FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITA KOMENSKÉHO
V BRATISLAVE

DIPLOMOVÁ PRÁCA

2003 Pavol Novotný

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITY KOMENSKÉHO V BRATISLAVE
KATEDRA POČÍTAČOVEJ GRAFIKY A SPRACOVANIA OBRAZU

DIPLOMOVÁ PRÁCA

Reprezentácia geometrických objektov pomocou poľa hustoty a gradientu

DIPLOMANT: PAVOL NOVOTNÝ
ŠKOLITEĽ: ING. MILOŠ ŠRÁMEK

BRATISLAVA 2003

Čestné prehlásenie:
Čestne prehlasujem, že predkladanú prácu som vypracoval samostatne pod odborným vedením školiteľa len s použitím uvedenej literatúry.

. . . . . .
Pavol Novotný

Poďakovanie

Predovšetkým sa chcem poďakovať môjmu školiteľovi Milošovi Šrámkovi za množstvo času, ktoré mi venoval pri zodpovedaní odborných i praktických otázok spojených s diplomovou prácou. Tiež som mu vďačný za sprostredkovanie študijného pobytu v zahraničí, ktorý bol pre mňa veľkým prínosom. Ďalej ďakujem mojim spolubývajúcim za vytváranie príjemnej atmosféry počas celého štúdia a pomoc pri riešení hardvérových i softvérových problémov. V neposlednom rade patrí vďaka mojej rodine a priateľom za ich neustálu morálnu podporu.

Abstrakt

Touto diplomovou prácou predstavujeme nový spôsob reprezentácie objemových dát. Vychádzame z techniky limitovaných vzdialenostných polí, ku ktorým pridávame normalizovaný gradient. Aby sme znížili pamäťové nároky, používame kompresiu založenú na podobnom princípe ako známa metóda run-length encoding. Ďalej prezentujeme novú metódu na realizáciu CSG operácií medzi objemami na voxelovej úrovni. Vďaka zapamätanému gradientu dosahujeme lepšie výsledky ako doteraz používané operácie využívajúce jednoduché minmaxové kritérium. Implementáciou uvedných techník vznikla knižnica `vxtRL`.

