Hypermedia/Time-based Structuring Language*. Hypermédiá sú aplikáciou multimédií, multimediálne encyklopédie, umožňujúce pomocou počítača asociatívne vyhľadávanie informácií cez krížové odkazy, označením kľúčového slova, glyfu alebo obrazu. HyTime a MHEG sú dve vedúce normy na popis multimediálnych a hypermediálnych dokumentov, určených na ľudské vnímanie. Špeciálne na kódovanie hyperdokumentov slúži **HyperODA,** ktorá buduje na princípoch ISO/IEC DIS 8613 (*Open Document Architecture and Interchange Format*). S pomocou týchto noriem možno budovať prezentáciu, už aj vrátane interakcie (pozri projekt PREMO, kapitola 20).

19

PHIGS

19.1 Súvis PHIGS a GKS

Spomenuli sme niektoré grafické systémy (**Xlib**, 2D celočíselný GS v rámci X Window System, a **GKS**, normu 2D GS s reálnou aritmetikou). Funkčnosť 2D/3D grafického systému si vyžaduje ďalšie pojmy a úvahy. **PHIGS (Programmer's Hierarchical Interactive Graphics System**, IS 9592) normuje rozhranie medzi aplikačným programom a grafickým systémom a pokúša sa kombinovať modelujúci a zobrazovací systém. Cieľom PHIGS-u je poskytnúť na zariadení a aplikácii nezávislý spôsob tvorby a práce s viacúrovňovými modelmi v 2D a 3D priestore. Priemyselnými normami s funkčnosťou blízkou PHIGS sú OpenGL (GL skracuje Graphics Library) firmy Silicon Graphics a HOOPS (Ithaca Software). Základnú koncepciu grafického systému prebral PHIGS z GKS, a tak väčšinu pojmov pre PHIGS sme už zaviedli. Napr. otvorenie a uzavretie už poznáme aj s parametrami: OPEN PHIGS (*err_file, mem_size*) a CLOSE PHIGS ().

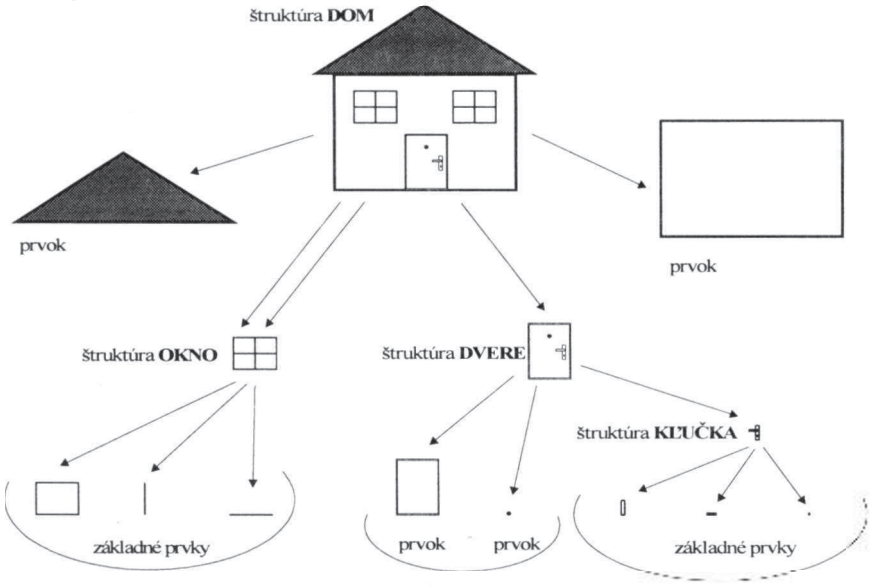


Obr. 19.1 Funkcie na čítanie a zápis do aplikačného modelu

Z troch častí normy (Funkčný popis, Formát archívneho súboru a Textové kódovanie archívneho súboru) informatívne predstavíme iba prvú časť a zmienime sa o norme PHIGS PLUS (niekedy aj PHIGS+), ktorú možno považovať za 4. časť PHIGS-u. PLUS tu nevyjadruje znamienko +, hoci sa tak niekedy píše, ale skracuje *Plus Lumiérre und Surfaces*. PHIGS publikovali v roku 1989 a PHIGS PLUS v roku 1992. Terminológia funkčnosti PHIGS sa všeobecne - i v učebniciach 3D grafiky - prijíma a na trhu sú implementácie PHIGS aj komerčne úspešné, dokonca, ako sme už spomenuli, DEC vyvinul hardverovú implementáciu. PHIGS má jazykové napojenia (*language bindings*) na niekoľko programovacích jazykoch, tu sa však môžeme venovať iba jeho funkčnosti, najmä podľa [ERTL93], [FOLE93] a [HEAR94].

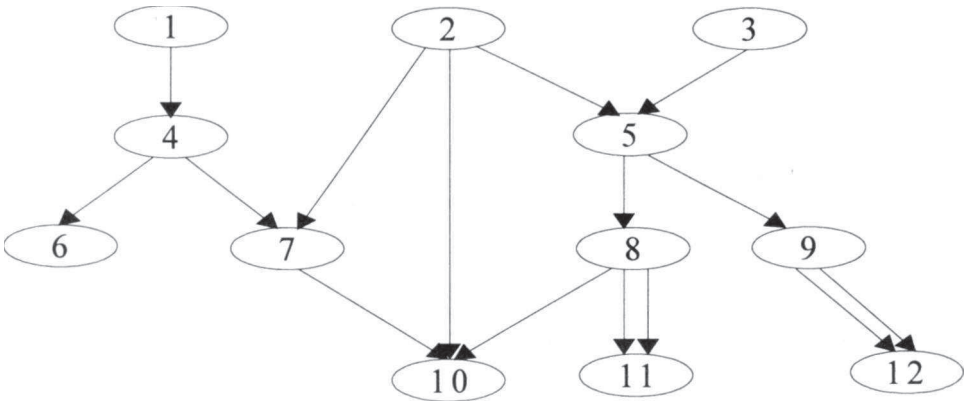
Základný pojem je **pracovná stanica**, tj. logické rozhranie, cez ktoré náš aplikačný program riadi fyzické zariadenia. Stanica má najviac jednu zobrazovaciu plochu a žiadne až niekoľko vstupných zariadení. Stanice delíme do 5 kategórií: INPUT, OUTPUT, OUTIN, METAFILE INPUT a METAFILE OUTPUT. Spojenie so stanicou otvoríme a uzavrieme ako v GKS: OPEN WORKSTATION (*ws_id, con_id, ws_type*) a CLOSE WORKSTATION (*ws_id*). Stanica má oproti GKS ďalšie funkcie, najmä 3D grafický vstup a výstup. Zdôraznime, že z hľadiska PHIGS je pracovná stanica **logické rozhranie** a nie hardver.

Vysoko nad rámec analogických pojmov segment a WISS z GKS sa v norme PHIGS rozvinuli **štruktúra** (*structure*) a **centralizovaná pamäť štruktúr** (*centralized structure store*, CSS) - hierarchická dátová štruktúra na ukladanie dát a manipuláciu s nimi. **Prvok štruktúry** (*structure element*) popíšeme neskôr, zatiaľ si ho možno predstavovať ako grafické výstupné prvky a ich atribúty v GKS. Prvky štruktúr ukladáme do **štruktúr** (*structures*), organizovaných ako acyklické orientované grafy nazývané **siete štruktúr** (*structure networks*). Práca so štruktúrami nezávisí na ich zobrazovaní. Štruktúry dovoľujú na rozdiel od segmentov v GKS viac ako dvojúrovňovú hierarchiu (nielen grafických) objektov.



Obr. 19.2 Viacúrovňová hierarchia

PHIGS dovoľuje to, čo GKS neumožní - vložiť štruktúru do štruktúry, napr. na obr. 19.3 do štruktúry 4 vložíme štruktúru 6. Vloženú štruktúru nazývame **potomok** a tú, do ktorej vkladáme, **rodič**. Obraz na stanici vytvára **prechádzanie štruktúrou** (*traversing structure*) a interpretácia jej prvkov. Mimografické dáta sa pri prechode štruktúrou odoslanou na stanicu ignorujú, kým geometrické dáta (súradnice) výstupných grafických prvkov štruktúry sa spracúvajú v dvoch etapách: na pracovnej stanici nezávislej a na pracovnej stanici závislej etape.



Obr. 19.3 Príklad siete štruktúr

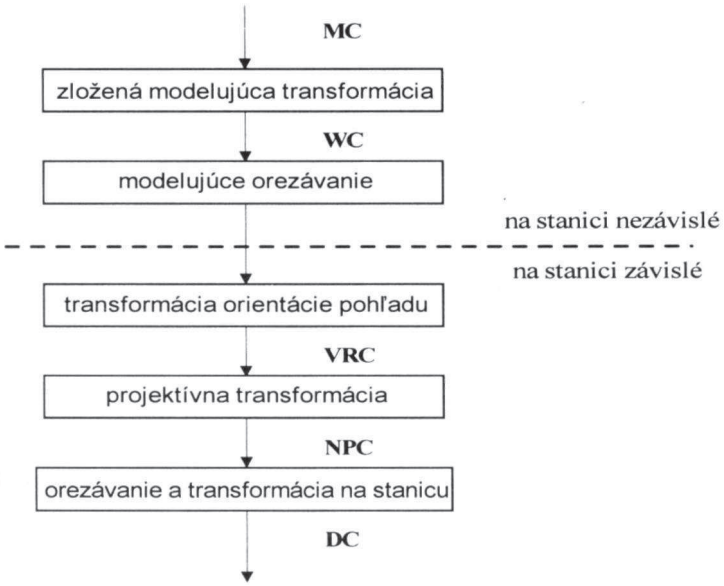
Grafický výstup buduje na dvoch skupinách základných prvkov nazývaných **výstupné grafické prvky** (*output primitives*) a **atribúty výstupných grafických prvkov** (*primitive attributes*). Výstupné grafické prvky sú abstrakciami základných akcií, ktoré môže zariadenie vykonať (vykreslí čiaru alebo reťazec znakov). Atribúty riadia vlastnosti výstupného prvku na zariadení (napr. typ čiary, farba a výška znaku). Organizujú sa jednotným spôsobom. Na vzhľad každého výstupného prvku sa aplikujú dve skupiny atribútov: atribúty výstupného prvku na stanici **nezávislé** a atribúty **závislé** na pracovnej stanici. Atribúty špecifikujeme modálne a sú napojené na výstupný prvok, keď sa vytvára počas prechodu štruktúrou. Atribútmi riadime všetky **geometrické** aspekty výstupných prvkov (výška znaku pre text, veľkosť vzorky pre výplňovú oblasť) aj **negeometrické** aspekty vzhľadu (farba čiary). Atribút ovplyvňuje vzhľad v tej štruktúre, v ktorej sa nachádza, a dedia ho aj štruktúry v hierarchii pod ňou. Zobrazenie (*rendering*) tej istej štruktúry na výstupe môže práve kvôli dedičnosti atribútov vyzerať rozdielne.

Vstup sa veľmi ponáša na vstup v GKS, na rozšírenia upozorníme. Vstupnou polohou z 3D aplikácie je prirodzene 3D bod.

PHIGS má **5 súradnicových systémov**, medzi nimi potrebné transformácie a orezávanie, z ktorých pozostáva **zobrazovací kanál** (*viewing pipeline*), čo rozoberieme podrobnejšie. Informatívne však preberieme výstup, vstup, pracovné stanice a rozšírenie PHIGS+. Nebudeme opakovať výklad metasúborov, zisťovacích funkcií, spracovania chýb a ďalšej funkčnosti, ktorú už poznáme z GKS.

19.2 Súradnicové systémy a transformácie

Každý z 5 súradnicových systémov PHIGS-u je **3D pravotočivý karteziánsky súradnicový systém**. Na zjednodušenie práce s 2D modelmi poskytuje PHIGS niektoré prvky štruktúr (pre výstupné grafické prvky a pre transformáciu a orezávanie) v dvoch verziách: 2D a 3D, napr. POLYLINE 3 a POLYLINE, číslo 3 v mene prvku znamená úplny 3D tvar prvku a verzia bez čísla 3 špecifikuje 2D tvar prvku, v ktorom aplikačný program dodá len *x*-ové a *y*-ové súradnice bodov (vektorov, atď.) a súradnice *z* sa považujú za nulové. Aj vtedy PHIGS spracuje všetky objekty v 3D súradniciach.



Obr. 19.4 Transformačný kanál PHIGS

19.2.1 Modelujúce transformácie a orezávanie

Obrázok možno skladať z oddelených častí, z ktorých každú môžeme definovať v rámci jej vlastného **modelujúceho súradnicového systému** (*modelling coordinate system*, MC). Vzájomnú polohu oddelených častí získame pomocou jediných **svetových súradníc** (*world coordinates, WC*), na ktoré všetky MC zobrazíme **zloženou modelujúcou transformáciou** (*composite modelling transformation*). Priestor svetových súradníc sa chápe ako na stanici nezávislý abstraktný priestor. Zložená modelujúca transformácia C sa skladá z lokálnej modelujúcej transformácie L (definovanej v aktuálnej štruktúre) a globálnej modelujúcej transformácie G (čo je zložená modelujúca transformácia na začiatku prechodu štruktúrou zdedená z rodičovskej štruktúry). V nasledujúcom výklade použijeme dve funkcie, ktoré podrobne vysvetlíme neskôr: POST odošle štruktúru na zobrazenie na stanici a EXECUTE "vloží" štruktúru do otvorenej štruktúry. Štruktúra na najvyššej úrovni odoslanej siete štruktúr má na začiatku (každého

prechodu sieťou štruktúr) G nastavenú na identickú transformáciu. L sa pre štruktúru takisto na začiatku (prechodu štruktúrou) nastaví na identickú transformáciu a modifikuje sa, keď sa pri prechode štruktúrou narazí na prvok SET LOCAL MODELING TRANSFORMATION 3 alebo SET LOCAL MODELING TRANSFORMATION. Zložená modelujúca transformácia sa vypočíta ako: C=G*L. Keď sa počas prechodu štruktúrou vyvolá podštruktúra, tak sa G aj L uschovajú a po návrate z prechodu podštruktúrou sa obe transformácie obnovia.

Prvky pre lokálne modelujúce transformácie vložíme do štruktúry funkciami: SET LOCAL TRANSFORMATION 3 (*matrix, type*) a SET LOCAL TRANSFORMATION (*matrix, type*), kde *matrix* je matica homogénnej transformácie (v prvom prípade 4*4, v druhom 3*3) a *type* určuje typ operácie s doteraz platnou lokálnou transformáciou. Označme LNEW ako výslednú lokálnu transformáciu, L pôvodnú lokálnu transformáciu a T transformáciu danú v *matrix*. Potom

LNEW = T pre *type* s hodnotou REPLACE

LNEW = L * T pre *type* s hodnotou PRECONCATENATE

LNEW = T * L pre *type* s hodnotou POSTCONCATENATE.

Vedľajší účinok tejto funkcie je samozrejme aj nastavenie zloženej modelujúcej transformácie na novú hodnotu CNEW = G * LNEW. Analogicky SET GLOBAL TRANSFORMATION 3 (*matrix*) a SET GLOBAL TRANSFORMATION (*matrix*) nastavia v stavovom zozname prechodu maticu globálnej transformácie na hodnoty podľa parametra *matrix*.

Príklad 19.1 Nastavenie transformácií pre štruktúry

```
/* Predpokladajme sieť štruktúr podľa obr. 19.3 a naviac, že štruktúry 4, 6, 7 obsahujú oi.
nasledujúce prvky pre nastavovanie transformácií a väzby na štruktúry v uvedenom
poradí: */

/* pre štruktúru 4 */
 SET LOCAL TRANSFORMATION 3 (L41, REPLACE)
 SET GLOBAL TRANSFORMATION 3 (G41)
 EXECUTE STRUCTURE (6)
 SET LOCAL TRANSFORMATION 3 (L42, POSTCONCATENATE)
 EXECUTE STRUCTURE (7)
/* pre štruktúru 6 */
 SET LOCAL TRANSFORMATION 3 (L61, REPLACE)
 SET GLOBAL TRANSFORMATION 3 (G61)
/* pre štruktúru 7 */
 SET LOCAL TRANSFORMATION 3 (L71, REPLACE)
 SET LOCAL TRANSFORMATION 3 (L72, PRECONCATENATE)
 EXECUTE STRUCTURE (10) /* prechod ďalšou podštruktúrou */
```

Aké budú hodnoty matíc zloženej, globálnej a lokálnej transformácie pre jednotlivé operácie pri prechode podsieťou štruktúry 4? Odpovede sú uvedené v nasledujúcej tabuľke, kde I znamená maticu identickej transformácie a operácie uvádzame len v skratkách.

Operácia	Lokálna transformácia	Globálna transformácia	Zložená transformácia	Poznámka
post(4)	I	I	I	(1)
local(L41,REP)	L41	I	L41	
global(G41)	L41	G41	G41*L41	
execute(6)	I	G41*L41	G41*L41	(2)
local(L61,REP)	L61	G41*L41	G41*L41*L61	
global(G61)	L61	G61	G61*L61	
návrat zo 6 do 4	L41	G41	G41*L41	(3)
local(L42,POST)	L42*L41	G41	G41*L42*L41	
execute(7)	I	G41*L42*L41	G41*L42*L41	(2)
local(L71,REP)	L71	G41*L42*L41	G41*L42*L41*L71	
local(L72,PRE)	L71*L72	G41*L42*L41	G41*L42*L41*L71*L72	
execute(10)	I	G41*L42*L41*L71*L72	G41*L42*L41*L71*L72	(2)
návrat z 10 do 7	L71*L72	G41*L42*L41	G41*L42*L41*L71*L72	(3)
návrat zo 7 do 4	L42*L41	G41	G41*L42*L41	(3)

Poznámky:
(1) Na začiatku prechodu sieťou štruktúr sa všetky modelujúce transformácie nastavia na identickú transformáciu. Treba si uvedomiť, že ak by sa aj v štruktúre 1 vyskytovali prvky na nastavenie modelujúcej transformácie, tieto by v tomto prípade nemali žiaden vplyv na globálnu a zloženú modelujúcu transformáciu, pretože prechod sieťou sa začína od štruktúry 4.
(2) Hodnoty matíc transformácie z predchádzajúceho riadku sa uložia, matica lokálnej transformácie sa nastaví na identickú, matica globálnej transformácie sa nastaví na hodnoty matice zloženej transformácie
(3) Pri návrate zo štruktúry sa obnovia uložené hodnoty.

Pomocou **modelujúceho orezávania** (*modelling clipping*) poskytuje PHIGS možnosť, aby sa z jednotlivých dielov obrázku transformovali ďalej len ich jednotlivé časti. Orezávanie sa robí v závislosti od toho, ako je nastavený **indikátor modelujúceho orezávania** (*modelling clipping indicator*) a vzhľadom na **modelujúci orezávací priestor** (*modelling clipping volume*). Indikátor modelujúceho orezávania nastavíme funkciou SET MODELLING CLIPPING INDICATOR (*mod_clip_ind*) na hodnoty CLIP alebo NOCLIP. Na určenie modelujúceho orezávacieho priestoru slúžia funkcie SET MODELLING CLIPPING VOLUME 3 (*operator, half-spaces*) a SET MODELLING CLIPPING VOLUME (*operator, half-spaces*).

Zoznam polpriestorov *half-spaces* má v prvom prípade polpriestor určený 3D bodom v MC a 3D normálovým vektorom v MC, v druhom prípade 2D bodom a 2D normálovým vektorom (ležiacimi v rovine MC z=0). Normálový vektor definuje normálu roviny určujúcej polpriestor, pričom normála ukazuje von z polpriestoru a rovina prechádza špecifikovaným bodom. Prienikom polpriestorov (transformovaných aktuálnou zloženou modelujúcou transformáciou) zoznamu *half-spaces* definujeme **orezávací priestor** S. Nech T a T' sú pôvodný a výsledný modelujúci orezávací priestor (transformované zloženou modelujúcou transformáciou). Závislosť T' od T a S určíme parametrom *operator* ako **nahradenie** T' = S alebo **prienik** T' = T$\cap$ S. Aplikačný program môže použiť aj iné hodnoty operátora, ich význam však nedefinuje PHIGS ale len implementácia.

Príklad 19.2: Modelujúci orezávací priestor

Úloha: nastavme modelujúci orezávací priestor v rovine na rovnostranný trojuholník s jednotkovou stranou.

Označme P1 = (0.0, 0.0) N1 = (0.0, -1.0)
P2 = (cos 60°, sin 60°) N2 = (-sin 60°, cos 60°)
P3 = (1.0, 0.0) N3 = (sin 120°, -cos 120°)

Modelujúci orezávací priestor nastavíme volaním SET MODELLING CLIPPING VOLUME (REPLACE,{3,(P1,N1),(P2,N2), (P3,N3)}).