Obsah

Zoznam obrázkov

Kapitola 1

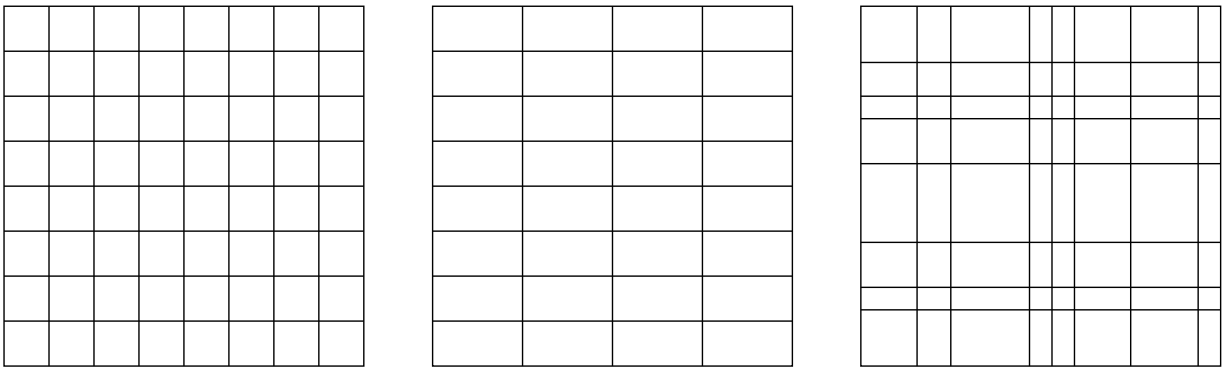
Úvod do problematiky

1.1 Objemová grafika a voxelizácia

Objemová grafika, pojem zavedený do života autormi Kaufman, Cohen a Yagel v roku 1993 [1], sa ako podoblasť počítačovej grafiky zaoberá štúdiom priestorových objektov v ich skutočnej trojrozmernej podstate. Tento prístup sa diametrálne odlišuje od tradičnej povrchovej grafiky. Zatiaľ čo tá je zameraná predovšetkým na vytváranie modelov, ktoré sa snažia napodobniť vzhľad reálneho sveta, objemová grafika má snahu skúmať vnútornú podstatu objektov. Už pred stáročiami výtvarní umelci pochopili (napríklad Michelangelo), že ak chcú verne zobrazovať realitu, nestačí sa na objekty pozerať len zvonku, ale treba dôkladne poznať aj ich vnútornú štruktúru, pretože tá do značnej miery ovplyvňuje vlastnosti povrchu. Povrchová grafika dosiahla v mnohých smeroch modelovania reality obdivuhodné výsledky, avšak stále je do istej miery viazaná na pomernú jednoduchosť zobrazovaného povrchu. Zrejme si ťažko dokáže poradiť s objektami, ktoré nemajú vôbec definovaný povrch, ako napríklad oheň, dym, či hmla. Naviac niektoré aplikácie z rôznych oblastí vedy a výskumu potrebujú zobrazovať dáta, ktoré sú vo svojej podstate trojrozmerné. Ako príklad možno uviesť medicínu (tomografia, magnetická rezonancia, ultrazvuk), biológiu (mikroskopické systémy), geografiu (seizmológia), časticovú fyziku (hustota elektrónov), priemysel (štruktúra materiálov, prúdenie kvapalín a plynov), meteorológiu (atmosférické javy), a mnohé ďalšie. Objemová grafika sa zrodila ako prirodzený nástroj na riešenie týchto nastolených poblémov. Má však potenciál súperiť s povrchovou grafikou aj v oblastiach, v ktorých zatiaľ tradičný prístup dominuje.

Najväčšími nevýhodami objemovej grafiky sú jej vysoké nároky na pamäť a výpočtový čas. Nakoľko však výkonnosť počítačov rýchlo narastá, tieto nedostatky postupne prestávajú hrať dôležitú rolu. Na druhej strane môžeme spomenúť výhody, medzi ktoré patria nezávislosť na zložitosti scény, jednotné spracovanie rôznych typov dát (namerané alebo vypočítané dáta, parametrické a implicitné povrchy), a jednoduchá realizácia *CSG operácií* [1].

Základom objemovej reprezentácie je *voxel*, elementárna jednotka objemu (skratka z anglického *volume element*). Každý voxel obsahuje určitú informáciu, ktorá lokálne charakterizuje daný objem. Typ tejto informácie závisí od konkrétnej implementácie. Existuje široká škála možností od *binárneho voxelu* (ten má len dvojakú hodnotu podľa toho, či leží vo vnútri alebo



(a) (b) (c)

Obrázok 1.1: *Typy mriežok*
(a) karteziánska, (b) regulárna, (c) rektilineárna

mimo objektu) až po voxel s mnohými atribútmi (hustota, normála, farba, materiál, indexy lomu a odrazu, . . .). Voxely môžu byť vo vnútri objemu rozložené rôznym spôsobom, obvykle sa však sa používa nejaký druh pravouhlej mriežky (pozri obrázok 1.1). V prípade karteziánskej mriežky môžeme definovať *voxelovú jednotku (VU — voxel unit)* ako vzdialenosť susedných voxelov.

Proces, počas ktorého sa voxelom priradí príslušná informácia, sa nazýva *voxelizácia*. Ako prvé boli prezentované techniky, ktoré vytvárali binárne dáta. Z hľadiska následného zobrazovania vytvorených dát však tento prístup nebol ideálny, pretože viedol k nežiadúcemu aliasingu. Problémom je najmä určovanie normály na povrch. Boli vytvorené viaceré metódy na jej výpočet zo širšieho okolia voxelu, ale žiadna nebola dostatočne presná na simuláciu takých javov, ako je odraz alebo lom na povrchu objektov. Lepšie výsledky sa dosiahli pomocou *diskrétneho raytracingu* [2], kde sa využíva znalosť analytického popisu objektu. Je však potrebné, aby si každý voxel pamätal dodatočnú informáciu, ktorému objektu patrí.