Funkciou RESTORE MODELLING CLIPPING VOLUME () obnovíme pôvodný modelujúci orezávací priestor na hodnoty nastavené na začiatku prechodu štruktúrou, t.j. zdedené z rodičovskej štruktúry.

GKS i PHIGS uľahčujú výpočet transformácie, ak ju vieme zložiť z posunutia, škálovania alebo otočenia. TRANSLATE 3 (*transl_vector, err_ind, transf_matrix*) a TRANSLATE (*transl_vector, err_ind, transf_matrix*) vrátia **maticu posunutia** určeného zadaným vektorom. SCALE 3 (*scale_factors, err_ind, transf_matrix*) a SCALE (*scale_factors, err_ind, transf_matrix*) vrátia **maticu škálovania** určeného trojicou resp. dvojicou škálovacích faktorov. ROTATE X (*angle, err_ind, transf_matrix*), ROTATE Y (*angle, err_ind, transf_matrix*) a ROTATE Z (*angle, err_ind, transf_matrix*) vrátia **maticu otočenie okolo osi *x, y*, resp. *z***, pričom uhol zadáme v radiánoch kladne v smere proti pohybu hodinových ručičiek. **Matica otočenia v 2D** sa vypočíta funkciou ROTATE (*angle, err_ind, transf_matrix*).

Skladanie dvoch transformácií (násobenie ich matíc) voláme funkciami:

COMPOSE MATRIX 3 (*matrix_A, matrix_B, err_ind, comp_matrix*) a

COMPOSE MATRIX (*matrix_A, matrix_B, err_ind, comp_matrix*).

Tak postupne získame maticu ľubovoľnej transformácie, no dá sa aj naraz:

BUILD TRANSFORMATION MATRIX 3 (*fix_point, shift_vector, angle_X, angle_Y, angle_Z, scale_factors, err_ind, transf_matrix*)

BUILD TRANSFORMATION MATRIX (*fix_point, shift_vector, angle, scale_factors, err_ind, transf_matrix*)

Poradie transformácií je: škálovanie, rotácia okolo osi ***x***, rotácia okolo osi ***y***, rotácia okolo osi ***z*** (resp. len rotácia) a posunutie. Parameter *fix_point* špecifikuje **pevný bod pre škálovanie a rotáciu**. Niekedy treba na výpočet matice transformácie **skladanie dvoch transformácií**, pričom pri prvej transformácii poznáme jej maticu a druhú máme rozloženú na posunutie, rotáciu a škálovanie. Sú to funkcie

COMPOSE TRANSFORMATION MATRIX 3 (*tran_matrix, fix_point, shift_vector, angle_X, angle_Y, angle_Z, scale_factors, err_ind, comp_matrix*)

COMPOSE TRANSFORMATION MATRIX (*tran_matrix, fix_point, shift_vector, angle, scale_factors, err_ind, comp_matrix*),

pričom poradie skladania jednotlivých zložiek druhej transformácie a význam pevného bodu je rovnaký ako v predchádzajúcich dvoch funkciách.

PHIGS ponúka aj dve funkcie, ktoré vrátia bod *transf_point* transformovaný maticou *transf_matrix* z bodu *point:*

TRANSFORM POINT 3 (*point, transf_matrix, err_ind, transf_point*)

TRANSFORM POINT (*point, transf_matrix, err_ind, transf_point*).

Tieto pomocné funkcie na výpočet matice transformácie nie sú prvkami štruktúr.

19.2.2 Snímanie

V 2D grafických systémoch je smer pohľadu pozorovateľa vždy kolmý k pozorovanej rovine. V 3D grafickom systéme pribúda **poloha pozorovateľa** vzhľadom k pozorovaným objektom, tak ako je to v reálnom 3D svete, čo na obr. 1.1 v kapitole 1 ilustruje dlhá šípka. Analógiou vytvárania obrázku z modelu môže byť práca fotografa alebo filmára pri fotografovaní alebo filmovaní. Vytváranie objektov v modelujúcich súradniciach zodpovedá zhotovovaniu objektov scény (navzájom nezávislo), zloženej modelujúcej transformácii zodpovedá ich vzájomné usporiadanie a celkové usporiadanie na scéne. Keď je scéna hotová, fotograf nájde najvhodnejšiu polohu (miesto a smer) pre fotoaparát a začne snímať. **Snímanie** časti modelu teraz popíšeme detailne. (Slovenským termínom snímanie sa vystihuje funkčnosť anglického *viewing*, [ERTL93]). Poslednou fázou fotografovania je výroba fotografií. Z jedného políčka filmu možno vyrobiť viac fotografií rôznych veľkostí, no nasnímaný záber už nezmeníme. Zväčšovanie či zmenšovanie detailov z políčka filmu na fotografický papier možno prirovnať k **transformácii na stanicu**.

PHIGS preto netransformuje hneď WC do obdoby NDC z GKS, ale medzi svetové a normalizované súradnice vsúva ďalší súradnicový systém. Svetové súradnice sa najskôr transformujú pomocou **matice orientácie pohľadu** (*view orientation matrix*) do **referenčných súradníc pohľadu** (*view reference coordinates,* VRC) a tieto sa potom transformujú pomocou **matice projekcie** (*view mapping matrix*) do **normalizovaných projekčných súradníc** (*normalized projection coordinates,* NPC). Matica orientácie pohľadu popisuje orientáciu a umiestnenie scény vzhľadom k pozorovateľovi, matica projekcie určuje rovnobežnú alebo stredovú projekciu. Pri transformácii z VRC do NPC je možnosť ďalšieho orezávania - **snímacie orezávanie** (*view clipping*) - vzhľadom na rovnobežnosten v NPC s hranami rovnobežnými so súradnými osami (na rozdiel od modelujúceho orezávania, kde orezávací priestor nemusí byť iba rovnobežnosten). Zloženie týchto dvoch transformácií aj s prípadným orezaním sa nazýva **normalizujúcou transformáciou** (*viewing transformation*). Každej stanici prislúcha jedna **tabuľka pohľadov** (*view table*), kde sú uložené jedna alebo viac normalizujúcich transformácií. Celý tento proces je teda na stanici závislý, t.j. každej stanici možno priradiť iný "uhol pohľadu" na tú istú scénu.

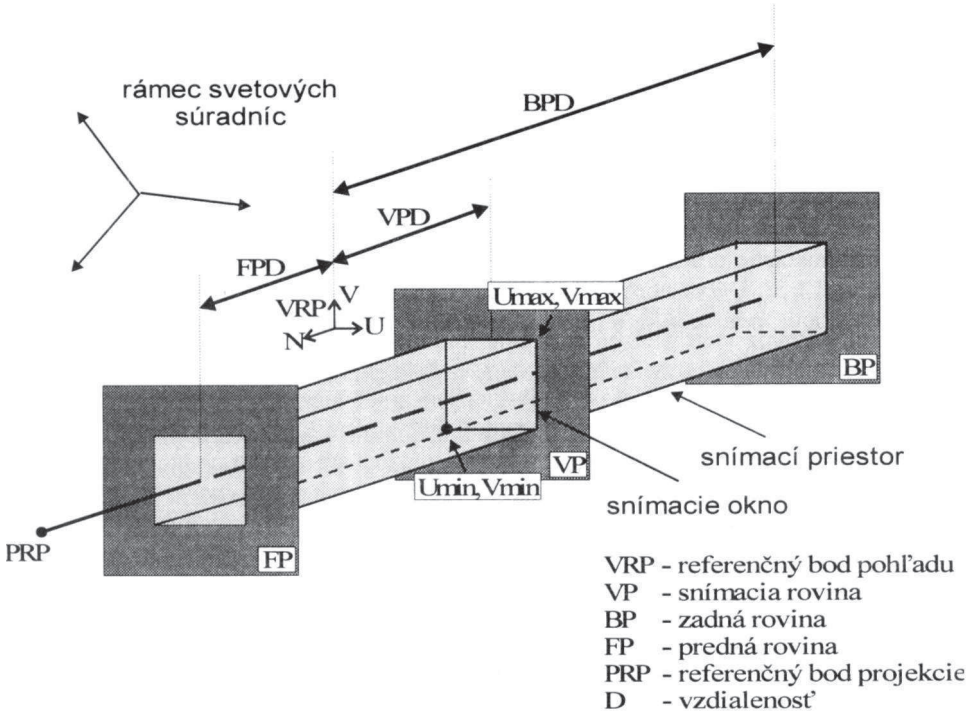
Implementácia PHIGS-u môže pre jednotlivé stanice preddefinovať niekoľko normalizujúcich transformácií, ktoré sa pri otvorení stanice nastavia na hodnoty z tabuľky popisu príslušnej stanice. Položky v tabuľke pohľadov môžeme nastavovať funkciami SET VIEW REPRESENTATION 3 (*ws_id, view_index, orient_matrix, map_matrix, clip_limits, xy_clip, back_clip, front_clip*) a SET VIEW REPRESENTATION (*ws_id, view_index, orient_matrix, map_matrix, clip_limits, xy_clip*), kde *view_index* je index položky v tabuľke pohľadov, *orient_matrix* je matica orientácie pohľadu, *map_matrix* je

matica projekcie, *clip_limits* sú hranice snímacieho orezávania (v prvej funkcii tri NPC dvojice, v druhej dve NPC dvojice) a posledné tri (resp. jeden) parametre sú indikátory snímacieho orezávania (CLIP alebo NOCLIP). Indikátor *xy_clip* riadi orezávanie vzhľadom k rovinám x=XMIN, x=XMAX, y=YMIN, y=YMAX, indikátorom *back_clip* riadime orezávanie vzhľadom k zadnej rovine, t.j. k rovine z=ZMIN a indikátorom *front_clip* riadime orezávanie vzhľadom k prednej rovine z=ZMAX. V každej tabuľke pohľadov je špeciálna normalizujúca transformácia s indexom 0, hodnoty ktorej sa nesmú meniť, predpisuje ich norma: matica orientácie = I, matica projekcie = I, vrcholy hranice snímacieho orezávania sú 0, 1, 0, 1, 0, 1 a všetky indikátory orezávania sú nastavené na CLIP. Výber normalizujúcej transformácie z tabuľky pohľadov riadime pomocou atribútu *index pohľadu* (*view index*), ktorý nastavíme funkciou SET VIEW INDEX (*view_index*).

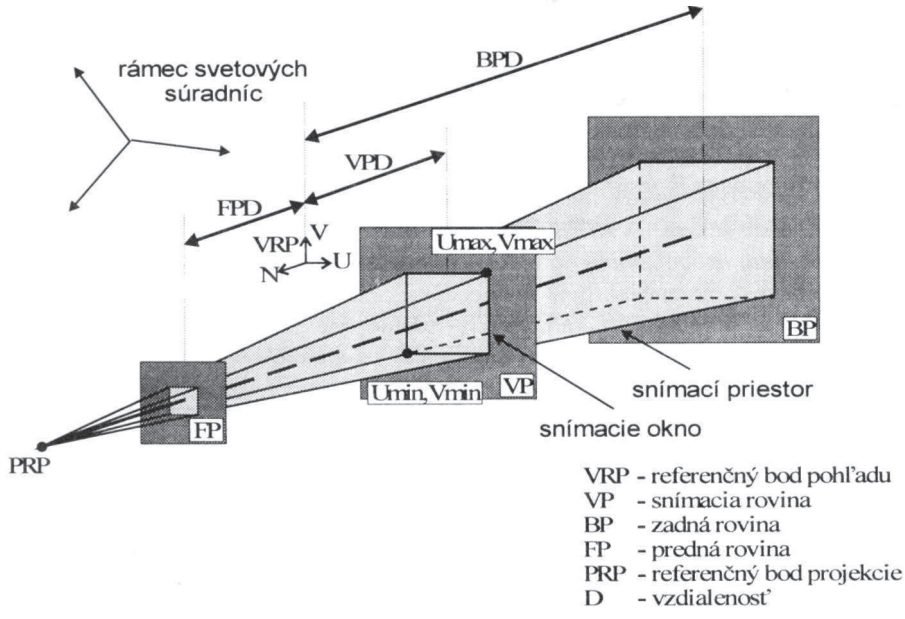
Každej položke v tabuľke pohľadov každej pracovnej stanice zodpovedá **vstupná priorita** (jej význam popíšeme pri vstupe). Počiatočné hodnoty vstupných priorít sa zhodujú s indexami jednotlivých pohľadov a možno ich zmeniť volaním SET VIEW TRANSFORMATION INPUT PRIORITY (*ws_id, index, ref_index, relat_priority*), ktorá nastaví na stanici *ws_id* vstupnú prioritu pre transformáciu *index* o 1 vyššiu alebo nižšiu než je priorita transformácie *ref_index* (vyššiu: HIGHER, nižšiu: LOWER). Najvyššia priorita je 0.

Podobne ako pri modelujúcich transformáciách, poskytuje PHIGS pomocné funkcie aj pre výpočet matíc potrebných pri normalizujúcej transformácii. Model, ktorý používa PHIGS pre orientáciu a projekciu nie je jediný - aplikácia môže používať vlastné modely a vlastné metódy pre výpočet matíc. Všetky štyri pomocné funkcie, ktoré ponúka PHIGS a ktoré sú popísané nižšie, majú dva výstupné parametre: indikátor chyby a výstupnú maticu. Pre nesprávny alebo nekonzistentný vstup *err_ind* hlási príčinu chyby. Funkcia na výpočet matice orientácie v 3D EVALUATE VIEW ORIENTATION MATRIX 3 (*ref_point, normal, up_vector, err_ind, orient_matrix*) očakáva referenčný bod *ref_point* vo WC a definuje počiatok VRC. Normálový vektor *normal* definuje tretiu os (N) VRC. Rovina obsahujúca referenčný bod a kolmá na normálu je **referenčná rovina**. Priemet vektora *up_vector* do referenčnej roviny (vzhľadom k referenčnému bodu) definuje druhú os (V) systému VRC. Prvá os (U), kolmá na osi V a N, je určená tak, aby osi UVN tvorili pravotočivý súradnicový systém.

V 2D prípade funkcia EVALUATE VIEW ORIENTATION MATRIX (*ref_point, up_vector, err_ind, orient_matrix*) považuje referenčný bod za počiatok VRC v rovine WC z=0, normála k referenčnej rovine sa neudáva, pretože je rovnobežná s osou z (a referenčnou rovinou je z=0 vo WC) a *up_vector* určuje smer osi V v rovine z=0. Os U je určená tak, aby UVN tvorili pravotočivý súradnicový systém.



Obr. 19.5 Snímací priestor pri rovnobežnej projekcii



Obr. 19.6 Snímací priestor pri perspektívnej projekcii

Na oboch obrázkoch 19.5 a 19.6 BPD, VPD a FPD znamenajú vzdialenosti príslušných rovín od počiatku. Na výpočet matíc projekcie slúžia funkcie EVALUATE VIEW MAPPING MATRIX 3 (*wind_limits, proj_viewport_limits, proj_type, proj_ref_point, view_plane_dist, back_plane_dist, front_plane_dist, err_ind, map_matrix*) a EVALUATE VIEW MAPPING MATRIX (*wind_limits, proj_viewport_limits, err_ind, map_matrix*).

Tieto funkcie vrátia maticu reprezentujúcu transformáciu, ktorá zobrazí **snímací priestor** (*view volume*) vo VRC do priestoru v NPC ohraničeného hranicami **záberu projekcie** (*projection viewport limits*). Záber projekcie je kváder v NPC s hranami rovnobežnými s osami NPC. Hoci NPC konceptuálne prekračuje oblasť [0,1]*[0,1]*[0,1], hranice záberu projekcie môžu byť umiestnené len do kocky [0,1]*[0,1]*[0,1]. Zvyčajne budú mať hranice záberu projekcie a hranice snímacieho orezávania (ktoré sú parametrom funkcií SET VIEW REPRESENTATION) rovnaké hodnoty, ale nie je to záväzné. **Predná rovina** (*front plane*), **zadná rovina** (*back plane*) a **snímacia rovina** (*view plane*) sú roviny rovnobežné s rovinou UV v systéme VRC. Sú špecifikované ako hodnoty súradníc na osi N pomocou parametrov *front_plane_dist, back_plane_dist a view_plane_dist.* Keďže predná rovina nemôže byť za zadnou rovinou, hodnota *front_plane_dist* nesmie byť menšia ako hodnota *back_plane_dist*. Predná a zadná rovina obsahujú prednú a zadnú stenu snímacieho priestoru. Ostatné steny snímacieho priestoru sa určia pomocou okna, špecifikovaného štvoricou VRC hodnôt UMIN, UMAX, VMIN, VMAX v parametri *wind_limits*. Tieto hodnoty definujú obdĺžnikovú oblasť v snímacej rovine. Bočné steny snímacieho priestoru prechádzajú hranami tohto obdĺžnika. Ich presná poloha závisí od typu projekcie, špecifikovaného parametrom *proj_type*, ktorý môže nadobúdať hodnoty PARALLEL alebo PERSPECTIVE. Pri rovnobežnej projekcii sú bočné steny rovnobežné s priamkou prechádzajúcou referenčným bodom projekcie (špecifikovaným vo VRC parametrom *proj_ref_point*) a stredom okna v snímacej rovine. Pri perspektívnej projekcii sú bočné steny časťami rovín určených priamkami, prechádzajúcimi referenčným bodom projekcie a príslušnými vrcholmi okna v snímacej rovine (v tomto prípade snímací priestor tvorí zrezaný ihlan). V dvojrozmernom prípade je to všetko jednoduchšie. Okno vo VRC leží v rovine N=0 a záber v NPC.

19.2.3 Transformácia na stanicu

Body z NPC sa nakoniec **transformáciou na stanicu** (*workstation transformation*) zobrazia do priestoru **súradníc zariadenia** (*device coordinates,* DC). Transformácia na stanicu je definovaná pomocou hraníc priestoru NPC v rozsahu [0,1]*[0,1]*[0,1] - **okna stanice** (*workstation window*), ktoré sa zobrazí na určený priestor v DC - **záber stanice** (*workstation viewport*). Okno aj záber stanice sú špecifikované rovnobežnostenmi s hranami rovnobežnými so súradnými osami v NDC, resp. v DC. Ak by okno a záber stanice mali pomer strán na osiach ***x, y,*** alebo ***z*** rozdielny, tak transformácia na stanicu by nebola izotropická. PHIGS však vyžaduje, aby transformácia na stanicu zachovávala pomer strán na osiach ***x*** a ***y***. Presnejšie, transformáciou na stanicu sa okno stanice zobrazí na najväčší rovnobežnosten vo vnútri záberu stanice, spĺňajúci nasledujúce podmienky: **1.** zachová sa pomer strán v smeroch osí ***x*** a ***y***, **2.** roh okna stanice najbližší k (0,0,0) sa zobrazí do ľavého dolného rohu, ktorý je ďalej od pozorovateľa, **3.** ***z***-ový rozsah okna stanice sa zobrazí na celý ***z***-ový rozsah záberu stanice. Ak sa zachová pomer strán na osiach ***x, y, z,*** tak sa okno stanice zobrazí na záber pri splnení bodu **2.**

Transformáciu na stanicu možno zmeniť pomocou funkcií

SET WORKSTATION WINDOW 3 (*ws_id, ws_wind_limits*)

SET WORKSTATION WINDOW (*ws_id, ws_wind_limits*)

SET WORKSTATION VIEWPORT 3 (*ws_id, ws_viewport_limits*)

SET WORKSTATION VIEWPORT (*ws_id, ws_viewport_limits*),

kde hranice okna stanice určíme tromi (resp. dvoma) dvojicami NPC súradníc a hranice záberu stanice sú určené tromi (resp. dvoma) dvojicami v DC. Implicitne sa okno a záber stanice nastaví tak, aby transformácia na stanicu zobrazila NPC kocku [0,1]*[0,1]*[0,1] na celý zobrazovací priestor stanice. Ak zobrazovací priestor nie je kockou, platia horeuvedené pravidlá. Zmena transformácie na stanicu môže spôsobiť **implicitnú regeneráciu obrazu**.

Pri transformácii na stanicu sa vždy orezáva na okno stanice. DC reprezentujú súradnice (napr. v metroch) fyzickej zobrazovacej plochy, na ktorú sa má obrázok vykresliť. Body v priestore DC zodpovedajú bodom na zobrazovacej ploche zariadenia podľa troch pravidiel: **1.** počiatku DC zodpovedá na zobrazovacej ploche bod v ľavom dolnom rohu najďalej od operátora, **2.** jednotky v DC zodpovedajú zobrazovacej ploche tak, že kocka v DC vyzerá na zobrazovacej ploche ako kocka, **3.** smery kladných polosí ***x, y, z*** sú doprava, hore a smerom k pozorovateľovi.

Celý transformačný kanál PHIGS znázorňuje obr. 19.4.

19.3 Grafický výstup

Oproti GKS pozná PHIGS popri 2D aj 3D prvky a navyše dva nové druhy: **relatívny popisný text** a **skupinu výplňových oblastí**. Relatívny popisný text generuje znakový reťazec v rovine ***x-y*** priestoru NPC. Skupinou výplňových oblastí špecifikujeme oblasti s dierami alebo nesúvislé oblasti, ktoré považujeme za jeden objekt. 2D prvky štruktúr pre výstupné grafické prvky popisujú rovinné objekty, ale 3D text, 3D výplňová oblasť, 3D skupina výplňových oblastí a 3D pole buniek ostanú rovinnými výstupnými prvkami v ľubovoľnej rovine. V 2D prípade je rovinou z=0, v 3D prípade sa pripúšťa ľubovoľná rovina priestoru MC. Konceptuálne tieto roviny majú dve strany a nulovú hrúbku. Normalizujúca transformácia určí, ktorú stranu rovinných výstupných prvkov vidno. Relatívny popisný text je tiež rovinný výstupný prvok, ale leží v rovine **x-y** NPC priestoru.

Výstupný prvok môže mať štyri typy atribútov: **geometrické, negeometrické, snímacie a identifikačné**. Prvé dve skupiny atribútov určujú presný vzhľad prvku.

Geometrické atribúty nezávisia na stanici a ak reprezentujú súradnicové dáta (napr. body, vektory), tak sú vyjadrené v MC (s výnimkou pre text a relatívny popisný text, kde sú vyjadrené v textovom lokálnom súradnicovom systéme). Hodnoty geometrických atribútov sa transformujú takisto ako geometrické parametre výstupných prvkov (znovu okrem relatívneho popisného textu, ktorý transformujú len tie transformácie, ktoré sa vykonajú po transformácii do NPC priestoru).

Negeometrickými atribútmi riadime vzhľad výstupných prvkov (typ čiary, farba textu) resp. tvar alebo veľkosť výstupných prvkov (napr. koeficient veľkosti značky).

Negeometrické aspekty nereprezentujú súradnicové dáta. Riadia sa tak ako v GKS - zväzkovo alebo individuálne. Konflikty riešia takisto **príznaky pôvodu aspektov** (*aspect source flags*, ASFs). Tie môžeme meniť príkazom:

SET INDIVIDUAL ASF (*aspect_id, aspect_source*),

kde enumeračným parametrom *aspect_id* určíme, ktorý aspekt nastaviť na hodnotu *aspect_source*. Tu na rozdiel od GKS meníme nie všetky ale práve jeden z príznakov: *LINETYPE ASF, LINEWIDTH SCALE FACTOR ASF, POLYLINE COLOUR INDEX ASF, MARKER TYPE ASF, MARKER SIZE SCALE FACTOR ASF, POLYMARKER COLOUR INDEX ASF, TEXT FONT ASF, TEXT PRECISION ASF, CHARACTER EXPANSION FACTOR ASF, CHARACTER SPACING ASF, TEXT COLOUR INDEX ASF, INTERIOR STYLE ASF, INTERIOR STYLE INDEX ASF, INTERIOR COLOUR INDEX ASF, EDGE FLAG ASF, EDGETYPE ASF, EDGEWIDTH SCALE FACTOR ASF, EDGE COLOUR INDEX ASF.*

Kvôli zväzkovému riadeniu atribútov má stanica **tabuľky zväzkov** pre každú skupinu aspektov, kde funkciou SET xxx REPRESENTATION... (xxx môže byť: POLYLINE, POLYMARKER, TEXT, INTERIOR, EDGE) vieme zväzky predefinovať alebo vytvoriť. Okrem tabuliek zväzkov má stanica aj **tabuľku reprezentácií pre vzorky** a **tabuľku farieb** (o nich neskôr).

Snímacie atribúty sú **index pohľadu** (*view index*) a **identifikátor hlhsr** (*hlhsr identifier*), hlhsr skracuje *hidden line hidden surface removal*. Prvý z nich sme už popísali, druhým riadime odstraňovanie neviditeľných čiar a plôch, ktoré závisia od implementácie a musíme ich riadiť na stanici závislým spôsobom pomocou **režimu hlhsr** (*hlhsr mode*) a nezávislo na stanici práve pomocou atribútu identifikátor hlhsr:

SET HLHSR IDENTIFIER (*hlhsr_id*),

SET HLHSR MODE (*ws_id, hlhsr_mode*).

Aj štvrtá skupina atribútov - **identifikačné** - má len dva prvky: **identifikátor výberu** (*pick identifier*) a **skupina mien** (*name set*). Identifikátorom výberu identifikujeme výstupný prvok (skupinu prvkov) v štruktúre po výbere vstupným zariadením. Pomocou atribútu **skupina mien (nezávisí na stanici)** a **filtrov (závisia na stanici)** sa riadi neviditeľnosť, zvýraznenie a detekovateľnosť výstupných prvkov. Obsah atribútu skupina mien sa počas prechodu štruktúrou riadi prvkami štruktúr

ADD NAMES TO SET (*names*),

REMOVE NAMES FROM SET (*names*). Na začiatku prechodu je skupina mien prázdna. Na každej pracovnej stanici (pre výber dokonca pre každé zariadenie výberu) existujú dvojice množín pre zvýraznenie, pre neviditeľnosť a pre výber, ktoré fungujú počas prechodu ako filtre. Označujeme ich ako **priepustná množina pre xxx** (*xxx inclusion set*) a **vylučovacia množina pre xxx** (*xxx exclusion set*), kde **xxx** môže byť **neviditeľnosť** (*invisibility*), **zvýraznenie** (*highlighting*), alebo **výber** (*pick*). Filtre nastavíme funkciami

SET HIGHLIGHTING FILTER (*ws_id, highl_filter*),

SET INVISIBILITY FILTER (*ws_id, invis_filter*),

SET PICK FILTER (*ws_id, pick_dev_number, pick_filter*),

kde každý filter je dvojicou množín v poradí priepustná, vylučovacia. V oboch môžu byť mená, z množiny mien, ale aj iné. Aby sa výstupný prvok zvýraznil, zneviditeľnil alebo vybral, musí jeho atribút skupina mien mať aspoň jedno meno v priepustnej množine a zároveň nesmie mať žiadne meno vo vylučovacej množine. Ak sú obe množiny príslušného filtra prázdne, tak všetky výstupné prvky na stanici implicitne nie sú zvýraznené, sú viditeľné a nie sú vybrateľné. Táto zdanlivo zložitá metóda dovoľuje neviditeľnosť, zvýraznenie a detekovateľnosť meniť na jednotlivej stanici (na rozdiel od GKS) a meniť tieto atribúty vzhľadom k jednotlivým výstupným prvkom štruktúry (v GKS zmeny postihnú celý segment).

Treba upozorniť, že v ďalšom nebudeme uvádzať pri výstupných prvkoch snímacie a identifikačné atribúty, lebo patria ku každému grafickému výstupnému prvku.

19.3.1 Špecifikácia farby

Farbu vždy špecifikujeme ako **index do tabuľky farieb** pracovnej stanice, kde musia existovať aspoň položky 0 a 1, implicitné farby pozadia a popredia. Farbu určíme farebným modelom a súradnicami farby v tomto modeli. Každá stanica môže používať svoj farebný model. Tabuľku farieb nastavíme volaním SET COLOUR REPRESENTATION (*ws_id, colour_index, colour_specif*), kde *colour_specif* sú súradnice farby vo farebnom modeli aktuálne platnom na stanici. Farebný model nastavíme funkciou SET COLOUR MODEL (*ws_id, colour_model*), pričom *colour_model* znamená: 0 závislé na implementácii, 1 RGB, 2 CIELUV, 3 HSV, 4 HLS, hodnoty nad 5 sa rezervujú pre ďalšiu registráciu.

19.3.2 Grafické výstupné prvky

Lomené čiary a individuálne riadenie ich atribútov poskytujú funkcie

POLYLINE 3 (*point_list*), POLYLINE (*point_list*),

SET LINETYPE (*linetype*),

SET LINEWIDTH SCALE FACTOR (*width_scale_factor*),

SET POLYLINE COLOUR INDEX (*colour_index*).

Index do tabuľky zväzkov nastavíme funkciou SET POLYLINE INDEX (*index*) a položky v tabuľke zväzkov lomenej čiary (reprezentáciu na stanici) nastavujeme volaním SET POLYLINE REPRESENTATION (*ws_id, index, linetype, width_scale_factor, colour_index*). Ako vidno, všetko okrem 3D súradníc sa zhoduje s GKS.

To isté platí pre **sled značiek**: POLYMARKER 3 (*point_list*), POLYMARKER (*point_list*), SET MARKER TYPE (*markertype*), SET MARKER SIZE SCALE FACTOR (*size_scale_factor*), SET POLYMARKER COLOUR INDEX (*colour_index*), SET POLYMARKER INDEX (*index*), SET POLYLINE REPRESENTATION (*ws_id, index, marker_type,size_scale_factor, colour_index*).

PHIGS ponúka **text** aj **relatívny popisný text** (*annotation text relative*):

TEXT 3 (*position, direction_vectors, string*), TEXT (*position, string*),

ANNOTATION TEXT RELATIVE 3 (*ref_point, offset, string*),

ANNOTATION TEXT RELATIVE (*ref_point, offset, string*).

Parametre *direction_vectors* určujú rovinu, v ktorej 3D text leží. Relatívny popisný text sa vzťahuje k referenčnému bodu *ref_point*, ktorého súradnice sú dané v priestore MC. Polohu anotačného textu zadáme v NPC priestore posunom *offset* z transformovaného referenčného bodu.

Medzi geometrické atribúty textu patria **výška znaku** (*character height*), **vertikálny vektor znaku** (*character up vector*), **smer textu** (*text path*) a **zarovnanie textu** (*text alignment*). K podobným atribútom relatívneho popisného textu pribúda **typ anotácie** (*annotation style*). Prvé štyri atribúty (aj pre text aj pre relatívny popisný text) majú rovnaký význam ako v GKS, iba geometrické údaje pre výšku znaku a vertikálny vektor znaku zadávame v tzv. **textovom lokálnom súradnicovom systéme** (TLC), ktorý je vždy dvojrozmerný:

SET CHARACTER HEIGHT (*char_height*)

SET CHARACTER UP VECTOR (*char_up_vector*)

SET TEXT PATH (*text_path*)

SET TEXT ALIGNMENT (*text_align_hor, text_align_ver*)

SET ANNOTATION TEXT CHARACTER HEIGHT (*char_height*)

SET ANNOTATION TEXT CHARACTER UP VECTOR (*char_up_vector*)

SET ANNOTATION TEXT PATH (*text_path*)

SET ANNOTATION TEXT ALIGNMENT (*text_align_hor, text_align_ver*)

Typ anotácie relatívneho popisného textu určuje, ako spojiť referenčný bod s bodom udávajúcim polohu popisného textu - nijako alebo úsečkou s atribútmi ako pre lomenú čiaru: SET ANNOTATION STYLE (*annot_style*).

Medzi negeometrické aspekty, ktoré riadia vzhľad oboch textov patria: **druh písma** (*text font*), **presnosť textu** (*text precision*), **koeficient rozšírenia znaku** (*character expansion factor*), **rozostup znakov** (*character spacing*) a **index farby textu** (*text colour index*). Ich význam je rovnaký ako v GKS, iba atribút **druh a presnosť textu** je rozdelený do dvoch. Na nastavovanie ich hodnôt slúžia funkcie:

SET TEXT FONT (*text_font*),

SET TEXT PRECISION (*text_precision*),

SET CHARACTER EXPANSION FACTOR (*char_exp_factor*),

SET CHARACTER SPACING (*char_spac*),

SET TEXT COLOUR INDEX (*colour_index*).

Zväzok atribútov pre text dosiahneme cez **index textu** pomocou SET TEXT INDEX (*index*) a **reprezentáciu textu** pre daný index nastavíme funkciou SET TEXT REPRESENTATION (*ws_id, index, text_font, text_precision, char_exp_factor, char_spac, colour_index*). **Orezávanie textu** závisí znova od jeho presnosti. Relatívny popisný text vidno vždy, aj pri zakrytí iným výstupným prvkom, ak jeho referenčný bod leží v orezávacom priestore.

Funkcie pre **výplňové oblasti** sú:

FILL AREA 3 (*point_list*), FILL AREA (*point_list*), FILL AREA SET 3 (*list_of_point_list*), FILL AREA SET (*list_of_point_list*).

Skupina výplňových oblastí slúži najmä na zostrojenie oblastí s dierami. Rovinu pre výplňovú oblasť a skupinu výplňových oblastí definujeme bodmi a za koplanárnosť bodov zodpovedná aplikačný program. Malé odchýlky (napr. z chýb zaokrúhľovania) by mal PHIGS odstrániť. Existuje dôležitý rozdiel medzi výplňovou oblasťou a skupinou výplňových oblastí. Kým výplňová oblasť má len známe **atribúty vnútra** (plus tie, ktoré sú spoločné pre všetky výstupné prvky), tak skupina výplňových oblastí má ešte navyše **atribúty hrán**, ktorými sú: **príznak hrany** (*edge flag*), **typ hrany** (*edgetype*), **koeficient hrúbky hrany** (*edgewidth scale factor*), **index farby hrany** (*edge colour index*), **index hrany** (*edge index*), **ASF príznaku hrany** (*edge flag ASF*), **ASF typu hrany** (*edgetype ASF*), **ASF koeficientu hrúbky hrany** (*edgewidth scale factor ASF*) a **ASF indexu farby hrany** (*edge colour index ASF*). Príznak hrany môže nadobúdať dve hodnoty: ON alebo OFF a nastavíme ho funkciou SET EDGE FLAG (*edge_flag*). Pre OFF sa hrana nevykreslí, inak sa vykreslí v súlade s hodnotami ostatných hranových atribútov, ktorých význam je rovnaký ako pri zodpovedajúcich atribútoch lomenej čiary. To isté platí pre tabuľku zväzkov.

Pole buniek môžme vyvolať 3D alebo 2D:

CELL ARRAY 3 (*parallelogram, colour_index_array*),

CELL ARRAY (*rectangle, colour_index_array*).

V oboch prípadoch sa vykreslí rovinné **dvojrozmerné pole buniek**. V 3D verzii leží pole buniek v rovine, ktorá je definovaná trojicou bodov v prvom parametri funkcie CELL ARRAY 3. Na rozdiel od GKS, v PHIGS-e nemusí byť pole buniek v 3D verzii uzavreté do obdĺžnika, ale môže byť rovnobežníkom. Ak si označíme body špecifikované v prvom parametri funkcie CELL ARRAY 3 ako P, Q, R, tak vrcholmi tohto rovnobežníka sú P, Q, R a bod (Q+R-P). Mriežka určujúca jednotlivé bunky je subjektom všetkých transformácií (a môže teda nastať transformácia buniek do nerovnobežníkov).

Zovšeobecnený výstupný prvok (GDP) slúži na dosiahnutie špeciálnych schopností pracovných staníc, týkajúcich sa grafického výstupu (kreslenie splajnov, kružnicových a eliptických oblúkov). Výstupné objekty charakterizujeme identifikátorom, možinou bodov a doplňujúcimi údajmi. Presné tvary funkcií na volanie 3D a 2D verzie GDP sú GENERALIZED DRAWING PRIMITIVE 3 (*point_list, GDP3_identifier, GDP3_data_rec*) a GENERALIZED DRAWING PRIMITIVE (*point_list, GDP_identifier, GDP_data_rec*).

Zovšeobecnený výstupný prvok nemá geometrické atribúty, ale geometrické informácie môžu byť v dátovom zázname. Vzhľad GDP riadime **žiadnou alebo niekoľkými skupinami negeometrických atribútov ostatných prvkov** podľa záznamu v tabuľke popisu pracovnej stanice.

19.4 Centralizovaná pamäť štruktúr

19.4.1 Prvky štruktúr a siete štruktúr

Každú štruktúru v CSS **identifikujeme** jednoznačným **menom** (celým číslom). Štruktúra pozostáva z **prvkov štrukúry**, no môže byť aj **prázdna**. Štruktúru môžeme vytvoriť viacerými spôsobmi, napr. v najjednoduchšom prípade prázdnu štruktúru vytvoríme odvolávkou na jej meno. Neprázdnu štruktúru naplníme podobne ako segment v GKS akýmikoľvek prípustnými prvkami medzi dvojicou funkcií OPEN STRUCTURE (*name*) a CLOSE STRUCTURE(). Keď v programe štruktúru vytvárame, môžeme popri jej prvkoch vykonávať akékoľvek ďalšie operácie, ktoré sa do štruktúry neuložia (výpočet transformačných matíc či práca s aplikačnou databázou). Prvky štruktúr sa definujú v 7 skupinách. Uvedieme iba názvy prvkov bez parametrov.

1. prvky štruktúr pre výstupné grafické prvky: polyline 3, polyline, polymarker 3, polymarker, text 3, text, annotation text relative 3, annotation text relative, fill area 3, fill area, fill area set 3, fill area set, cell array 3, cell array, generalized drawing primitive 3, generalized drawing primitive.

2. prvky štruktúr pre špecifikáciu atribútov: set polyline index, set polymarker index, set text index, set interior index, set edge index, set linetype, set linewidth scale factor, set polyline colour index, set marker type, set marker size scale factor, set polymarker colour index, set text font, set text precision, set character expansion factor, set character spacing, set text colour index, set character height, set character up vector, set text path, set text alignment, set annotation text character height, set annotation text character up vector, set annotation text path, set annotation text alignment, set annotation style, set interior style, set interior style index, set interior colour index, set edge flag, set edgetype, set edgewidth scale factor, set edge colour index, set pattern size, set pattern reference point and vectors, set pattern reference point, add names to set, remove names from set, set individual ASF, set hlhsr identifier, set view index, set pick identifier

3. prvky štruktúr pre transformáciu a orezávanie: set local transformation 3, set local transformation, set global transformation 3, set global transformation, set modelling clipping volume 3, set modelling clipping volume, set modelling clipping indicator, restore modelling clipping volume

4. riadiaci prvok štruktúry: execute structure

5. editovací prvok štruktúry: label

6. zovšeobecnený prvok štruktúry: generalized structure element

7. prvok aplikačných dát: application data

Sieť štruktúr popisuje hierarchiu štruktúr v CSS ako orientovaný acyklický graf, ktorého vrchol znamená štruktúru a hrana vyvolane inej štruktúry funkciou EXECUTE STRUCTURE (*struct_id*). Vyvolanie nazývame **väzba na štruktúru** (*structure reference*). Siete štruktúr nesmú byť rekurzívne. Počet vyvolaných štruktúr ani hĺbku hierarchie PHIGS neobmedzuje. So sieťami štruktúr v CSS môžeme robiť mnoho typov operácií: **štruktúru zobraziť, editovať, manipulovať s ňou, v štruktúre hľadať a zisťovať, archivovať a vyberať štruktúry**. Zmeny v CSS, spôsobené týmito operáciami, sú

atomické a konceptuálne majú okamžitý účinok, hoci skutočný čas ešte závisí na schopnostiach stanice a **stave aktualizácie zobrazovania** (*display update state*) stanice. Štruktúra existuje pre všetky operácie, ak PHIGS pozná jej identifikátor.

19.4.2 Prechod štruktúrou a zobrazenie štruktúry

Sieť štruktúr zobrazíme volaním **odosielajúcej funkcie** (*posting function*) POST STRUCTURE (*ws_id, struct_id, disp_prior*), kde *ws_id* identifikuje stanicu, *struct_id* štruktúru a *disp_prior* je zobrazovacia priorita štruktúry (reálne číslo z intervalu [0,1]: tento pojem poznáme už pre segmenty v GKS). Zobrazovanie štruktúry zastavíme funkciou UNPOST STRUCTURE (*ws_id, struct_id*). Zobrazovanie všetkých štruktúr zastavíme funkciou UNPOST ALL STRUCTURES (*ws_id*).