Iný spôsob, ako sa vysporiadať s aliasingom sú *filtračné techniky* [3][4]. Tie používajú na úpravu binárnych dát nízkofrekvenčný filter, napríklad *Barletov filter*, *gausián* alebo *oriented box*. Konvolúciou objektu s filtrom vznikne spojitá funkcia, ktorá sa navzorkuje do danej mriežky. Za predpokladu, že doména filtra má menšie rozmery ako voxelizovaný objekt, bude vnútro reprezentované istou „vnútornou“ hustotou a vonkajšia oblasť „vonkajšou“ hustotou. Na povrchu objektu vznikne tenká prechodová oblasť, v ktorej hustota prechádza spojite od vnútornej do vonkajšej. Takto vzniknuté dáta majú podobné vlastnosti ako tie, ktoré vzniknú skenovaním reálnych objektov (napríklad pomocou tomografu), vďaka čomu ich možno spolu ľahko kombinovať a vizualizovať ich jednotným spôsobom. Ponúkajú sa viaceré možnosti, ako pri zobrazovaní využiť prechodovú oblasť. Môžeme presne definovať povrch pomocou prahovania na strednej hodnote hustoty, alebo využiť tento „rozmazaný“ povrch priamo na antialiazing tým, že danému lúču priradíme intenzitu podľa hustoty voxelov, ktoré zasiahol [3].

Dôležitou sa ukazuje byť otázka, aké filtre použiť, aby sme dosiahli čo najlepší výsledok. Autori článku [4] zistili, že výber filtra je úzko spätý s nasledovnou metódou, ktorú použijeme na interpoláciu hustoty a výpočet normály. Pri nevhodnej kombinácii voxelizačných filtrov s rekonštrukčnými technikami dochádza k systematickej chybe, ktorá nezávisí od krivosti povrchu, ale

od jeho orientácie, čo je nežiaduce. Podrobnejšie sa o tom zmienime v časti *Chyby rekonštrukcie*.

1.2 Reprezentácia objektov pomocou vzdialenostných polí

Ďalší prístup, ktorým sa dá redukovať aliasing, je objemová reprezentácia využívajúca *vzdialenostné polia*. V tomto prípade sa do voxelu zapisuje informácia o vzdialenosti k najbližšiemu bodu na povrchu. Aby sme rozlíšili medzi vnútornou a vonkajšou oblasťou, majú voxely na rôznych stranách povrchu opačné znamienko. Prvýkrát boli vzdialenostné polia realizované naplnením celého objemu hodnotami vzdialenostnej funkcie určenej povrchom objektu [5], teda informáciu o objekte obsahovali všetky voxely v mriežke. To je v kontraste oproti tradičnému chápaniu objektu, ktorý je priestorovo lokalizovateľný. Vylepšené návrhy sa snažia vybrať si presnejšie množinu voxelov, ktoré sú pre popis povrchu potrebné, a ukladať informáciu len do nich. Ako príklady môžeme spomenúť *limitované vzdialenostné polia (truncated distance fields)* [4] a *adaptívne vzdialenostné polia (adaptively sampled distance fields — ADF)* [6].

Limitované vzdialenostné polia vychádzajú z myšlienky, že na kódovanie povrchu je potrebná len úzka vrstva voxelov v jeho okolí. Vo vnútri tejto vrstvy sa hustota mení lineárne vzhľadom na vzdialenosť od povrchu a vo vzdialenejšom okolí majú voxely konštantnú vnútornú alebo vonkajšiu hustotu. Formálne to podľa [4] môžeme popísať funkciou:

$$D(x,y,z) = \begin{cases} 2T & \text{pre } d(x,y,z) < -\delta \\ 0 & \text{pre } d(x,y,z) > \delta \\ T\left(1 - \frac{d(x,y,z)}{\delta}\right) & \text{inak} \end{cases}$$