Je jedno, či štruktúru najprv vytvoríme a potom odošleme na zobrazenie, alebo ju najskôr odošleme na zobrazenie (čím vytvoríme prázdnu štruktúru) a potom ju naplníme. Nasledujúce štvorice príkazov rovnako zobrazia tú istú lomenú čiaru:

OPEN STRUCTURE (*line*)	POST STRUCTURE (*ws, line, pr*)
POLYLINE (*point_list*)	OPEN STRUCTURE (*line*)
CLOSE STRUCTURE	POLYLINE (*point_list*)
POST STRUCTURE (*ws, line, pr*)	CLOSE STRUCTURE

Spolu s odoslanou štruktúrou sa zobrazia aj všetky štruktúry s ňou viazané, a preto musí PHIGS získať z CSS prvky štruktúr na spracovanie a zobrazenie. Atomická operácia použitá na spracovanie štruktúry sa nazýva **prechod štruktúrou** (*structure traversal*). Prechod zobrazuje grafický výstup zo siete štruktúr na všetky stanice, kam bola sieť odoslaná. S každým prechodom sa združí **stavový zoznam prechodu** (*traversal state list*), ktorého položkami sú atribúty výstupných prvkov (modifikovateľné prvkami štruktúr pre špecifikáciu atribútov), globálna a lokálna modelujúca transformácia, modelujúci orezávací priestor a indikátor modelujúceho orezávania, ktoré možno meniť prvkami štruktúr pre transformáciu a orezávanie. Pri každom začatí prechodu sieťou štruktúr sa inicializuje podľa tabuľky popisu PHIGS. Ak nejaká štruktúra má v CSS rodičov, ale v odoslanej sieti štruktúr je najvyššie (t.j. vystupuje ako parameter POST STRUCTURE), tak hodnoty položiek stavového zoznamu sa nededia z jej rodičovských štruktúr. Pre prípad modelujúcich transformácií to ilustruje príklad 19.1.

Proces prechodu štruktúrou interpretuje všetky prvky v štruktúre sekvenčne, počnúc prvým prvkom. Každá zmena v odoslanej sieti štruktúr spôsobí jej znovuprechádzanie, takže všetky zmeny v odoslanej sieti štruktúr sa spracujú ihneď. Viditeľnosť zmien na stanici závisí od stavu aktualizácie zobrazovania stanice. Keď sa počas prechodu narazí na prvok EXECUTE STRUCTURE, PHIGS pozastaví prechod aktuálnou štruktúrou, uloží stavový zoznam prechodu, nastaví globálnu a lokálnu transformáciu, urobí kompletný prechod "podsieťou štruktúr", obnoví uložený stavový zoznam prechodu a pokračuje v prechode pôvodnou aktuálnou štruktúrou. Prechod štruktúrou **nemôže spôsobiť chybu**, lebo vždy sú definované implicitné akcie.

19.4.3 Editovanie štruktúr a manipulácia s nimi

PHIGS poskytuje možnosť individuálnej modifikácie každého prvku v štruktúre. Editovacie funkcie umožňujú vložiť nový prvok, nahradiť existujúce prvky novými, zrušiť prvky, hľadať v štruktúre a informovať sa o jej obsahu. Pozície prvkov štruktúr sú implicitne očíslované číslami 0, 1 až n, kde pozícia 1 je prvý prvok a pozícia 0 je pred

prvým prvkom. Toto implicitné očíslovanie PHIGS stále udržiava: pri pridaní prvku zvýši poradové číslo o 1 a príslušne zmení čísla pri editovaní. Štruktúru treba na editovanie otvoriť a potom uzavrieť, napr.

```
OPEN STRUCTURE (XY)          /* 0 štruktúra XY */
POLYLINE (...)               /* 1 POLYLINE */
POLYMARKER (...)             /* 2 POLYMARKER */
TEXT (...)                   /* 3 TEXT */
CLOSE STRUCTURE              /* koniec štruktúry XY */
```

Pri každom otvorení štruktúry PHIGS poskytne **smerník na prvok**, ktorý ukazuje na posledný prvok. Jeho hodnotu možno zistiť volaním INQUIRE ELEMENT POINTER (*err_ind, elem_pos*). Toto je príklad funkcie, ktorá mení obsah štruktúry a nemá parameter *struct_id.* Také funkcie sa vzťahujú k aktuálne otvorenej štruktúre, a preto ich používame len medzi dvojicou funkcií OPEN STRUCTURE a CLOSE STRUCTURE.

Smerník na prvok riadime absolútne alebo relatívne k aktuálnej pozícii alebo návestím. Podľa potreby voláme funkcie SET ELEMENT POINTER (*elem_pos*), OFFSET ELEMENT POINTER (*elem_pos_offset*), SET ELEMENT POINTER AT LABEL (*label_id*). Návestie (celé číslo) *label_id* vložíme do štruktúry funkciou LABEL (*label_id*), čím zavádzame popri implicitnom očíslovaní aj svoje vlastné.

Príklad 19.3: Použitie návestí

```
OPEN STRUCTURE (XYZ)         /* 0 štruktúra XYZ */
LABEL (10)                   /* 1 návestie 10 */
POLYLINE (...)               /* 2 POLYLINE */
LABEL (20)                   /* 3 návestie 20 */
POLYMARKER (...)             /* 4 POLYMARKER */
LABEL (15)                   /* 5 návestie 15 */
TEXT (...)                   /* 6 TEXT */
CLOSE STRUCTURE              /* koniec štruktúry XYZ */
```

To, či sa prvky v štruktúre vkladajú, alebo či nahrádzajú prvky už existujúce je určené *editovacím režimom* (*edit mode*), ktorý sa nastavíme funkciou SET EDIT MODE (*edit_mode*), kde *edit_mode* môže byť INSERT alebo REPLACE. V oboch prípadoch sa zvýši smerník o 1 (a v prvom prípade sa samozrejme prečíslujú všetky prvky za vloženým prvkom).

Kopírovanie všetkých prvkov zo štruktúry *struct_id* do otvorenej štruktúry zabezpečí COPY ALL ELEMENTS FROM STRUCTURE (*struct_id*). Pri tejto operácii sa vždy kopírované prvky vkladajú; editovací režim v tomto prípade nemá význam. Po skopírovaní bude smerník na prvok ukazovať na posledný skopírovaný prvok.

Prvky rušíme troma funkciami: DELETE ELEMENT (), DELETE ELEMENT RANGE (*elem_pos1, elem_pos2*) a DELETE ELEMENT BETWEEN LABELS (*label_id1, label_id2*) so zrejmým významom. Po každej z nich sa smerník nastaví na prvok bezprostredne pred prvok alebo skupinu prvkov, ktoré sme zrušili. Všetky prvky štruktúry zrušíme funkciou EMPTY STRUCTURE (*struct_id*), pričom sa zachová jej meno. Ak chceme zrušiť aj meno, treba štruktúru zrušiť.

Manipulovanie so štruktúrami. Štruktúry rušíme funkciami DELETE STRUCTURE (*struct_id*), DELETE ALL STRUCTURES () a DELETE STRUCTURE NETWORK (*struct_id, refer_hand_flag*), kde príznakom *refer_hand_flag* riadime, či sa majú zrušiť aj štruktúry, na ktoré sú väzby zo štruktúr mimo špecifikovanej siete. Identifikátor štruktúry môžeme zmeniť: CHANGE STRUCTURE IDENTIFIER (*orig_struct_id, new_struct_id*). Ak potrebujeme zmeniť identifikátory štruktúr, na ktoré sa odvolávame pomocou EXECUTE STRUCTURE, tak to dosiahneme volaním CHANGE STRUCTURE REFERENCES (*orig_struct_id, new_struct_id*) a obe akcie naraz spraví funkcia CHANGE STRUCTURE IDENTIFIER AND REFERENCES (*orig_struct_id, new_struct_id*). Prirodzene všetky tieto funkcie môžu zmeniť usporiadanie siete štruktúr.

19.5 Grafický vstup

Vstupný model PHIGS je prevzatý z GKS a rozšírený na 3D. Navyše môžeme identifikovať prvok štruktúry zariadením výberu. Vstupná hodnota z logického vstupného zariadenia triedy lokátor alebo sled polôh má konceptuálne tri súradnice. PHIGS preto dopĺňa 2D vstupné zariadenia pridaním tretej hodnoty získanej vnútorne (zo stavového zoznamu stanice) alebo zvonka (napr. vyžiadaním od operátora). Detaily sú vecou implementácie. Ďalší problém súvisiaci so vstupnými zariadeniami patriacimi do jednej z týchto dvoch tried spočíva v určení bodu vo WC, ktorý zodpovedá bodu zadanému v DC operátorom. Aby sa do aplikačného programu vrátila poloha vo WC, je nutné transformovať vstupné údaje z DC do WC spôsobom inverzným k transformačnému kanálu. Najskôr sa údaje transformujú z DC do NPC pomocou transformácie inverznej k transformácii na stanicu, ktorá je platná pri použití vstupu z logického vstupného zariadenia triedy lokátor alebo sled polôh (táto je jediná). Polohu v NPC je nutné transformovať z NPC do WC transformáciou inverznou k normalizujúcej transformácii. Ktorou konkrétne, to sa určí zo vstupnej priority normalizujúcich transformácií v tabuľke pohľadov danej stanice. Táto inverzná transformácia sa robí vzhľadom k transformácii s najvyššou vstupnou prioritou, ktorej orezávací priestor obsahuje danú pozíciu. Takáto transformácia vždy existuje, lebo transformácia pohľadu s indexom nula ma ako orezávací priestor celú jednotkovú kocku v NPC.

Vstupná hodnota z logického vstupného zariadenia triedy výber okrem stavu výberu (OK, NOPICK) obsahuje identifikáciu vybraného výstupného prvku. Na jednoznačnú identifikáciu vybraného výstupného prvku v hierarchii štruktúr treba, aby sa vrátila úplna cesta prechodu štruktúr. Cesta sa špecifikuje zoznamom položiek, každá z ktorých pozostáva z identifikátora štruktúry, identifikátora výberu a pozície prvku v štruktúre. Vybrať možno samozrejme len objekty viditeľné po všetkých orezávaniach a po použití filtra pre viditeľnosť a detekovateľné na danej pracovnej stanici. Zopakujme, že každé zariadenie výberu má vlastný filter. Preto, ak sú na stanici aspoň dve zariadenia výberu, jedným z nich môže operátor ten istý objekt vybrať a druhým nemusí.

19.6 Pracovné stanice

Zopakujme, že základným pojmom PHIGS je **pracovná stanica** (*workstation*). V tejto časti uvedieme niektoré ďalšie podrobnosti o charakteristikách a činnosti pracovných staníc. Údaje uložené v CSS môžu byť zobrazené na každej otvorenej stanici. Ku

každej stanici patrí niekoľko tabuliek obsahujúcich údaje popisujúce stanicu, stav stanice a atribúty závislé na stanici. Pri otvorení pracovnej stanice sa vytvorí **tabuľka popisu pracovnej stanice**, obsahujúca informácie získané zo samotného zariadenia a z implementačne závislých zdrojov (patrí sem typ a kategória stanice, a potom v závislosti od kategórie ďalšie údaje, napr. pre stanice kategórie OUTPUT sem okrem iného patrí: počet preddefinovaných indexov pohľadu, počet typov čiar, zoznam typov čiar, počet prípustných hrúbok čiar, nominálna hrúbka čiary, tabuľka preddefinovaných zväzkov, atď.).

Zobrazenie obrázku na zobrazovacej ploche sa prvotne riadi stavom CSS a hodnotami **stavového zoznamu stanice**. Stavový zoznam stanice obsahuje údaje súvisiace s danou pracovnou stanicou, ktoré sa môžu meniť v čase, keď je stanica otvorená. Patria sem napr. (pre stanice kategórie OUTPUT) počet položiek tabuľky pohľadov, tabuľka pohľadov, zoznam odoslaných štruktúr, režim zdržania, tabuľky zväzkov, aktuálny farebný model, atď.

Každá funkcia, ktorá môže ovplyvniť výsledný obrázok, môže zmeniť obsah CSS alebo stavový zoznam stanice. PHIGS rozlišuje **17 kategórií zmien obrázku**: modifikáciou reprezentácie pohľadu, reprezentácie zväzku lomenej čiary, reprezentácie zväzku sledu značiek, reprezentácie zväzku textu, reprezentácie zväzku vnútra, reprezentácie zväzku hrany, reprezentácie zväzku vzorky, reprezentácie farby, transformácie na stanicu, filtra pre zvýraznenie, filtra pre neviditeľnosť, režimu HLHSR, obsahu štruktúry, odoslanie štruktúry, zastavenie odosielania štruktúry (*unpost structure*), zrušenie štruktúry, zmena identifikátorov štruktúr. Pre každú zo 17 kategórií existuje v tabuľke popisu stanice položka *dynamic modification accepted for xxx*, ktorá môže nadobudnúť hodnotu: **1. IMM** - všetky zmeny obrázku danej kategórie môžu byť vykonané okamžite **(IMMediately)**, **2. IRG** - niektoré zmeny obrázku danej kategórie možno vykonať len pomocou regenerácie obrázku **(Implicit ReGeneration)**, **3. CBS** - niektoré zmeny obrázku danej kategórie sa dajú správne vykonať len pomocou regenerácie obrázku, a preto pre ne existuje rýchla aktualizačná metóda a zmenu obrázku ňou možno simulovať **(Can Be Simulated)**, aby sa dosiahla indikácia efektu bez regenerácie. Celkom správny efekt si ešte vyžiada regeneráciu. Simulovanie má poskytnúť okamžitý vizuálny výstup spolu s relatívne jasnou indikáciou, že zmena obrázku sa simulovala.

Aby sa vzali do úvahy aj schopnosti pracovnej stanice a požiadavky aplikačného programu, môže aplikačný program voliť medzi množstvom stratégií pre riadenie zmien obrázku. Aplikácia môže riadiť stupeň a čas, v ktorom obrázok odrazí skutočný stav CSS a tabuliek stanice. Aplikácia môže riadiť časovanie zobrazenia modifikácií, aby získala určité vizuálne efekty a využila schopnosti stanice efektívnejšie.

Položka stavového zoznamu stanice **stav vizuálnej reprezentácie** popisuje vzťah medzi aktuálnym stavom zobrazovacej plochy a správnym obrázkom určeným CSS a stavom stanice. Nastaviť hodnotu tejto položky nemôžeme, lebo sa nastavuje vnútorne, dá sa však zistiť. Môže mať jednu z troch hodnôt: **1.** CORRECT - zobrazovacia plocha ukazuje správny obrázok, **2.** DEFERRED - momentálne sa aspoň jedna zmena obrázku pozdržala, **3.** SIMULATED - momentálne sa aspoň jedna zmena obrázku simulovala a žiadna sa nepozdržala.

Položka stavového zoznamu stanice **režim zdržania** indikuje aktuálny čas, keď je obrázok správne vykreslený vzhľadom k CSS a stavu stanice. Jej možné hodnoty sú:

ASAP (*As Soon As Possible*) - obrázok sa na stanici stane vizuálne korektným ihneď, ako to bude možné

BNIG (*Before Next Interaction Globally*) - obrázok je vizuálne korektný pred nasledujúcou globálnou interakciou (t.j. pred nasledujúcou interakciou na hociktorom vstupnom zariadení na ľubovoľnej pracovnej stanici)

BNIL (*Before Next Interaction Locally*) - obrázok je vizuálne korektný pred nasledujúcou lokálnou interakciou (t.j. pred nasledujúcou interakciou na ľubovoľnom vstupnom zariadení na pracovnej stanici, ktorej sa režim BNIL týka)

ASTI (*At Some TIme*) - implementácia si na vykreslenie vizuálne korektného obrázku zvolí čas, ktorý jej najlepšie vyhovuje

WAIT (*When the Application requests IT*) - obrázok sa opraví, keď to aplikácia požaduje funkciou UPDATE WORKSTATION a REDRAW ALL STRUCTURES. Medzi volaniami týchto funkcií sa vyskytnú len tie zmeny obrázku, ktoré mohli vzniknúť pomocou metódy určenej režimom modifikácie.

Kým sa obrázok opravuje, určuje v stavovom zozname stanice položka **režim modifikácie,** akou metódou sa vykonávajú zmeny obrázku:

NIVE (*No Immediate Visual Effects*) - nevyskytnú sa žiadne zmeny

UWOR (*Update WithOut Regeneration*) - všetky zmeny obrázku, ktoré môžu byť správne realizované okamžite bez regenerácie alebo simulácie, sa vykonali. Toto zahŕňa (ale neobmedzuje sa len na) zmeny obrázku, ktorých položky "dynamic modification accepted" sú IMM. Žiadne iné zmeny nie sú vykonané.

UQUM (*Use Quick Update Method*) - všetky zmeny obrázka, ktoré môžu byť realizované okamžite správne bez použitia regenerácie spolu s tými, ktoré môžu byť simulované, sú vykonané. Toto zahŕňa (ale neobmedzuje sa len na) zmeny obrázku, ktorých položky *dynamic modification accepted* sú nastavené na IMM a CBS. Žiadne iné zmeny sa nevykonajú.

Režim zdržania aj režim modifikácie nastavíme jediným volaním SET DISPLAY UPDATE STATE (*ws_id, defer_mode, modif_mode*).

19.7 PHIGS PLUS

Na danej ploche nemožno úplne popísať PHIGS+, preto tu neuvádzame presný tvar jednotlivých funkcií, všetky atribúty a ich význam, apod. Niektoré charakteristiky PHIGS+ sú nové výstupné grafické prvky, obsahujúce geometrické aj negeometrické dáta, ďalšie atribúty na riadenie osvetlenia a tieňovania pre všetky výstupné prvky PHIGS aj PHIGS+, všeobecná špecifikácia farby, aj pre neindexovú farbu a ukladanie a zobrazovanie B-splajnových kriviek a plôch

PHIGS+ špecifikuje dodatočnú funkčnosť PHIGS-u, preto treba odchýlky v ich funkčnosti zosúladiť. Napr. funkcia SET COLOUR MODEL je z hľadiska PHIGS+ zbytočná, lebo pri špecifikácii farby PHIGS+ vždy určí aj farebný model aj hodnotu - i napriek tomu však implementácia vyhovujúca PHIGS+ musí obsahovať túto funkciu, lebo PHIGS-ovský program musí pod PHIGS+ bežať rovnako.

Všeobecná špecifikácia farby. PHIGS definuje iba nepriamu farebnú špecifikáciu. PHIGS+ definuje priamu farebnú špecifikáciu a vyžaduje od GS podporu oboch - priamej i nepriamej. Farbu PHIGS+ definuje pomocou pojmu všeobecnej farby, špecifikovanej priamo alebo nepriamo. Všeobecná farba určí typ a hodnotu farby. Typ farby udáva buď farebný model (RGB, CIELUV, HSV a HLS) v zodpovedajúcej ***n***-tici farebných súradníc alebo že farba je špecifikovaná nepriamo - indexom do tabuľky farieb stavového zoznamu pracovnej stanice. Definované rozsahy sú intervaly [0,1] pre každú súradnicu RGB a CIELUV, [0,1] pre súradnice polomeru a výšky HSV a HLS, a všetky reálne čísla pre súradnice uhlov HSV a HLS. V prvkoch štruktúr môžu byť aj súradnice farieb mimo rozsahu pre daný farebný model. Pri prechode štruktúrou sa také hodnoty spracujú na stanici závislým spôsobom.

Ak sa počas prechodu štruktúrou narazí na nepodporovaný dátový typ, tak sa použije dátový typ INDIRECT a hodnota farby 1. Takisto sa implicitne správa farba vo PHIGSe. Nedefinovaný index farby sa považuje za hodnotu 1.

Na podporu špecifikácie všeobecnej farby sú polia stavového zoznamu prechodu XXX COLOUR INDEX (kde XXX je polyline, polymarker, text, interior alebo edge) nahradené zodpovedajúcimi poliami XXX COLOUR. Pre každý prvok štruktúry SET XXX COLOUR INDEX definovaný v časti 1 definuje PHIGS+ zodpovedajúci prvok štruktúry SET XXX COLOUR.

PHIGS+ rozšíril množinu výstupných grafických prvkov o ďalšie, pričom k niektorým z nich môžeme voliteľne pridružiť rôzne údaje. (napr. kombinácia normál vrcholov (*vertex normals*), normál stien (*facet normals*), príznakov viditeľnosti hrán a údajov o vlastnej (*intrinsic*) farbe). Podľa potreby PHIGS+ používa tieto dáta v rôznych fázach **nanášacieho kanála** (*rendering pipeline*) pre osvetlenie, tieňovanie, apod. Kvôli nedostatku miesta uvedieme iba zoznam prvkov bez parametrov. Podrobnosti uvádza [ERTL93]: **skupina lomených čiar s údajmi, skupina výplňových oblastí s údajmi, pole buniek PLUS, skupina skupín výplňových oblastí s údajmi, skupina trojuholníkov s údajmi, pás trojuholníkov s údajmi, sieť štvoruholníkov s údajmi, neuniformná B-splajnová krivka, uniformná B-splajnová krivka s farbou, neuniformná B-splajnová plocha, neuniformná B-splajnová plocha s údajmi**.