kde $d(x,y,z)$ je vzdialenosť bodu od povrchu (vo vnútri objektu záporná), 2δ je šírka prechodovej oblasti a T je prahová hodnota definujúca povrch objektu. Dôležitá je správna voľba konštanty δ. Jej veľkosť určuje rozmery najmenšieho reprezentovateľného detailu. V prípade, že je táto hodnota príliš veľká, vedie to k vyhladeniu detailov. Na druhej strane veľmi malá hodnota nám spôsobí problémy pri rekonštrukcii, v limitnom prípade vlastne získame binárne dáta. Z výsledkov prezentovaných v [4] vyplýva, že ideálna hodnota δ je približne 1.8, čo je o niečo viac ako dĺžka telesovej uhlopriečky jednotkovej kocky. Vychádza to z faktu, že pri rekonštrukcii povrchu potrebujeme interpolovať hodnoty z najbližších 8 voxelov vo vrcholoch kocky, v ktorej sa nachádza bod povrchu, čiže je vhodné, aby všetky tieto vrcholy ležali v prechodovej oblasti. V prípade, že bod povrchu sa nachádza blízko jedného vrcholu kocky, protiľahlý vrchol je od neho vzdialený práve o spomínanú dĺžku telesovej uhlopriečky ($\sqrt{3}$).

Adaptívne vzdialenostné polia efektívne reprezentujú najmä scény, v ktorých sú kombinované veľké hladké povrchy s drobnými detailami. Pri homogénnom vzorkovaní priestoru by sme potrebovali extrémne veľké rozlíšenie i kvôli niekoľkým drobným detailom, ktoré zaberajú len zlomok priestoru. ADF rieši tento problém hierarchickým delením priestoru. Na rozdiel od iných systémov, kde rekurzívne delenie prebieha na základe kategorizácie priestorových blokov na vnútorné, vonkajšie a prechodové, ADF riadi budovanie oktantového stromu využívajúc kritérium lokálnej krivosti povrchu. Výsledkom je, že v okolí detailov prebieha vzorkovanie hustejšie ako v homogénnejších oblastiach, vďaka čomu je pamäť rozumne využitá.

Vzdialenostné polia boli úspešne využité v rôznych druhoch aplikácií, ako napríklad v robotike (plánovanie pohybu, šablónovanie (*swept volumes*)), pri objemovom renderovaní, definovaní odsadených povrchov (*offset surfaces*), na morfovanie medzi povrchovými modelmi a v objemových modelovacích systémoch (*sculpting systems*) [7].

1.3 Určovanie povrchovej normály

Pri kvalitnom zobrazovaní volumetrických dát musíme z diskrétnej informácie v mriežke rekonštruovať okrem povrchu aj normálu na povrch, ktorú potrebujeme na simuláciu rôznych svetelných efektov (tieňovanie, odraz, lom, ...). Najbežnejší spôsob na určovanie povrchovej normály je výpočet gradientu hustoty pomocou *stredových diferencií* a následná normalizácia tohto vektoru:

$$\begin{aligned} g^x_{i,j,k} &= d_{i+1,j,k} - d_{i-1,j,k} \\ g^y_{i,j,k} &= d_{i,j+1,k} - d_{i,j-1,k} \\ g^z_{i,j,k} &= d_{i,j,k+1} - d_{i,j,k-1} \\ n_{i,j,k} &= \frac{g_{i,j,k}}{\|g_{i,j,k}\|} \end{aligned}$$

Niektorí autori poukázali na fakt, že tento filter vyhladzuje na obrázku detaily. Preto bola navrhnutá *adaptívna technika* [8], ktorá berie do úvahy maximum z doprednej a spätnej diferencie:

$$g^x_{i,j,k} = \max(d_{i+1,j,k} - d_{i,j,k},\ d_{i,j,k} - d_{i-1,j,k})$$

Iný prístup zvolil Goss [9], ktorý vyvinul nastaviteľný filter založený na orezaní ideálneho gradientového filtra $\frac{\cos \pi x}{x}$ pomocou Kaiserovho okna. Citlivosť na vyššie frekvencie sa dá nastaviť voľbou parametra α, ktorým sa modifikuje šírka Kaiserovho okna.