Skupiny výstupných prvkov prevzala z PHIGS+ aj inovácia GKS. Všetky nové výstupné prvky PHIGS+ slúžia na podporu modelovania telies, popisu 3D scény a fotorealistického zobrazovania.

19.8 Syntéza PHIGS + X = PEX

Ciele návrhu PEX boli: poskytnúť efektívnu podporu pre PHIGS a značné časti PHIGS PLUS, rozšíriť možnosti X a podporiť viaceré (hardverové) platformy X. PEX rozširuje protokol X a nemusí nutne duplikovať funkčnosť X. Platformy na podporu PEX môžu byť od jednoduchej bitovo mapovanej stanice po dômyslené 3D-displeje. Implementácia by mala využiť výhody hardveru, kdekoľvek to je možné. Niektoré funkcie PHIGS (napr. archivovanie súborov) sa doporučuje ošetriť klientovi, a tak nie sú zahrnuté v protokole. Podobne to vyzerá s pamäťou štruktúr - server ju môže podporovať alebo vyžadovať od klienta (aplikácie). Špecifikácia PEX nepredpisovala vstup, ktorý má byť ošetrený kanálmi mimo protokolu PEX. Takisto sa nepredpisuje reakcia

na zmenu veľkosti okna. Celý protokol PEX je z pohľadu X umožnený mechanizmom rozšírenia protokolu X.

Použitie normy PHIGS v prostredí X vyžaduje určitú kooperáciu oboch systémov i nejakú mieru prispôsobenia. Napr. PHIGS buduje na pracovnej stanici. V neoknovom systéme to je zvyčajne celý displej. Ak ale displej spravuje server X, nie je to principiálne možné. Bežným riešením tohto problému je, že stanica PHIGS zodpovedá oknu X. Vtedy PHIGS kreslí do okna ako ktorákoľvek iná aplikácia. Lenže X dovoľuje veľkosť okna meniť! Vtedy PHIGS musí prinajmenšom prepočítať svoje transformácie na stanicu. A ako riešiť **problém exponovania** (tzv. *expose problem*), že zo stanice PHIGS môže byť čosi zakryté-odkryté? Ďalšou trecou plochou medzi X a PHIGS je **riadenie farieb**. Ešte vážnejším problémom je však, že X môže implementácii PHIGS znemožniť použitie špeciálneho hardveru, napr. 3D-grafiky. Ak zas PHIGS používa takýto hardver priamo, stráca sa jedna z najlepších vymožeností X - sieťová transparentnosť. Z týchto diliem sú principiálne dve východiská. **1.** PHIGS používa X ako 2D grafiku, čiže po vykonaní potrebných 3D výpočtov sa na X obracia len s 2D grafickou informáciou. **2.** Treba rozšíriť server X, aby rozumel funkciám a akciám PHIGS, čím sa však server môže stať ťažkopádnym.

PEX rozširuje X o 9 nových zdrojov (*resources*). Tým sa programátorsky PEX približuje k PHIGS PLUS. Odchýlky predstavuje najmä fakt, že stanicou je okno X a PEX nedefinuje podporu pre vstupné zariadenia PHIGS. PEX dedí od X mnohé ďalšie vlastnosti, ktoré musí programátor brať do úvahy, ale v PHIGS PLUS sa vôbec nevyskytujú, napr. obsluhu udalostí. V súčasnosti existuje niekoľko implementácií PEX, z toho aspoň jedna bez zodpovedajúcej implementácie PHIGS. Implementátori zväčša mapujú vstup X na vstupné zariadenia a režimy PHIGS.

Diskusia s pohľadmi i kritikou z oboch strán (X i PHIGS) pokračuje. Už dnes však vidno, že prístupy oboch systémov sa navzájom relativizujú a hľadá sa riešenie, ktoré spojí maximum výhod oboch. PEX je tak aspoň dvojnásobnou syntézou - oknového a neoknového, nenormalizovaného a normalizovaného. A zároveň poznávacou iteráciou v rozvoji budúcej normy, budúceho jazyka na myslenie o grafike.

Cvičenia

1. Napíšte postupnosť volaní funkcií pozostávajúcu len z OPEN STRUCTURE, EXECUTE STRUCTURE a CLOSE STRUCTURE, ktorá reprezentuje viacúrovňovú hierarchiu na obr. 19.2 a sieť štruktúr na obr. 19.3.
2. Ako možno spraviť pomocou štandardných možností inverzné orezávanie (*shielding*)? Aký typ operátora by mala podporovať implementácia PHIGS-u, aby sa dalo inverzne orezávať jednoduchším spôsobom ?
3. Napíšte v PHIGS-e program znázorňujúci rotáciu trojice úsečiek. Najdlhšia úsečka sa otáča okolo jedného zo svojich koncov, úsečka strednej veľkosti je pripojená na jej druhý koniec, okolo ktorého sa otáča, a najkratšia úsečka je napojená na jej voľný koniec, okolo ktorého sa otáča.
4. Znázornite časť plochy popísanej rovnicou z = f(x,y) pre niekoľko funkcií f(x,y).
5. Vykreslite drôtový model tórusu.
6. Modelujte a animujte drôtový model elipsoidu.
7. Modelujte a zobrazte 14-sten (vzniknutý zrezaním vrcholov kocky) tak, aby žiadne dve steny nemali rovnakú farbu.

20

Multimédiá

20.1 Architektúra multimediálneho systému

Dosiaľ popisované systémy mali za cieľ maximálne efektívne spracovať grafickú informáciu. Metódy a pojmy počítačovej grafiky a spracovania obrazu spadajú do širšieho rámca multimediálnej technológie, kde sa často využívajú. Multimediálna stanica by mala integrovať funkčnosť počítača s funkčnosťou videorekordéra.

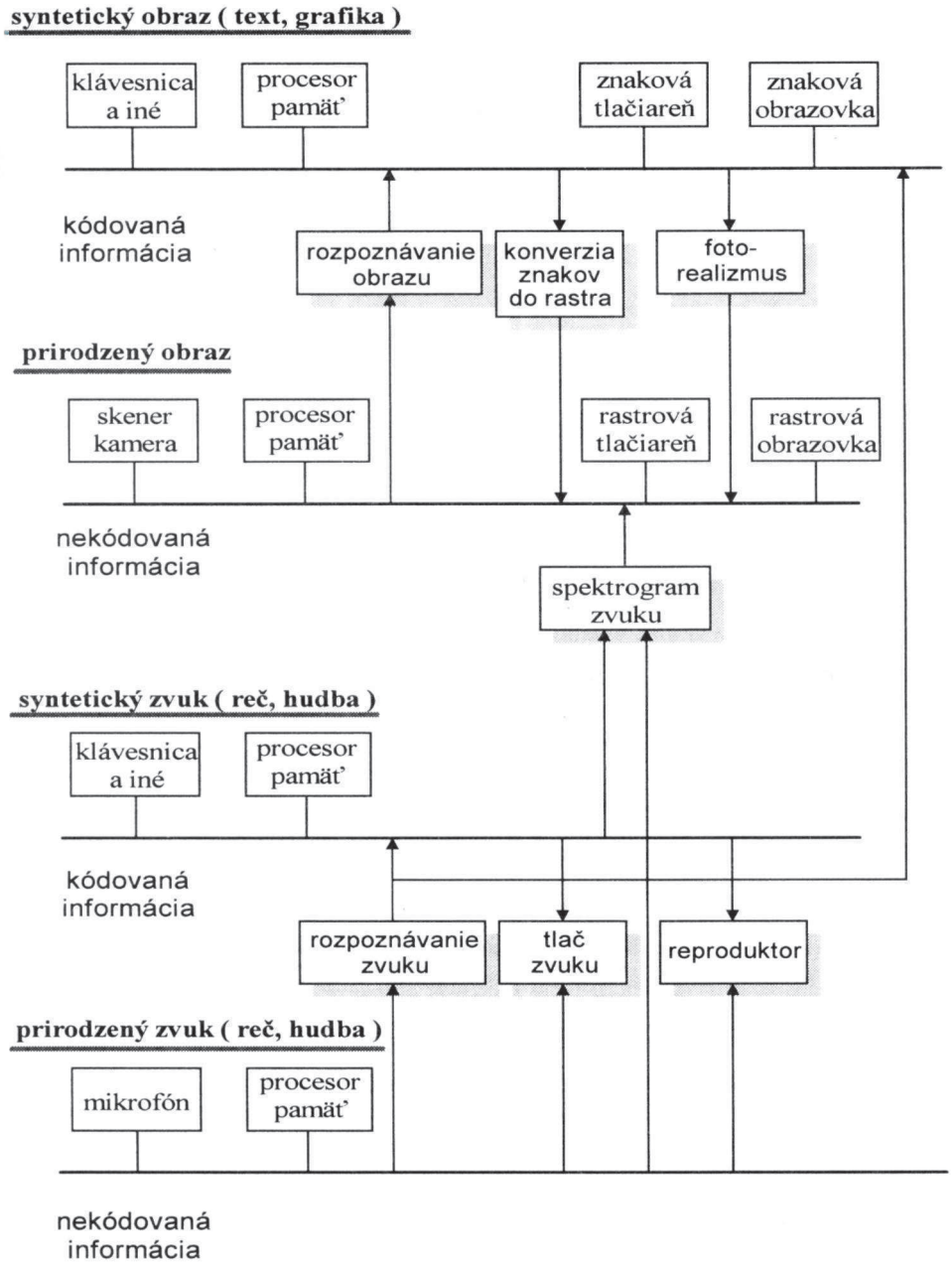
Multimédiá definuje ISO ako **vytváranie, úpravu, skladanie a/alebo prezentáciu produktov pozostávajúcich z akejkoľvek kombinácie médií**. Médium je prostriedok, ktorým sa informácia vníma, vyjadruje, ukladá alebo prenáša.

Multimédiá možno charakterizovať **oblasťami záujmu** (*domains of interest*), **základnými komponentami** (*key-elements*) a **funkčnými blokmi** resp. jednotkami (*functional units*) a štruktúrou inštalácie, [STUC91].

Oblasti záujmu sú počítačová grafika, spracovanie obrazu, animácie, konverzia a ukladanie obrazu, spracovanie zvuku a multimediálne/hypermediálne (mm/hm) aplikácie.

Základnými komponentami sú procesory, optické pamäťové médiá na čo najlacnejšie ukladanie enormne rozsiahlych súborov obrazovej i zvukovej informácie. Dôležitým komponentom multimédií sú normy na kódovanie a prezentáciu mnohorakých foriem informácie: textu, dát, obrazu, zvuku a hudby, video, animácií, operácií a metód. Všetky dátové typy môžeme rozlišovať podľa kódovanosti, štrukturovanosti a formátovanosti. Multimédiá dedia všetky obmedzenia jednotlivých médií, napr. problém kompresie, no majú aj ďalšiu úroveň zložitosti, ktorá vzniká práve kombináciou médií.

Obrázok 20.1 ilustruje druhy dát a ich toky v typickej architektúre multimediálneho systému, [STUC91]. Štyri procesory, pamäti a disky v multimediálnom systéme môžu znamenať aj jediné fyzické zariadenie svojho druhu, ale treba zdôrazniť, že logicky pre každú dátovú zbernicu zo štyroch (**syntetický a prirodzený obraz, syntetický a prirodzený zvuk**) ide o iné spracovanie, čomu už zodpovedajú ďalšie špecializované zariadenia. Podrobný rozpis funkčných blokov pre počítačovú grafiku a spracovanie obrazu sme uviedli v prvej kapitole. V multimediálnom systéme k nim patrí aj funkčný blok pre spracovanie zvuku, znázornený na obr. 20.2. Príbuzné oblasti sú **hlasový výstup** a **rozpoznávanie zvuku**, ktoré je však už zahrnuté do spodnej časti obr. 20.2. Funkčný blok pre hlasový výstup by sa ponášal na zjednodušenie obrázku 20.2, s obrátenými šípkami. Analógie so spracovaním grafickej informácie vidno na obr. 1.4 v kapitole 1.



Obr. 20.1 Architektúra multimediálneho systému

20.2 Spracovanie zvuku

Od doteraz popísaných metód sa spracovanie informácie, potrebné pre multimediálnu aplikáciu, principiálne odlišuje spracovaním zvuku, pri ktorom sa preto pristavíme podrobnejšie.

Spracovanie zvuku príslušnými funkciami si vyžaduje vhodné kódovanie. Medzinárodná norma sa nazýva **MIDI** (*Musical Instruments Digital Interface*) a umožňuje vzájomné prepojenie syntetizátorov, sekvencerov, osobných počítačov, rytmerov, atď. MIDI normuje rozhranie, v podstate je to upravené počítačové rozhranie RS-232. Každý nástroj obyčajne obsahuje prijímač a vysielač. MIDI normuje nielen **formát dátového súboru** na kódovanie zvuku, ale aj **hardver**, oi. dvojito odtienené káble s maximálnou dĺžkou 15 metrov, zakončené päťkolíkovým konektorom. Okrem kódovania MIDI existuje mnoho neštandardných kódovaní, avšak MIDI je svetová priemyselná norma, ktorá od svojho uverejnenia v roku 1985 (takmer súčasne s GKS) prakticky odstránila problémy s kompatibilitou. Nepochybne aj vďaka vtedy prekvapivej dohode popredných svetových výrobcov hudobnej elektroniky.

Podobne ako pri vytváraní obrázka a spracovaní obrazu rozlišujeme **zvuk syntetický** (kódovanú informáciu) a **zvuk prirodzený** (nekódovanú informáciu). Za zvuk považujeme reč, hudbu a niekedy aj šum. Napr. pre syntetickú hudbu poznáme jej štruktúru, napr. notopis, kým v zázname nasnímanej (nekódovanej) hudby máme zachytenú len jej znejúcu podobu, aj keď zachytenú v digitálnej konvencii.

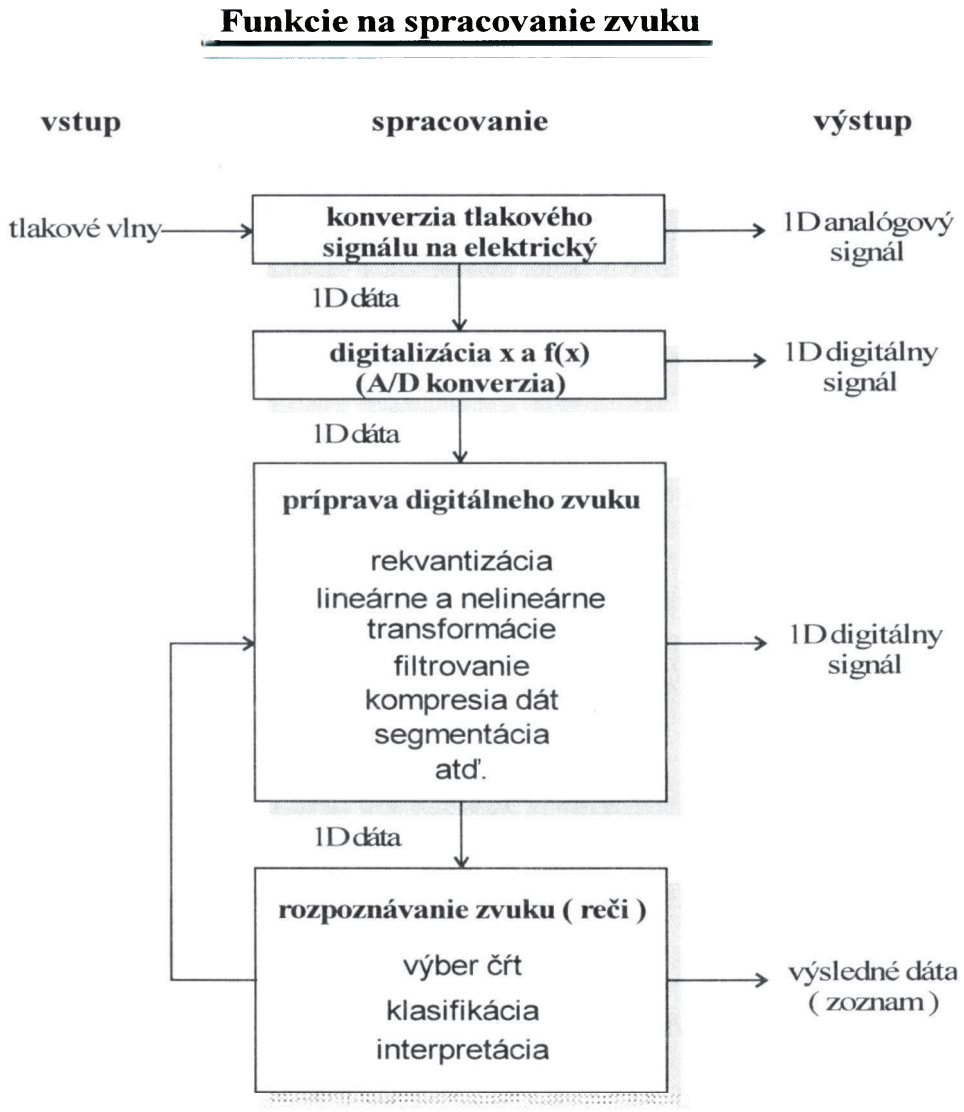
Na analýzu hudby je potrebné aj kódovanie analytickej informácie, napr. zápis akordov, tonálnych úsekov a harmonických funkcií. Na vyjadrenie takýchto poznatkov nestačí notopis. Tomuto a ďalším cieľom, súvisiacim s hudbou, slúži aplikácia všeobecnejšej normy **HyTime** (ISO/IEC 10744), nazývaná **Standard Music Description Language** (**SMDL,** ISO/IEC 10744), ktorá umožňuje aj vyjadrenie notového záznamu v textovom kódovaní, zachytenie viacerých interpretácií tej istej skladby a ďalšie potrebné s hudbou súvisiace informácie. Existuje už napr. program na konverziu dokumentu SMDL do jazyka povelov na riadenie syntetizátora.

Možno postrehnúť určité analógie funkčnosti pri spracovaní zvuku s funkčnosťou pre počítačovú grafiku a spracovanie obrazu, no pri zvládaní zložitosti multimediálneho prostredia sú jednotlivé druhy spracovania oddelené a zmyslovo bohatý multimediálny efekt vzniká až u operátora. Človek má viac vstupných kanálov resp. druhov vnímania informácie a multimédiá sú cestou, ako maximálne skvalitniť komunikáciu človek-stroj.

Pre multimediálne GUI pribúdajú nové požiadavky, napr. **WYPIWYP** (*what you play is what you print*), tj. tlač nôt, ktoré hrá skladateľ na klaviatúre syntetizátora. V animácii sa na susediacich obrázkoch vyskytujú javy analogické aliasingu a koherencii. Vznikajú aj kuriózne pojmy, napr. vykreslenie zvuku (*sound rendering*), kde prezentácia audio média zdedila zaužívaný a pôvodne čisto grafický pojem.

Problémom zostáva **inportabilita**, neprenositeľnosť softveru z jednej na inú multimediálnu inštaláciu. Systémovým riešením, na ktorom sa intenzívne pracuje, budú medzinárodné normy. Na úrovni kódovania sú to už v kapitole 18 spomenuté normy MHEG a HyTime resp. HyperODA. Veľkú pozornosť vzbudzuje vývoj normy na **prezentáciu multimediálnych objektov** - PREMO. Na záver odbočenia do spracovania zvuku pripomeňme, že animácia (syntetický film) ani video (nasnímaný film) už neprinášajú z

hľadiska spracovania informácie novú kvalitu média, lebo kombinujú v čase obraz a zvuk. Novú kvalitu média prináša skôr výskum kódovania tanečnej informácie.



Obr. 20.2 Funkcie na spracovanie zvuku

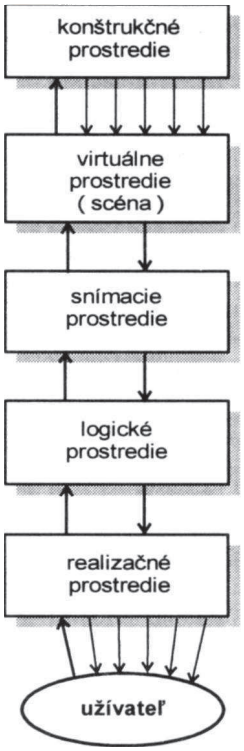
20.3 PREMO

Multimediálny objekt je objekt obsahujúci jeden alebo viac typov médií, ktoré možno prezentovať užívateľovi. **Prezentácia** je transformácia média do formy vhodnej pre pozorovateľa. Jedným z prototypov multimediálneho objektu je miestnosť, v ktorej sa v reálnom čase modelujú všetky bežné informačné zariadenia a ich funkcie - televízor ukazujúci aktuálny program, počítač, telefón, rádio, hodiny, videoprehrávač...

PREMO je anagram ***Presentation Environment for Multimedial Objects*** (ISO/IEC CD 14782). Tento projekt vznikol z projektu PREGO (G skracovalo Graphical). Táto budúca norma integruje objektovo orientovanú technológiu a multimédiá, pričom sa má vyhovieť prezentačným požiadavkám CAD/CAM, medicínskeho zobrazovania, virtuálnej reality ai., a takým prezentačným technikám ako animácia, súčasné používanie viacerých médií, GUI, vedeckotechnická vizualizácia, simulácia, apod. Motívom na rýchlu normalizáciu technológií, ktoré sa v mnohých ohľadoch ešte len vyvíjajú, je zložitosť multimédií. Problém inportability na úrovni kódovania i prezentácie multimédiálnych aplikácií sa ukazuje byť obzvlášť masívnou bariérou ďalšieho rozvoja. PREMO využíva na popis rozhrania medzi objektami IDL (Interface Design Language). Objekt PREMO vychádza z vlastností definovaných v dokumente Common Object Request Broker, ktorý v roku 1992 vydala **OMG** (Object Management Group). V podstate ide o mechanizmus, ktorým objekty transparentne vysielajú požiadavky a dostávajú odpovede. Zmyslom je dosiahnúť súčinnosť medzi aplikáciami na rôznych strojoch v distribuovaných prostrediach. Objekty PREMO sú dynamické, vytvárajú sa programovaním, ich životný cyklus je od vytvorenia po deštrukciu, pričom možno meniť ich referenciu (a pravdepodobne bude možné meniť aj ich typ). PREMO objekty sú inštanciami typov a možno ich špecializovať podtypmi s dovolenou viacnásobnou dedičnosťou. Majú byť schopné reprezentovať model akéhokoľvek druhu entity, napr. osobu, loď, dokument, časť obrázku, hodnotu farby, a reagovať na operátorov vstup. PREMO podporuje aj neobjektové typy, napr. celé čísla alebo udalosti.

Návrh normy má 4 časti - **Fundamentals of PREMO, Foundation Component, Modeling and Presentation Component, Multimedia System Services Component**. Norma je v prudkom vývoji. Diskutuje sa napr. či má mať špeciálnu časť **PREMO Windows**. PREMO je jedným z pokusov zachytiť včas rýchly vývoj a ponúknuť firmám i trhu kvalitnú koncepciu programovania multimediálnej prezentácie. Kým funkčné normy GKS a PHIGS reprezentujú **1. generáciu noriem**, rozvíja sa metodológia pre **2. generáciu noriem**. Túto etapu možno datovať od roku 1992 a charakterizuje ju príklon k objektovej technológii programovania a integrácia počítačovej grafiky s inými médiami. PREMO na rozdiel od budovania predchádzajúcich grafických noriem ako uzavretých monolitov sa koncipuje ako otvorená norma, adaptabilná na nové technológie, čím by sa malo predísť zastarávaniu. PREMO pre programovanie grafiky popisuje ešte detailnejšie rozlíšené úrovne abstrakcie medzi aplikáciou a operátorom: **konštrukčné prostredie** na modelovanie, **virtuálne prostredie** pre grafické prvky nezávislo na zariadení, **snímacie prostredie**, **logické prostredie**, kde sa naviažu na zariadení závislé atribúty a **realizačné prostredie**, kde sa tvorba obrázku skončí a napr. konzultáciou v tabuľke farieb vzniká definitívny obraz. Týchto 5 prostredí je užitočných na popis postupnej transformácie geometrických súradníc z aplikačného modelu až do súradníc fyzického zariadenia. Multimediálne dáta prechádzajú tie isté vrstvy spolu s grafickými dátami.

Interaktívny vstup prechádza týmito vrstvami v opačnom poradí. Vstupom v aplikácii PREMO však môže byť hoci záznam z videokamery alebo mikrofónu.



Obr. 20.3 Multimediálna scéna sprostredkuje objekty realizácii

Mediátorom medzi rôznymi komponentami modelovania a prezentácie je **scéna**. Upozornime, že už nie grafická, ale **multimediálna scéna**. Zdroje modelov môžu byť rozličné: modeler telies, animačný systém, modeler na fyzikálnej báze, virtuálna realita alebo modelery vytvárajúce multimediálne resp. hypermediálne modely z dokumentov podľa noriem MHEG alebo HyperODA. Zo všetkých takýchto zdrojov multimediálnych objektov sa komponujú **objekty scény**, ktoré sa prezentujú operátorovi prostredníctvom rendererov potrebného zamerania: 2D, 3D, zvuk, video, a iné.

20.4 Dve multimediálne aplikácie

Multimediálne a hypermediálne aplikácie integrujú a skladajú rôzne médiá na riešenie daného aplikačného problému. Uvedieme príklady aplikácií v noninvazívnej diagnostike [STUC91] a v komunikáciách [PAVL95].

Automatická rekonštrukcia povrchu 3D objektov z CT rezov (získaných počítačovou tomografiou) má dve fázy.

1. Pre každý 2D tomogram sa vykoná zistenie obrysu, stenčovanie a segmentácia obrysu. Výstupom z tejto fázy je aproximácia obrysu hľadaného objektu v danom 2D reze, teda nejaký polygón.

2. Obrysy v susedných rezoch sa triangulujú, čím vznikne 3D teleso, reprezentované triangulovaným povrchom. Takto sa získa drôtový model, ktorý možno vytieňovať a následne animovať. Už v tejto mediálnej prezentácii môže lekár diagnostikovať podľa rekonštruovaného objektu. Ak je objekt priveľmi zložitý, možno ho previesť do ďalšej mediálnej reprezentácie buď prostredníctvom stereoskopického zobrazenia alebo výrobou stereolitogramu, čím vznikne 3D model z umelej hmoty, ktorý možno ďalej už fyzicky - nie informačne - spracovávať alebo využiť vo vyučovaní diagnostiky.

World Wide Web - hypermediálna globálna komunikácia. Spoločnosť na prahu 21. storočia čím ďalej tým viac závisí na informácii prenášanej cez telekomunikačné siete. V roku 1969 sa podarilo preniesť prvú elektronickú správu na sieti ARPANET, ktorá sa dodnes rozrástla na globálnu sieť **INTERNET**, prepájajúcu desiatky milónov užívateľov. Elektronická pošta (e-mail) a ďalšie služby (ftp, telnet), založené na modeli klient-server, sa čoskoro začali používať na prenos obrázkov i hudby. V roku 1989 Tim Berner-Lee prišiel s ideou využiť hypertext na uľahčenie sieťovej komunikácie. Podstatou nevídaného úspechu projektu **WWW** (*World Wide Web*) bolo znova odformalizovanie práce s informáciou, GUI pre túto aplikáciu. Užívatelia tohto druhu komunikácie nepotrebujú študovať nijaký formalizmus, ani hľadať sieťové adresy. Stačí im zvoliť si v hypertexte, tj. texte zovšeobecnenom o ďalšie médiá a o systém odkazov (*hyperlink*), príslušné kľúčové "slovo", teda hoci aj časť obrázku. (Dokument, obsahujúci hyperlinky na iné ako textové informácie, sa nazýva hypermediálny.) Požadovaná multimediálna informácia sa vyhľadá a prenesie.

Hyperdokumenty vo WWW sa kódujú v jazyku **HTML** (*HyperText Markup Language*), súvisiacom s normou **SGML** (*Standard Generalized Markup Language*). Web server a Web klient komunikujú spolu cez **HTTP** *(HyperText Transmission Protocol)*, ale všetko kódovanie i komunikácia ostane pre bežného užívateľa skryté - Web klient, napr. populárny Mosaic, od užívateľa vyžaduje naozaj iba interaktívnu prácu s myšou a klávesnicou. Web klient sám podľa hyperlinku vyhľadá príslušný server a ten na požiadanie odošle hľadanú informáciu. Užívateľ sa v žargóne nazýva "surfer", lebo pomocou WWW "surfuje" na informačnom "oceáne", hnaný vetrom vlastných volieb.

Práve popísaná aplikácia je jednou z tých, ktoré opodstatňujú prognózy o vzniku **globálnej informačnej autostrády** (*global information highway*). Počet Web serverov sa podľa [PAVL95] každých 56 dní zdvojnásobuje...

20.5 Najnovšie trendy

Popri vývoji PREMO sa intenzívne pracuje aj v iných smeroch rozvoja myslenia o spracovaní obrazovej a multimediálnej informácie. Očakávajú sa špecifikácie **New API** (*Application Programming Interface*) - prvej normy grafického systému 2. generácie. Možno čakať, že na rozvoj budúcich noriem budú mať silný vplyv nasledujúce rapídne sa rozvíjajúce metodológie: ***Windowing, Object Oriented Programming, Hypermedia a Multimedia, Modern User Interface Methodologies, Modern Hardware Facilities*** (vrátane 3D), protokoly v súlade s **OSI** (*Open Systems Interconnection*) a kompati-

bilitou medzi normami. V týchto trendoch však PREMO zostane jedným z vedúcich projektov s cieľom anticipovať ďalší vývoj myslenia o grafike v rámci multimediálnej technológie.

Na trhu sa prejavuje nesporný nárast multimediálnych i pseudomultimediálnych aplikácií. Predpovedá sa napr. revolúcia vo vzdelávaní na báze multimédií. Na trhu sú už prvé multimediálne a hypermediálne CD namiesto kníh a MHEG karty na dekódovanie filmov na obrazovku a zvukovú kartu počítača. Zatiaľ sú však multimediálne aplikácie veľmi drahé a vízia osobného počítača s kompletnou multimedialitou za masovo prijateľné ceny naráža na množstvo obmedzení.

Príkladom neuspokojivo vyriešených praktických problémov je prenos dynamických dát (zvuk, video) na väčšie vzdialenosti a kooperatívna multimediálna práca, tj. problémy pri prechode multimediálnej technológie do otvorených a distribuovaných prostredí.

Spracovanie informácie je, a vždy bolo, základom každého podnikania - v priemysle, ekonomike, armáde, vede, školstve, politike, umení a sociálnej sfére. Od polovice storočia možno hovoriť o automatizácii spracovania informácie. Tri základné kvalitatívne zmeny umožnili vznik modernej informačnej technológie: objav **mikroprocesora**, rozvoj **komunikácií** a koncepcia **otvorených systémov** (open systems), ktorá vznikla v 70. rokoch.

V tomto rámci možno predpovedať vo všetkých aplikačných oblastiach hľadanie maximálneho prínosu z multimedializácie. Z tohto hľadiska stojíme na prahu prevratných zmien v našich komunikačných systémoch.

Literatúra

[AGOS79] Agoston G.A.: Color Theory and its Application in Art and Design, Springer-Verlag, - Berlin, 1979.

[AMME85] Ammeraal L.: Programming Principles in Computer Graphics, John Wiley & Sons, New York 1988. (U)

[ANGE81] Angel I.O.: A Practical Introduction to Computer Graphics, New York, 1981. (*)

[APPE68] Appel A.: Some Techniques for Shading Machine-Renderings of Solids, SJCC 1968, Thompson Books, Washington, D.C., pp. 37-45.

[ARBO88] Arnold D. B., Bono P. R.: CGM and CGI, Metafile and Interface Standards for Computer Graphics, Springer-Verlag, Berlin, 1988. (U)

[ATWG78] Atherton P., Weiler K., Greenberg D.: Polygon Shadow Generation, SIGGRAPH'78 proceedings, Computer Graphics, 12(3), 1978, pp. 275-281.

[AUBL94] Autstalaknis S., Blatner D.: Reálně o virtuální realitě, JOTA Brno 1994 (Angl. originál Silicon Mirage, Peachpitt Press 1992).

[BFJN90] Balogh G., Ferko A., Jankovič V., Niepel Ľ., Ružický E.: SYSTÉM COMGRAPHSYS - Algoritmická časť, in: Zborník ALGORITMY 1991.

[BARS88] Barsky, B.: Computer Graphics and Geometric Modelling Using Beta-splines, Springer-Verlag, Berlin, 1988.

[BART87] Bartels R., Beatty J., Barsky B.: An Introduction to Splines for Use in Computer Graphics and Geometric Modeling, Morgan Kaufmann, Los Altos, CA, 1987.

[BERN85] Bernhard A.: ICONS and User-Interface Communication, Computer Graphics, Visual Technology and Art, Proceedings edited by T. L. Kunii, Springer, Tokyo 1985.

[BEZI72] Bézier, P.: Numerical Control: Mathematics and Applications, Wiley, Chichester, UK, 1972.

[BLIN77] Blinn, J.F.: Models of light reflection for Computer Synthesised Pictures, Computer Graphics, 11(2), 1977.

[BURG89] Burger P., Gillies D.: Interactive Computer Graphics, Addison-Wesley,1989.

[BOUK70] Bouknight W.J.: A Procedure for Generation of Three-Dimensional Halftoned Computer Graphics Representations, CACM, 13(9), Sep 1970, pp. 527-536.

[BOZE91] Božek M.: Smoothing Multiplication of Curves, JŠPG, Bratislava 1991

[BRES65] Bresenham J.E.: Algorithm for Computer Control of Digital Plotter, IBM System Journal, 4(1), 1965, pp. 25-30.

[BRES77] Bresenham J.E.: A Linear Algorithm for Incremental Digital Display of Circular Arcs, Communications of the ACM, 20(2), Feb 1977, pp. 100-106.

[BUIT75] Bui-Tuong, Phong: Illumination for Computer-generated Pictures, Communications of the ACM, 18(6), 1975.

[CARP84] Carpenter, L.C.: The A-buffer, an Anti-aliased Hidden Surface Method, Computer Graphics, 18(3), 103, 1984.

[CAST85] Casale M.S., Stanton E.L.: An Overview of Analytic Solid Modeling, IEEE Computer Graphics and Applications, 5(2), pp. 45-56.

[CATM78] Catmull E.: A Hidden-Surface Algorithm with Anti-Aliasing, SIGGRAPH'78 Proceedings, Computer Graphics, 12(13), Aug 1978, pp. 6-11.

[CATM80] Catmull E., Smith A.R.: 3-D Transformations of Images in Scanline Order, SIGGRAPH'80 Proceedings, Computer Graphics, 14(3), 1980, pp. 279-285.

[CLAR76] Clark J.H.: Hierarchical Geometric Models for Visible Surface Algorithms, CACM, 19(10), pp. 547-554, 1976.

[COHE69] Cohen D., Lee T.M.P.: Fast Drawing for Curves for Computer Display, SJCC 1969, AFIPS Press, Montvale, N.J., pp. 297-307.

[COHE85] Cohen M.F., Greenberg D.P.: The Hemi-Cube: A Radiosity Solution for Complex Environments, SIGGRAPH'85, pp. 31-40, 1985.

[COHE86] Cohen M.F., Greenberg D.P., Immel D.S.: An Efficient Radiosity Approach for Realistic Image Synthesis, IEEE Computer Graphics and Appl., 6(2), pp. 26-35, 1986.

[COHE88] Cohen M.F et al.: A Progressive Refinement Approach to Fast Radiosity Image Generation, SIGGRAPH'88, pp. 75-84, 1988.

[COMU89] Computing in Musicology, A Directory of Research, Menlo Park 1989.

[COOK82] Cook, R.L., Torrance, K.E.: A Reflectance Model for Computer Graphics, ACM Trans. on Graphics, 18(3), pp. 137-144, 1982.

[COPC84] Cook, R.L., Porter T., Carpenter L.: Distributed Ray Tracing, SIGGRAPH'84 Proceedings, Computer Graphics, 18(3), pp. 137-145.

[CROW81] Crow, F.C.: A Comparison of Antialiasing Techniques, IEEE Computer Graphics and Applications, 1(1), pp. 40-49.

[CYRU78] Cyrus, M., Beck, J.: Generalized Two- and Three-Dimensional Clipping, Computer and Graphics, 3(1), 1978, pp. 23-28.

[DRS_84] Drs L.: Plochy ve výpočetní technice, SNTL, Praha 1984.

[DUHA68] Duda R.D., Hart P.E.: Experiments in the Recognition of Hand-printed Text, II, Context Analysis, FJCC 1968, Thompson Books, Washington, D.C., pp. 1139-1150.

[EARN88] Earnshaw R.A. (Ed.): Theoretical Foundations of CG and CAD, Springer 1988.

[ENCA80] Encarnacao J., et al.: The Workstation Concept of GKS and the Resulting Conceptual Differences to the GSPC Core System, Computer Graphics, 14(3), July 1980, pp. 226-230.

[ENKP84] Enderle G., Kansy K., Pfaff G.: Computer Graphics programming: GKS - The Graphics Standard, Springer-Verlag Berlin, 1984, 1987. (*)

[ERTL88] Ertl J.: Úvod do normy PHIGS, pp. 29-37, JŠPG Stará Turá 1988

[ERTL91] Ertl J.: New Primitives and Attributes in the CGM, pp. 28-34, JŠPG Bratislava 1991

[ERTL93a] Ertl J.: The Short Summary of Meeting Results for the 6th ISO/IEC JTC1 SC24 Plenary, Chiemsee, Oct 1992, in: Computer Graphics'93, Budmerice 1993

[ERTL93] Ertl J., Ferko A.: Normalizované grafické systémy, MFF UK, Bratislava 1993 (S)

[FAPR79] Faux I.D., Pratt M.J.: Computational Geometry for Design and Manufacture, John Wiley, New York, 1979. (*)

[FARI90] Farin, G.: Curves and Surfaces for Computer Aided Design, 2nd edn., Academic Press, Boston, 1990.

[FIUM89] Fuime, E.L.: The Mathematical Structure of Raster Graphics, Academic Press, San Diego, CA, 1989.

[FKOU88] Ferko A., Kossaczký I., Učníková D.: Norma CGM, pp. 38-41, JŠPG Stará Turá 1988

[FOVD82] Foley J.D., van Dam A.: Fundamentals of Interactive CG, Addison-Wesley 1982. (U)

[FOLE90] Foley J.D., van Dam A., Feiner S., Hughes J.: Computer Graphics: Principles and Practice, Second Edition, Addison-Wesley 1990 (U)

[FVFH93] Foley J.D., van Dam A., Feiner S., Hughes J., Phillips R.L.: Introduction to Computer Graphics, Addison-Wesley 1993 (U)

[FOWA74] Foley J.D., Wallace V.L.: The Art of Natural Graphic Man-Machine Conversation, Proceedings IEEE, 62(4), Apr 1974, pp. 462-470.

[FOLE80] Foley J.D.: The Structure for Interactive Command Language, in: Guedj R.A. et al., Methodology of Interaction, North-Holland, Amsterdam, 1980, pp. 227-234.

[FOUR82] Fournier A., Fussel D., Carpenter L.: Computer Rendering of Stochastic Models, CACM, 25(6), pp. 371-384, 1982.

[FOWE81] Foley J.D., Wenner P.A.: The George Washington University Core System Implementation, SIGGRAPH'81, 15(3), pp. 123-131, Aug 1980.

[FRLO67] Freeman H., Loutrel P.P.: An Algorithm for the Solution of the Two-Dimensional Hidden-line Problem, IEEE Trans. on Computer, EC-16(6), Dec 1967.

[FUJI85] Fujimura K., Kunii T.L.: A Hierarchical Space Indexing Method, Computer Graphics Conference, pp. 21-34, Tokyo, Springer 1985.

[FUJI86] Fujimoto, A., Tanaka, T., Iwata, K.: Accelerated Ray Tracing System, IEEE Computer Graphics and Applications, 6(4), pp. 16-26, 1986.

[FURO84] Fu K.S., Rosenfeld A.: Pattern Recognition and Computer Vision, 17(10), pp. 274-282.

[GAJK90] Galbavý R., Jajcayová T., Kossaczký I.: Systém COMGRAPH - demonštračná časť, in Zborník ALGORITMY'91, Vysoke Tatry 1991.

[GAMT86] Gašpar D., Mederly P., Tvarožek J.: Dátové štruktúry a hierarchia v systéme ISAN2, Návrh obvodů počítačem, Praha 1986.

[GILO78] Giloi W.: Interactive Computer Graphics - Data Structures, Algorithms, Languages, Prentice-Hall, 1978. (*)

[GLAS84] Glassner A.S.: Space Subdivision for Fast Ray Tracing, IEEE Computer Graphics and Applications, 4(4), 1984.

[GLAS89] Glassner A.S.: An Introduction to Ray Tracing, Academic Press, London, 1989.

[GONZ87] Gonzales R.C., Wintz P.: Digital Image Processing, Addison-Wesley, Mass., 1987.

[GOUR71] Gouraud H.: Continuous Shading of Curved Surfaces, IEEE Transactions on Computers, C-20(6), June 1971, pp. 623-628.