Obvykle sú rekonštrukčné filtre na určovanie gradientu používané bez špeciálnych nárokov na vlastnosti objemových dát (okrem predpokladu na ohraničenosť pásma). V prípade, že voxelizujeme geometrické objekty, môžeme využiť výhodu, že máme pod kontrolou vlastnosti vytváraných objemových dát. Konkrétne, dá sa zvoliť voxelizačný filter v súlade s rekonštrukčným filtrom tak, aby sme získali čo najlepšie výsledky. Výhodou filtra používajúceho stredové diferencie je jeho jednoduchá implementácia a nízke výpočtové nároky. Šrámek a Kaufman v [4] ukázali, že pomocou tohto filtra dostaneme skutočnú normálu na povrch len v prípade, že hustota sa v blízkosti povrchu mení lineárne vzhľadom na vzdialenosť od povrchu. To znamená, že stredové diferencie dávajú najlepšie výsledky práve v kombinácii so vzdialenostnými poľami.

1.4 Reprezentovateľnosť objektov

1.4.1 Chyby rekonštrukcie

Voxelizácia je proces, ktorý zo spojitých dát vytvorí diskrétne, preto ho nutne sprevádza istá strata informácie. Pri rekonštrukcii týchto dát sa snažíme dostať pôvodné spojité dáta, ale v sku-

Obrázok 1.2: *Problémy s rekonštrukciou*
(a) detail, (b) hrana

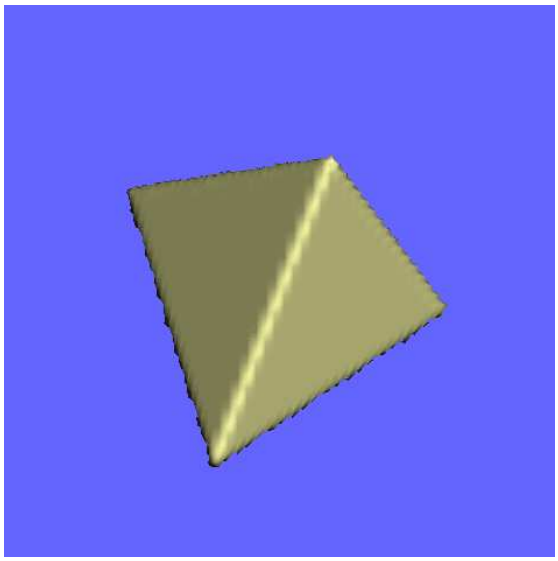
točnosti sme schopní získať len určitú aproximáciu. Nakoľko je táto aproximácia dobrá, závisí od vlastností zobrazovaných objektov a od hustoty vzorkovania. Ak sa snažíme zachytiť objekty, pre ktoré je daná reprezentácia nevhodná, vznikajú pri zobrazovaní chyby v podobe rôznych druhov artefaktov.

Treba si uvedomiť, že voxel nesie informáciu o vzdialenosti len od jedného (najbližšieho) povrchu. V prípade, že sa v blízkosti voxelu nachádzajú viaceré povrchy, tento fakt vo voxeli s jednou hodnotou hustoty nie sme schopní zachytiť a nutne sa to musí prejaviť pri rekonštrukcii. Situácia je naznačená na obrázku 1.2. Na kódovanie povrchu v okolí bodov A aj B potrebujeme informáciu vo voxeli V. Môžeme tu však mať korektnú informáciu pre najviac jeden z bodov A, B — povedzme, že A. Ak pri rekonštrukcii povrchu v okolí bodu B použijeme informáciu z tohto voxelu, prirodzene získame chybné výsledky. Ak sa snažíme kódovať drobný detail, problém sa dá riešiť zvýšením rozlíšenia (voxel V už nebude „v blízkosti“ oboch povrchov). V okolí hrany je však problém zásadnejší — problematické voxely budú existovať pri ľubovoľne veľkom rozlíšení (vedie to k artefaktom ako na obrázku 1.3 (a)). I keď pri dostatočnom rozlíšení takto vzniknutá chyba nemusí byť pri zobrazení viditeľná, nič to nemení na fakte, že „hranaté“ objekty týmto spôsobom nie sú korektne reprezentovateľné.