[GRAN89] Granát L.: Development of Computer Graphics Standards, Počítačová grafika'89, Smolenice 1989

[GRIF78] Griffiths J.G.: A Bibliography of Hidden-Line and Hidden-Surface Algorithms, Computer Aided Design, 10(3), May 1978, pp. 203-206.

[GUID88] Guid N.: Računalniška grafika, Maribor 1988. (U)

[HALL83] Hall, R., Greenberg, D.P.: A Testbed for Realistic Image Synthesis, IEEE Computer Graphics and Applications, 3(8), Nov 1983.

[HALL89] Hall R.: Illumination and Color in Computer Generated Imagery, Springer, New York, 1989.

[HALZ87] Halzl O.: Grafické systémy a grafické normy, SVŠT Bratislava 1987. (S)

[HAGE77] Hamlin G., Gear C.: Raster-Scan Hidden Surface Algorithm Techniques, Proceedings, Computer Graphics, 11(2), pp. 206-215, 1977.

[HARR83] Harrington S.: Computer Graphics: A Programming Approach, New York, McGraw-Hill, 1983. (U)

[HARR87] Harrington S., Computer Graphics, McGraw - Hill, New York 1987. (U)

[HEBA86] Hearn D., Baker M.P.: Computer Graphics, Prentice-Hall, London, 1986. (U)

[HEBA91] Hearn D., Baker P.: Scientific Vizualization, EUROGRAPHICS'91, Vienna 1991.

[HEAR94] Hearn D. - Baker M. P.: Computer Graphics, Prentice Hall 1994.

[HESE90] Hewlett W.B., Selfridge-Field E. (Eds.): Computing in Musicology, A Directory of Research, Menlo Park, CA, Oct 1990.

[HLAV92] Hlaváč V., Šonka M.: Počítačové vidění, Grada, Praha 1992.

[HOPG86] Hopgood F.R.A., Duce D.A., Gallop J.R., Sutcliffe D.C: Introduction to the Graphical Kernel System (GKS), Academic Press 1983, 1986. (U)

[HOWA91] Howard T.L.J. et al: A Practical Introduction to PHIGS and PHIGS PLUS, Addison-Wesley 1991. (U)

[HUDE93] Hudec B.: Základy počítačové grafiky, Vydavatelství ČVUT, Praha 1993. (S)

[ISO646] ISO 646 - 7-bit Coded Character Set for Information Interchange, ISO 1983.

[IS2022] ISO 2022 - ISO 7-bit and 8-bit Coded Character Sets - Code Extension Techniques, ISO 1983.

[IS7942] ISO 7942 Graphical Kernel System [GKS] - Functional Description, ISO 1985.

[IS8632] ISO/IEC 8632: Information Processing Systems - Computer Graphics Metafile for the Storage and Transfer of Picture Description Information, ISO 1992.

[IS8632c] ISO/IEC 8632-1:1987: Computer Graphics Metafile (CGM) for the Storage and Transfer of Picture Description Information, ISO 1987.

[IS8651] ISO 8651 - GKS Language Bindings, ISO 1988.

[IS8805] ISO/DIS 8805 - Graphical Kernel System for Three Dimensions (GKS-3D), ISO 1987.

[IS9592] ISO 9592 - Programmer's Hierarchical Interactive Graphics System (PHIGS), ISO 1989

[IS9636] ISO 9636 - Computer Graphics Interface (CGI), ISO 1991.

[I12087] ISO/IEC 12087:1994 Image Processing and Interchange, ISO 1994

[I14782] CD ISO/IEC 14782:1994 Presentation Environment for Multimedia Objects (PREMO), ISO 1994

[KAJI83] Kajiya, J.T.: New Techniques for Ray Tracing Procedurally Defined Objects, Computer Graphics, 18(3), pp. 165-174, 1983.

[KAJI86] Kajiya, J.T.: The Rendering Equation, Computer Graphics, 20(4), pp. 143-150, 1986.

[KAY_86] Kay T.L., Kajiya J.T.: Ray Tracing Complex Scenes, SIGGRAPH'86, pp. 269-278, 1986.

[KAPL85] Kaplan, M.R.: Space Tracing, a Constant Time Ray Tracer, SIGGRAPH'85 Tutorial, San Francisco, 1985.

[KRCJ91] Krč-Jediný, J.: ISO/IEC JTC1/SC24 (Computer Graphics) Plenary London, Apr 22-23 1991 - Results and Trends, pp. 90-95, JŠPG, Bratislava 1991.

[KNUT73] Knuth D.E.: The Art of Computer Programming, Addison-Wesley, 1973.

[KRCJ92] Krč-Jediný J.: ISO/IEC JTC1/SC24 Plenary Report, Amsterdam, Feb 92, pp. 84-88, JŠPG Bratislava 1992.

[KRME87] Krč-Jediný J., Mederly P.: Vývoj medzinárodných noriem v počítačovej grafike, JŠPG Richňava 1987.

[KRME88] Krč-Jediný J., Mederly P.: Medzinárodné normy v počítačovej grafike, in: OOP a modelovanie systémov, pp. 98-109, Alfa Bratislava 1988.

[KRYL78] Krylov A. A.: Praktikum po inženernoj psichologii, Leningrad 1978.

[KWOK88] Kwok, P.C.K.: A thinning algorithm by contour generation, CACM 31, 11, 1988.

[LIAN84] Liang, Y.D., Barsky, B.: A New Concept and Method for Line Clipping, ACM TOG, 3 (1), Jan, 1984, pp. 1-22.

[LIMP94] Limpouch A.: Graphical User Interfaces, JŠPG Bratislava 1994.

[LOUT70] Loutrel P.P.: A Solution to the Hidden-Line Problem for Computer-drawn Polyhedra, I-EEE Transactions on Computers, EC-19(3), March 1970, pp. 205-213.

[LUGA83] Lucas M., Gardan Y.: Techniques Graphiques Interactives et C.A.O., Hermes Publishing, Paris, 1983. (*)

[MAND77] Mandelbrot B.: Fractal Geometry of Nature, Freeman and Co., New York 1977.

[MEFG88] Mederly P., Ertl J., Ferko A., Gašpar D., Krč-Jediný J.: Grafické systémy, ČSVTS, Bratislava 1988. (S)

[MIDA78] Michener J.C., van Dam A.: A Functional Overview of the Core System with Glossary, Computing Surveys, 10(4), 1978, pp. 445-464.

[MICA80] Michener J.C., Carlbom I.B.: Natural and Efficient Viewing Parameters, SIGGRAPH'80 proceedings, Computer Graphics, 14(3), pp. 238-245.

[MSWI90] Microsoft Windows, User's Guide v. 3.0, 1990

[MUSK88] Mumford A. M., Skall M. W. (Eds.): CGM in the Real World, Springer-Verlag 1988

[NARO72] Nake F., Rosenfeld A., eds.: Graphic Language, North-Holland, 1972.

[NEWS72] Newell M.E., Newell R.G., Sancha T.L.: A New Approach to the Shaded Picture Problem, Proc. ACM Nat. Conf., 1972, pp. 443-453.

[NESP79] Newman W.M., Sproull R.F.: Principles of Interactive Computer Gra phics, 2nd ed., McGraw-Hill, New York, 1979. (*)

[NFKP91] Neumann L., Feda M., Kopp M., Purgathofer W.: Radiosity for very Complex Scenes, JŠPG Bratislava 1994.

[PAVL79] Pavlidis T.: Filling Algorithms for Raster Graphics, Computer Graphics and Image Processing, 10(2), Jun 1979, pp. 126-141.

[PAVL81] Pavlidis T.: Contour Filling in Raster Graphics, SIGGRAPH'81 Proceedings, Computer Graphics, 15(3), Aug. 1981, pp. 29-36.

[PAVL82] Pavlidis T.: Algorithms for Graphics and Image Processing. Rockville, Md.: Computer Science Press, 1982. (*)

[PAVL95] Pavlík, R.: Svět pavoučí sítí, Silicon World II., 1995

[PEIT92] Peitgen H., Jurgens H. Saupe D.: Chaos and Fractals, Springer Verlag 1988.

[PERL89] Perlin, K.: Hypertexture, Computer Graphics, 23(3), pp. 253-262, 1989.

[PEX390] PEX Introduction and Overview, PEX Version 3.20, pp. 1-14, 1990.

[PHON75] Phong B.T.: pozri [BUIT75]

[PLAS86] Plastock R.A., Kalley G.: Theory and Problems of CG, McGraw-Hill, 1986.

[POLA92] Poláček J., Ježek F., Kopincová E.: Počítačová grafika, ČVUT, Praha 1992. (S)

[PREP85] Preparata F.P., Shamos M.I.: Computational Geometry: An Introduction, Springer-Verlag Berlin, 1985. (*) (U)

[PUEY94] Pueyo, X.: The Use of Coherence in Radiosity, JŠPG Bratislava 1994.

[PURG91] Purgathofer W.: Grafische Datenverarbeitung, TU Wien 1991. (S)

[RUBI80] Rubin M.S., Whitted T.: A 3D Representation for Fast Rendering of Complex Scenes, SIGGRAPH'80, pp. 110-116, 1980.

[ROGA76] Rogers, D. F., Adams, J. A.: Mathematical Elements for Computer Graphics, McGraw-Hill, New York, 1976. (*)

[ROGE85] Rogers, D.F.: Procedural Elements for CG, New York, McGraw-Hill, 1985. (*)

[ROGS85] Rogers, D.F., Rogers S.D.: A Raster Display Graphics Package for Education in Computer Graphics '85, Tokio 1985.

[ROST89] Rost R., Friedberg J., Nishimoto P.: PEX: A Network-Transparent 3D Graphics System, IEEE CG and Applications 9 (4), pp. 14-26, 1989

[RUBI80] Rubin, S.M., Whitted, T.: A Three-dimensional Representation for Fast Rendering of Complex Scenes, Computer Graphics, 14(3), pp. 110-116, 1980.

[RUFE89] Ružický, E., Ferko A.: The demonstration programs for teaching of computer graphics, In : Počítačová grafika '89, Smolenice, pp. 110-112.

[RUHO89] Ružický, E., Horáček J.: Princípy skeletovania pre rozpoznávanie znakov, JŠPG Bratislava, 1989.

[RUZI91a] Ružický, E.: Ortopolygons and Windows System, pp. 101-106, JŠPG, Bratislava 1991

[RUZI91b] Ružický, E.: Úvod do počítačovej grafiky, MFF UK, Bratislava 1991. (S)

[RYAN86] Ryan D.: Modern Graphics Communications a CAD Approach, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[SAMA85] Samarskij A.: Současná aplikovaná matematika a počítačové modelování, Pokroky matematiky, fyziky a astronomie 1/85, Praha 1985.

[SERR82] Serra J.: Image Analysis and Mathematical Morphology, Academic Press, London, 1982.

[SCGE86] Scheifler R., Gettys J.: The X Window System, ACM Trans. on Graphics 5 (2), pp. 79-109, 1986.

[SKAL92] Skala V.: Algoritmy počítačové grafiky I, II, III, VŠSE Plzeň 1992. (S)

[SKAL93] Skala V.: Světlo, barvy a barevné systémy v počítačové grafice, Academia, Praha, 1993. (U)

[SLAV87] Slavkovský P.: Problém viditeľnosti v počítačovej grafike, MFF UK Bratislava 1987.

[SMDL90] Standard Music Description Language (SMDL), X3V1.8M/SD-8 ANSI Draft Standard, pp. 53-56, in: [HESE90].

[SOCH93] Sochor J., Žára J.: Algoritmy počítačové grafiky, ČVUT, Praha 1993. (S)

[SPRO79] Sproull R.F.: Raster Graphics for Interactive Programming Environmets, SIGGRAPH'79 Proceedings, Computer Graphics, 13(2), pp. 83-93, Aug 1979.

[SPER88] Šperka M.: Príspevok k návrhu akcelerátorov pre 3D počítačovú grafiku, ÚTK SAV, Bratislava 1988.

[SOKO94] Sokolowsky P., Šedivá Z.: Multimédia, současnost budoucnosti, Grada, Praha, 1994.

[STUC91] Stucki P.: Graphics and Multimedia, Tutorial No. 10, Eurographics'91, Vienna 1991.

[SUNG91] Sung H. C. K., Rogers G., Kubitz W.: A Critical Evaluation of PEX, IEEE CG and Applications, Vol 10, No 6, pp. 65-75, Nov 1991

[SUHO74] Sutherland I.E., Hodgman G.W.: Reentrant Polygon Clipping, Communications of the ACM, 17(1), Jan. 1974, pp. 32-42.

[SUSS74] Sutherland I.E., Sproull R.F., Schumacker R.A.: A Characterization of Ten Hidden-Surface Algorithms, Computing Surveys, 6(1), March 1974, pp. 1-55.

[THOM90] THOMAS, S.W.: X and PEX Programming, Tutorial, Eurographics'90, Zurich 1990.

[WATT89] Watt A.: Fundamentals of Three-dimensional Comp. Graphics, Addison-Wesley, 1989.

[WATT92] Watt A., Watt M.: Advanced Animation and Rendering Techniques: Theory and Practice, ACM Press, New York, 1992.

[WALL81] Wallace B.: Merging and Transformation of Raster Images for Cartoon Animation, SIGGRAPH'81 Proceedings, Computer and Graphics, 15(3), 1981, pp. 253-262.

[WARN69] Warnock J.A.: A Hidden-Surface Algorithm for Computer Generated Half-Tone Picture, Univ. Utah Computer Sci. Dept., 1969.

[WEGH84] Weghhorst H., Hooper G., Greenberg D.P.: Improved Computational Methods for Ray Tracing, ATM TOG 3(1), pp. 52-69, 1984.

[WHIT80] Whitted T.: An Improved Illumination Model for Shaded Display, Comm. ACM 26, N. 6, pp. 342-349, 1980.

[WRIG73] Wright T.J.: A Two Space Solution to the Hidden Line Problem for Plotting Functions of Two Variables, IEEE Trans. on Computers, TC22(1), 1973, pp. 28-33.

[ZHAN84] Zhang, T.Y., Suen, C.Y.: A fast parallel algorithm for thinning digital patterns, CACM 27, 3, 1984.

[ZATK93] Zaťko J.: On Multisided Bezier Patches, JŠPG Budmerice 1993.

[ZARA92] Žára J. a kol.: Počítačová grafika, principy a algoritmy, Grada, Praha 1992.

JŠPG je skratka In: Jarná škola počítačovej grafiky, zborník prednášok

(*) - existuje ruský preklad, (U) - učebnica, (S) - skriptum

Register

A

aditívne skladanie, 182
akcelerátor, 244
aktuálna správa o udalosti, 265
algoritmus orezávania Cohen-Sutherland, 26
algoritmus Cyrus-Beck, 30
algoritmus delenia okna, 169
algoritmus pre sledovanie lúča, 210
algoritmus prieniku mnohouholníka s polrovinou, 34
algoritmus priority, 168
algoritmus Scan_Line, 71
algoritmus určenia súvislosti hranice, 67
algoritmus vlnového vypĺňania, 65
algoritmus vypĺňania do hraničných bodov, 66
algoritmus vypĺňania podľa parity, 68
algoritmus priority, 168
alias, 62
ambientná zložka, 195
analytické plochy, 151
analytický popis plôch, 47
analytický spôsob zadania kriviek, 39
analýza algoritmu, 243
analýza modelu problému, 243
analýza problému, 243
analýza výsledkov, 243
antialiasing, 62
antiťažisko, 47
aplikačné programové rozhranie API, 242, 309
Application Programming Interface
aplikačný program, 8
aplikačný programátor, 241
aproximačné krivky, 43
aproximačné plochy, 51
aproximačné zadanie, 39
ARJ, 273
ASAP, 268, 300
As Soon As Possible
ASCII kód, 273
ASF indexu farby hrany, 294
ASF koeficientu hrúbky hrany, 294
ASF príznaku hrany, 294
ASF typu hrany, 294
aspekty vzhľadu, 256
ASTI, 268, 300
At Some Time
atribútové prvky, 275
atribúty, 256
atribúty hrán, 294
atribúty vnútra, 294
atribúty výstupných grafických prvkov, 281
automatická regenerácia, 268

B

B-splajnová plocha, 53
balíky nezávislé od zariadenia, 8
balíky podprogramov, 8
Bézierova plocha, 52
Bézierove krivky, 44
bikubická Coonsova záplata, 50
bilineárna Coonsova záplata, 49
bilineárna interpolácia, 230
binárne kódovanie 274, 276
bitmapa, bitová mapa, 273
blending funkcie, 49
Blinnov osvetľovací model, 202
BMP, 273, 277
Microsoft Windows Bit Map
BNIG, 268, 300
Before the Next Interaction Globally
BNIL, 268, 300
Before the Next Interaction Locally
bod objektu, 107
bod pozadia, 107
body siete, 51
Bouknightov model, 195
Bresenhamov algoritmus, 58, 60
Brownovho pohyb, 235
B-splajnové krivky, 46

C

CAD, 10, 278
CAM, 10, 278
CBS 299
Can Be Simulated
CDE, 252
Common Desktop Environment
CDR, 273
centralizovaná pamäť štruktúr, 280
centralized structure store, CSS
CGI, 253, 255
Computer Graphics Interface
CGM, 253, 271, 273, 274, 277
Computer Graphics Metafile
CGRM 272
Computer Graphics Reference Model
CIE, 184
CMY model, 184
Coded Representation of Multimedia and Hypermedia Information Objects, 273
Cookov model, 202
COSE, 252
Common Operating System Environment
CSG strom, 152
cyklická zámena, 167

Č

časová pečiatka 249
timestamp
čiarové prvky, 36

D

dátová rukavica, 10
dátový oblek, 10
deaktivovať, 266
degradačná funkcia, 88
detekovateľnosť, 263
detekovateľnosť segmentu, 265
DIB, 273
difúzna zložka, 195, 199
digitálny obraz, 4
dilatácia, 126
displej, 251
distribučná funkcia normál mikroplôšok, 201
dominantná frekvencia, 186
doplnkové farby, 187
druh a presnosť textu, 293
druh písma, 293
DRW, 273
dvojrozmerné pole buniek, 294
DWG, 277
DraWinG
DXF, 273, 278

E

echo, 244, 264
editovací prvok štruktúry, 295
efektívne spracovanie grafickej informácie, 241
elektromagnetické vlnenie, 179
EPS, 273, 274, 277
Encapsulated PostScript
erózia, 126
Euklidovská metrika, 104
Eulerova formula, 156
externé prvky, 275

F

farba, 256, 257, 292
farebné (fotopické) zrakové vnímanie, 2, 182
farebný rozsah, 188
Fergusonova kubika, 40
Fergusonova záplata, 51
FIF, 273
filter Butterworth, 101
filtrácia, 95
filtrácia metódou mediánu, 99
filtrácia obyčajným priemerovaním, 98
formát dátového súboru, 305
form-faktor, 221
Foundation Component 307
fraktál, 235
fraktálne povrchy, 236
funkcia hustoty rozdelenia, 87
funkčná norma, 254
funkčná špecifikácia, 274
funkčnosť systému, 241

G

Gaussovské rozdelenie, 98, 110
generácia noriem, 307
geometrické atribúty, 290
geometrický popis, 243
geometrický útlmový faktor, 198
GIF, 9, 273, 277
Graphics Interchange Format
GKS, 253, 279
Graphical Kernel System
GKS metasúbor, 271
GKS OPEN, 269
GKS-3D, 253
GKS-94, 272
globálne atribúty textu, 257
globálne prahovanie, 109
glyf, 274
Gouraudovo tieňovanie, 204
gradient, 103
gradientné operácie, 95
grafické systémy, 8
grafické užívateľské rozhranie GUI, 241, 242
Graphical User Interface
grafické výstupné prvky, 256, 275
grafický systém, 8, 242
grafický výstup, 256, 281
grafika, 251

H

hardver, 305
hašovacia tabuľka, 72
Hausdorffova dimenzia, 234
hierarchické okná, 248
high-pass filtering, 105
histogram, 87
hlasový výstup, 303
homogenita oblastí, 122
homogénne súradnice, 11
HPGL, 273
hrana, 102
hraničné vypĺňajúce algoritmy, 65
hrúbka čiary, 63
hudobná informácia, 274
Huygensov princíp, 179
hyperdokument, 274
Hypermedia a Multimedia, 309
HyperODA, 274, 278
Open Document Architecture and Interchange Format