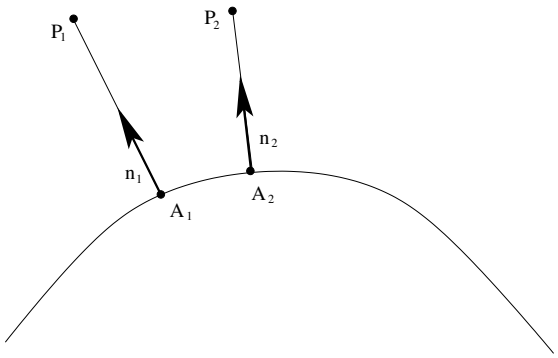
Chyba pri rekonštrukcii vzniká aj vtedy, ak všetky potrebné voxely nesú korektnú informáciu o povrchu. Jej veľkosť závisí od použitej interpolačnej metódy a od lokálnej krivosti povrchu. Na základe obrázku 1.3 (b) sa dá podľa [5] odvodiť vzťah pre hustotu d v bode P_2:

$$d(P_2) = (P_2 - A_1) \cdot \vec{n_1} + (P_2 - A_2) \cdot (\vec{n_2} - \vec{n_1}) - (A_2 - A_1) \cdot \vec{n_1},$$

kde A_1, A_2 sú porade najbližšie body povrchu k bodom P_1, P_2 a $\vec{n_1}$, $\vec{n_2}$ sú povrchové normály v týchto bodoch. Prvý člen rovnosti tvorí lineárnu zložku trojrozmerného vzdialenostného poľa, zvyšné dva členy sú nelineárne. Čím má povrch väčšiu lokálnu krivosť, tým väčšiu váhu majú nelineárne členy. V prípade lokálne rovinného povrchu sú tieto členy nulové, inak sú v blízkosti povrchu veľmi malé, ale s narastajúcou vzdialenosťou začínajú dominovať. Pri použití lineárnych rekonštrukčných filtrov môžeme na základe tohto vzťahu určiť optimálne vzorkovanie pre daný objekt, ak si stanovíme maximálnu akceptovateľnú chybu pri rekonštrukcii povrchu. Krivosť je totiž veličina, ktorá nepriamo úmerne závisí od zvolenej jednotky dĺžky, ktorá je v



(c)



(d)

Obrázok 1.3: *Chyby rekonštrukcie*
(a) artefakty na hranách, (b) ilustrácia k odhadu chyby

našom prípade definovaná vzdialenosťou susedných voxelov. Zvýšením rozlíšenia teda získame pre daný povrch menšiu krivosť a tým zmenšíme chybu rekonštrukcie.

Autori článku [4] uskutočnili sadu experimentov, ktorých cieľom bolo porovnať rôzne voxelizačné filtre z hľadiska chyby, ktorá sa vyskytne pri rekonštrukcii filtráciou získaných dát. Presnú polohu povrchového bodu dostaneme prahovaním na strednej hodnote hustoty, pričom túto získame v spojitom priestore trilineárnou interpoláciou hodnôt z okolitých ôsmich voxelov. Experimenty ukázali, že použitie Barletovho filtra spôsobuje veľkú systematickú chybu — povrch je posunutý dovnútra objektu. Vhodnejšie sa opäť javia byť vzdialenostné techniky, ktoré podobný posun nevyvolávajú.

Ďalší experiment v [4] sa venoval odchýlke skutočnej povrchovej normály od vypočítanej normály. Z výsledkov vyplýva, že použitie gausiánu vedie až k 50-násobne väčšej chybe ako použitie filtra oriented box. Z analýzy je tiež zrejmé, že najvhodnejšia metóda na výpočet normály sú už spomínané stredové diferencie.

Filtrovanie objektov pomocou filtra oriented box vedie k identickej reprezentácii ako technika vzdialenostných polí. Z uvedených poznatkov vychádza použitie vzdialenostných polí spolu s metódou stredových diferencií v tejto diplomovej práci.

1.4.2 Reprezentovateľné objekty

Ako už bolo vyššie naznačené, niektoré objekty nie sú pomocou vzdialenostných polí korektne reprezentovateľné. Podrobnejšie sa tejto problematike venuje článok [10]. Boli tu stanovené dve podmienky, ktoré musí spĺňať objekt vhodný na voxelizáciu:

Podmienka 1: Krivosť povrchu musí byť relatívne malá v porovnaní s rozlíšením.