HyTime, 274, 278, 305
Hypermedia/Time-based Structuring Language

I

ideálny filter, 100
identifikátor hlhsr 291
identifikátor výberu 291
IEF, 277
Image Exchange Format
IFF, 273
IGES, 273, 277
Initial Graphics Exchange Specification
ikony, 245
Image Processing and Interchange IPI, 253, 254
IMG, 273
IMM, 299
IMMediately
implementátor grafického systému, 241
implicitná regenerácia obrazu, 290
impulzný šum, 100
index do tabuľky farieb, 292
index hrany 294
index lomenej čiary, 256
index pohľadu 291
index textu, 259, 293
indikátor modelujúceho orezávania 284
infračervené žiarenie, 179
inicializácia vstupu, 269
inportabilita, 305
INPUT, 246
intenzita vrcholu, 204
interaktívna počítačová grafika, 3
interpolačné krivky, 40
interpolačné plochy, 48
interpolačné zadanie, 39
IRG, 299
Implicit ReGeneration
IS 2022, 273
IS 646, 273
ISO/IEC JTC1/SC24 , 254
ISO/IEC JTC1/SC29, 18/254

J

jadro, 97
jas, 4, 181, 189
jasová korekcia, 87
jazyk popisu stránok 277
page description language
jazykové napojenie, 254
JBIG, 273, 274, 276
Progressive Bi/level image compression standard
JPEG, 273, 274, 277
Joint Photographic Experts Group

K

kabinetné premietanie, 143
kategórie pracovných staníc, 266
klient-server, 251
kódovanie, 273
kódovanie textom, 276
kódovanie znakmi, 276
koeficienty masky, 96
koherentná zložka, 199
koherentnosť, 70
kompresia, 273
konštantné tieňovanie, 204
konštrukčné prostredie, 307
konvexný mnohouholník, 36
konvolučná maska, 98
konzistencia, 245
korelačná vzdialenosť, 198
kosouhlý priemet, 143
kostra, 131
kubické splajnové krivky, 42
kumulatívny histogram, 91
kurzor, 244

L

Laplaceov gradientný operátor, 105
lexikálny návrh, 243
logické prostredie, 307
logické rozhranie, 280
logické vstupné zariadenia, 256
logická operácia, 152
lokálne prahovanie, 111
lokátor, 263
lom svetla, 179
lomená čiara, 256, 292
L-systémy, 237

M

magnetický vektor, 179
manhattanská metrika, 104
manipulovanie so štruktúrami, 297
mapovacia matica záplaty, 49
maska okolia, 96
matica otočenia v 2D, 285
matica posunutia, 20, 285
matica orientácie pohľadu 286
matica projekcie 286
matica otočenie okolo osi, 285
matica škálovania, 285
medián, 99
medzinárodné normy, 253
menu, 243
metasúbor, 256, 273
metóda vyrovnávania histogramu, 89
metodika matematického modelovania, 243
metodika návrhu dialógu, 243

metodika zobrazovania, 243
mezopické vnímanie, 2
MHEG, 274, 278
Multimedia/Hypermedia Experts Group
MIDI, 274, 305
Musical Instruments Digital Interface
MIME, 274
Multipurpose Internet Mail Extensions
model CMY, 188
model HSV, 189
model klient-server, 251
model RGB, 187
model XYZ, 185
Modeling and Presentation Component, 307
modelujúce orezávanie, 284
modelujúci súradnicový systém MC, 282
modelling coordinate system
modelujúci orezávací priestor 284
modely empirické, 194
modely prechodové, 194
Modern Hardware Facilities, 309
Modern User Interface Methodologies, 309
modifikácia obrazu pomocou histogramu, 87
modulácia normály plochy, 232
modulovaná veličina, 231
monochromatické obrazy, 274
monochromatické žiarenie, 179
monochromatický obraz 274
morfológia, 125
MPEG, 273, 274, 277
Moving Picture Experts Group
MPS, 273
MS Windows, 250
multimédiá, 303
Multimedia Presentation and Interchange, 254
Multimedia System Services Component, 307
multimediálna informácia, 274
multimediálna scéna, 308
multimediálna technológia, 8
multimediálny objekt, 307
multimediálny systém, 274

N

nanášací kanál 301
rendering pipeline
negeometrické atribúty, 290
nekoherentná zložka, 199
neštrukturovaný obraz 273
neuniformná B-splajnová krivka, 301
neuniformná B-splajnová plocha, 301
neuniformná B-splajnová plocha s údajmi, 301
neviditeľnosť, 291
New API, 309
NFF, 273, 277
Neutral File Format
NIVE, 300
No Immediate Visual Effects
normalizované grafické systémy 246, 253
normalizované súradnice NDC, 261
Normalized Device Coordinates
normalizované zariadenie, 261
normalizované projekčné súradnice NPC, 286
normalized projection coordinates
normalizujúca transformácia, 261, 286

O

obálka znaku, 37
Object Oriented Programming, 309
objektový priestor, 161
objekt scény, 308
objemové modelovanie, 153
oblasť, 64
oblasť hranične definovaná, 65
oblasť vnútorne definovaná, 65
oblasťi záujmu 17, 303
obraz, 4
obrázok, 4
obrazovka, 8, 251
screen
obrazový priestor, 161
obsluha okna 246
ochrana informácie, 245
oddeľovače, 275
odlesky, 196
odosielajúca funkcie 296
posting function
odraz svetla, 179
odstránenie nekonvexných uhlov, 36
OFF, 273, 277
Object File Format
ohnisková vzdialenosť, 2
okno, 17, 251, 261
okno stanice 289
oknový systém, 242
window-management systems, WMS
oknový systém X, 250
X window system
OMG, 307
Object Management Group
OPEN LOOK, 245
OpenGL, 9
operačné stavy, 269
operátor, 241
optimálne prahovanie, 110
orezávací priestor, 284
orezávanie, 25, 241
orezávanie na záber, 262
orezávanie textu, 293
orezávanie úsečky, 25
orientovaná priamka, 20
OSF/Motif, 245, 251
OSI 309
Open Systems Interconnection
osový kríž, 244
ostrenie obrazu, 95
otáčanie, 12, 20

otvorenie, 126
otvorený segment, 262
OUTIN, 266
OUTPUT, 266
ožiarenie, 197

P

pamäťové obrazovky, 8
parametrické orezávanie, 28
pasívna (zobrazovacia) grafika, 3
PCX, 9, 273, 277
CompuServe Graphics Metafile
pevný bod, 262
pevný bod pre škálovanie a rotáciu, 285
PEX, 301
PHIGS, 253, 279
Programmer's Hierarchical Interactive Graphics System
PHIGS PLUS, 253, 300
Phongov model, 196
Phongovo tieňovanie, 205
PIC, 273
pixel, 4
pixmapa, 273
plošné prvky, 36
počítačová grafika, 3
pohyblivé obrázky, 274
moving pictures, film, animácia
pole buniek, 259, 294
pole buniek PLUS, 301
poloha pozorovateľa, 286
popisovače metasúboru, 275
popisovače obrázku, 275
porovnávacie krivky, 183
PostScript, 9, 252
posuvné šablónovanie, 151
poškodenie okna, 249
potomok, 281
potvrdenie, 244
pracovná stanica, 254, 256, 266, 280, 298
workstation
pracovný stôl, 245
desktop, 225
prahovanie, 107, 108
prahovanie dynamické, 108
prahovanie globálne, 108
prahovanie lokálne, 108
pravidlá štýlu, 242
pravotočivá sústava súradníc, 18
prechádzanie štruktúrou 281, 296
PREMO 305, 307
Presentation Environment for Multimedial Objects
prenos obrazu, 5
prenositeľnosť, 9, 274
Portability
prepínač súradníc, 262
presnosť textu, 293
prezentácia, 303, 307
priama manipulácia, 245
priebežná hodnota, 264
priemerná odchýlka, 198
priemerné hodnoty, 110
priemerný sklon, 198
priemetňa, 141
prienik, 25, 284
prienik dvoch mnohouholníkov, 36
prienik polroviny s mnohouholníkom, 32
priepustná množina pre xxx 291
xxx inclusion set
priestorová textúra, 231
príkazový jazyk, 245
command language
priorita segmentu, 263
prírastkový algoritmus, 56, 57
prirodzený kubický splajn, 42
prirodzený obraz, 4, 303
prirodzený zvuk, 303
príznak hrany 294
príznak pôvodu, 261
príznak pôvodu aspektov ASF, 291
aspect source flag
problém exponovania 302
prostredie, 256
prostriedky na riadenie interakcie, 246
protokol X, 251
pružná čiara, 244
rubber band
prvok štruktúry 280,295
structure element

R

radiačná metóda, 193, 220
Radiosity
rasterizácia, 55
rastrová grafika, 8
rastrový rozklad kružnice, 60
rastrový rozklad úsečky, 55
ray-tracing, 193
realizačné prostredie, 307
referenčná rovina, 287
referenčný model grafických noriem, 254
referenčné súradnice pohľadu - VRC 286
view reference coordinates
relatívny popisný text, 290, 292
annotation text relative
reprezentácia prvku na stanici, 256
reprezentácia textu, 293
reťazec, 263
string
režim udalosti, 265
režim vzorkovania, 265
režim vyžiadania, 265, 264
režim zdržania, 299
režim hlhsr 291
RGB model, 184

riadenie farieb, 302
riadiace prvky, 275
riadiaci prvok štruktúry, 295
RIB, 273, 278
RenderMan Interface Bytestream
Robertsov operátor, 104
rodič, 281
rotačná plocha, 149
rotačné teleso, 149
rovnobežné premietanie, 142
rozklad na trojuholníky, 36
rozlišovacia schopnosť, 5
rozostup znakov, 293
rozpoznávanie obrazcov, 4
rozpoznávanie zvuku, 303
rozširujúci prvok, 275
rozširujúcou funkcia, 272

S

scéna, 277, 308
segment, 256, 262
segmentácia, 107
segmentové prvky, 275
sémantický návrh, 243
Serrova transformácia, 129
SGML IS 8879, 275
Standard Generalized Markup Language
Shading, 203
sieť, 51
sieť štvoruholníkov s údajmi, 301
siete štruktúr, 280, 295
structure networks
sila reakcie gradientu, 120
skanovacie algoritmy, 117
skelet, 125, 131
skladanie dvoch transformácií, 285
skopické videnie, 182
skotopické vnímanie, 2
skupina lomených čiar s údajmi, 301
skupina mien, 291
skupina výplňových oblastí s údajmi, 301
skupina trojuholníkov s údajmi, 301
skupina výplňových oblastí, 290
sled polôh, 263
sled značiek, 256, 292
sledovanie lúča, 207
SMDL, 305
Standard Music Description Language
smer gradientu, 120
smer textu, 257, 258, 297
smerník na prvok, 297
Snellov zákon lomu, 180, 197
snímací priestor, 289
view volume
snímacie orezávanie, 286
view clipping
snímacie prostredie, 307
Sobelov operátor, 105
softver stanice, 255
spätná väzba, 244
feedback
spektrálna farba, 180
spektrálna krivka, 180
splajnové krivky, 41
splajnové plochy, 151
spojitý obraz, 4
spracovanie chýb, 256, 269, 270
spracovanie obrazu, 4
správa okien, 251
správca okien, 246
window manager
spúšť, 244, 264
trigger
statické obrázky, 274
still pictures
stav vizuálnej reprezentácie, 299
stav aktualizácie zobrazovania, 296
stavový zoznam stanice, 270, 299
stavový zoznam GKS, 270
stavový zoznam prechodu, 296
stenčovanie, 126, 131
stereoskopia, 160
stochastické samopodobné fraktály, 237
súbor chýb, 270
subtraktívne skladanie, 182
súmernosť podľa roviny, 23
súradnice zariadenia - DC, 289
device coordinates
súradnicové systémy, 256
susednosť, 64
sústava súradníc, 18
svetelné pero, 8
svetelný lúč, 179
svetlosť, 181
svetové súradnice - WC, 261, 282
World Coordinates, WC
synchronizácia pohľadu, 160
syntaktický návrh, 243
syntetický a prirodzený obraz, 303
syntetický a prirodzený zvuk, 303
syntetický obraz, 4, 303
systém správy užívateľského rozhrania, 246
UIMS - user interface management system
sýtosť, 181, 189

Š

šikmé premietanie, 142
škálovacie faktory, 262
škálovanie, 13
šošovka, 2
šrafovanie mnohouholníka, 74
štandardné odchýlky, 110
štandardný zdroj, 185
štruktúra, 280
štrukturálny prvok, 125
šumová funkcia, 232

T

tabuľka kódu, 276
tabuľka pohľadov, 286
view table
tabuľka popisu GKS, 270
tabuľka popisu stanice, 270, 299
tabuľka zväzkov, 260, 291
tabuľka farieb, 291
tabuľka reprezentácií pre vzorky, 291
tabuľky popisu a stavové zoznamy GKS, 270
ťah písma, 259
text, 256, 257, 292
textúra objektu, 229
TGA, 273
tieňovacia technika, 194
tieňovanie, 203
TIFF, 273, 277
Tag Image File Format
Timer, 249
tón farby, 189
transformácia na stanicu, 267, 289
transformácia segmentu, 262
transformačná funkcia procesu, 100
translačné teleso, 151
trichromatické spektrálne súradnice, 187
turbulencia, 233
typ anotácie, 293
typ čiary, 256
typ hrany, 294
typ vnútra, 257

U

udalosti, 264
ultrafialové žiarenie, 179
UNICODE, 274
uniformná B-splajnová krivka s farbou, 301
uniformná parametrizácia, 40
UQUM, 300
Use Quick Update Method
úrovne GKS, 271
UWOR, 300
Update WithOut Regeneration
uzatvorenie, 126
uzavretie krivky, 43
užívateľský model, 243

V

valuátor, 263
varieta, 156
väzba na štruktúru, 295
vektor posunu, 262
vektorové obrazovky, 8
vertikálny vektor znaku, 257
viditeľnosť, 263
viditeľný bod, 161
virtuálna realita, 245
virtuálne prostredie, 307
vizuálne vnímanie, 2
vojenská perspektíva, 143
voľba, 243, 263
choice
vstup, 271, 281
vstup polohy, 244
vstup reťazca, 244
vstup udalosťou, 264
EVENT
vstup vyžiadaním, 264
REQUEST
vstup vzorkovaním, 264
SAMPLE
vstup z metasúboru, 272
vstupná fronta, 265
vstupná hodnota, 244
vstupná priorita, 287
vstupné režimy, 256
vstupné zariadenia, 246
vstupný jazyk, 243
všeobecná špecifikácia farby, 301
všeobecná textúra, 229
výber, 263
pick
vyhľadávanie obrysu, 122
vyhladenie obrazu, 95
vyhladzovanie, 62
vyjadrenie geometrického popisu, 243
vylučovacia množina pre xxx, 291
xxx exclusion set
vypĺňanie podľa parity, 68
výplňová oblasť, 256, 294
vysokofrekvenčné filtrovanie, 105
výstup do metasúboru, 271
výstupné grafické prvky, 281
výstupný jazyk, 243
výška znaku, 257, 258, 293
výzva, 244, 264
prompt
vzhľad a funkčnosť resp. správanie, 242
look and feel
vzorkovane, 264

W

WAIT, 300
When the Application requests IT
window manager, 246
window system, 246
Windowing, 309
Windows Management System, 253
WMF, 273
WPG, 273
WWW, 274
World-Wide Web
WYPIWYP, 305
what you play is what you print

WYSIWYG, 245
what you se is what you get

X

X Toolkit, 251
X Window System, 250
Xlib, 251, 279
Xt Intrinsics, 251
XToolkit, 251

Z

záber, 17, 261
záber stanice, 289
záber projekcie, 289
zadanie čísla, 244
základné teleso, 152
základné interaktívne úlohy, 244
základné komponenty, 303
zákon lomu, 180
zákon o vzájomnej nezávislosti lúčov, 180
zákon odrazu, 180, 197
zákon priamočiareho šírenia svetla, 180
zákon zachovania energie, 199
záplata, 48
zariadenie, 8
zarovnanie textu, 258, 293
zdržanie zmien obrazu, 268
ZIP, 273
zisťovacie funkcie, 270
zložená modelujúca transformácia, 282
zmena mierky, 13, 19
zmena sústavy súradníc, 20
zobrazenie, 243
zobrazovací kanál, 281
zobrazovací model, 242
zobrazovacie pole, 17
zobrazovací zoznam, 248
zosilňovanie, 126
zovšeobecnený grafický výstupný prvok, 256, 260
zovšeobecnený prvok štruktúry, 295
zovšeobecnený výstupný prvok, 294
zrkadlová zložka, 195
zväzky, 256
zväzky atribútov na stanici, 267
zvuk prirodzený, 305
zvuk syntetický, 305
zvýraznenie, 263, 291

Ž

žiarivý tok, 180, 197

1..9

2D vstup i výstup, 8
3D pravotočivý karteziánsky súradnicový systém, 282
4-súvislé, 65
8-súvislé, 65

PANEURÓPSKA VYSOKÁ ŠKOLA

Fakulta informatiky

Príhovor dekana fakulty

Sme prvá a jediná súkromná fakulta informatiky na Slovensku. Naša malá fakulta Paneurópskej vysokej školy (**PEVŠ**) sa líši od iných etablovaných vysokých škôl tým, že poskytujeme individuálne prístupy, ktoré iné fakulty neposkytujú.

Preto našim cieľom je pripraviť absolventov tak, aby boli odborníci nie len v úzkom špecializovanom odbore, ale aj so základnými vedomosťami a zručnosťami z manažmentu, ekonómie, práva, komunikácie, etiky a psychológie.

Doc. Ing. Martin ŠPERKA, CSc.

Prečo študovať na Fakulte informatiky PEVŠ

1. **Otvorenosť.** Sústreďujeme sa predovšetkým na tie aplikácie informatiky, v ktorých má Paneurópska vysoká škola najlepšie kompetencie. Okrem všeobecnej aplikácie informatiky, sú to oblasti spolupráce s ostatnými fakultami práva, ekonómie a podnikania, médií a tiež psychológie.

2. **T profil absolventa**. Mnohé IT firmy vyžadujú absolventov, ktorí nie sú len odborníci v úzkom špecializovanom odbore, ale majú aj základné vedomosti z manažmentu, ekonómie, práva, komunikácie, etiky a psychológie (t.j. T profil absolventa). Paneurópska vysoká škola má k tomu ideálne predpoklady.

3. **Flexibilnosť štúdia informatiky**. Možnosť navštevovať predmety iných fakúlt PEVŠ a tak flexibilne upraviť svoje štúdium napĺňa myšlienku univerzitného vzdelania, kde absolventi nie sú len dobrí a úzko zameraní špecialisti ale aj integrované osobnosti.

4. **Kvalita pedagogických pracovníkov.** Naším krédom je zabezpečiť pre jednotlivé predmety najlepších odborníkov a pedagógov – akademikov ale aj odborníkov z praxe. Angažujeme externých profesorov z iných vysokých škôl na Slovensku alebo zo zahraničia. V oblasti podnikových informačných systémov prednášajú na Fakulte informatiky profesori z pražskej VŠE a Karlovej univerzity, ktorí patria k najlepším odborníkom nielen v Čechách ale aj v Strednej Európe.

5. **Individuálny prístup.** Sme súkromná vysoká škola. Študenti si musia za štúdium platiť, to však neznamená, že automaticky absolvujú školu bez námahy, ale na druhej strane snažíme sa im poskytnúť čo najlepšie služby vrátane individuálneho prístupu v ich štúdiu.

6. **Spojenie s praxou** je podmienka aby naši absolventi boli úspešní na trhu práce. Napríklad mnohí poslucháči majú školiteľov záverečných prác z etablovaných firiem. Medzi voliteľnými predmetmi je aj odborná prax v IT firmách.

7. **Poskytujeme externé štúdium.** Mnohí naši poslucháči sú zamestnancami počítačových firiem, kde sú naplno vyťažení, takže nemôžu študovať popri zamestnaní počas pracovných dní. Z tohto dôvodu prebieha štúdium v sobotu a formou individuálnych projektov a konzultácií, pričom majú k dispozícii množstvo kvalitných študijných materiálov.

8. **Moderné technické vybavenie.** Multimediálne centrum Fakulty masmédií PEVŠ je najlepšie vybavené štúdio na slovenských vysokých školách. Prebieha v ňom aj výučba niektorých predmetov našej fakulty. K tomuto prispieva aj moderné prostredie, učebne, sociálne zariadenia a mimoškolské aktivity vrátane klubu študentov.

9. **Medzinárodná spolupráca.** Fakulta informatiky PEVŠ spolupracuje s inými zahraničnými univerzitami a študenti môžu absolvovať štúdium aj v zahraničí.

Kontakt: www.paneurouni.com

Fakulta informatiky PEVŠ
Nevädzová 5
821 01 Bratislava 2

Tel.: +421 2 4828 7259, +421 2 4828 7320
Email: studijne.fi@paneurouni.com