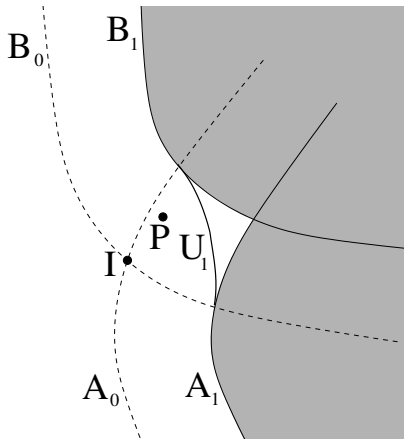
Podmienka 2: Filtre rekonštruujúce povrch a normálu nesmú používať vzorky rozložené na oboch stranách mediálneho povrchu. Mediálny povrch je definovaný ako množina bodov P, pre ktoré existujú aspoň dva povrchové body vo vzdialenosti d, kde d je vzdialenosť bodu P od povrchu. Inými slovami, pre P existuje viac ako jeden najbližší povrchový bod.

Dôvod na prvú podmienku je zrejmý z toho, čo bolo spomínané v predošlých statiach — s rastúcou krivosťou rastie chyba rekonštrukcie. Druhá podmienka už bola taktiež neformálne vysvetlená — vyplýva z problémov, ktoré ilustruje obrázok 1.2.

Na základe uvedených podmienok dospeli autori [10] k záveru, že geometrický objekt X je vhodný na voxelizáciu s daným rozlíšením, ak X je S_r-*otvorená* aj S_r-*uzavretá*, kde $r > \sqrt{6}$ je zvolené tak, aby chyba rekonštrukcie bola akceptovateľná pre danú aplikáciu. Pritom treba mať na pamäti, že výberom r je určené aj rozlíšenie, pretože r je vo voxelových jednotkách. Kritérium otvorenosti a uzavretosti sa dá intuitívne popísať ako vlastnosť povrchu, po ktorom sa dá z oboch strán kotúľať guľa s polomerom r.

S problematikou reprezentovateľnosti geometrických objektov úzko súvisí realizácia CSG operácií s objemovými dátami. Zatiaľčo CSG operácie s binárnymi dátami sa dajú implementovať veľmi jednoducho pomocou blokových operácií, pri vzdialenostných poliach je situácia komplikovanejšia. Hlavný problém spôsobuje fakt, že výsledok CSG operácie je vo všeobecnosti „hranatý“, a teda nespĺňa kritérium otvorenosti a uzavretosti. Cieľom je preto vytvoriť pomocou CSG operácie objekt, ktorý toto kritérium spĺňa a čo najviac sa približuje želanému výsledku. V [11] bol zvolený postup, kedy sa CSG operácia vykoná bez ohľadu na „hranatosť“ výsledku a artefakty na hranách sa potom korigujú následnou *revoxelizáciou*. Táto metóda vedie k istému vizuálnemu zlepšeniu, ale nezaručuje, že spomínané kritérium bude splnené. Korektnejší sa javí prístup, kedy priamo voxelizujeme reprezentovateľný objekt. V článku [12] bola navrhnutá nasledovná schéma: (a) Rekonštruovať pôvodné telesá z ich objemovej reprezentácie; (b) vykonať CSG operáciu v spojitom priestore; (c) modifikovať výsledok tak, aby spĺňal kritérium otvorenosti a uzavretosti; (d) následnou voxelizáciou výsledku získať konečnú objemovú reprezentáciu. Prezentovaný bol algoritmus, ktorý nepostupuje presne podľa tejto schémy, ale dáva rovnaký výsledok.

Algoritmus pracuje s jedným vzorkovaným objemom a jedným analytickým objektom, ktorý prostredníctvom CSG operácie modifikuje daný objem. Proces prebieha v dvoch fázach, pričom v oboch sa prechádza cez všetky voxely. Popíšeme si ho pre realizáciu zjednotenia objektov (prienik a rozdiel sa vytvoria podobne). Pri vytváraní vzdialenostného poľa zjednotenia nám pre väčšinu voxelov stačí poznať vzdialenosť voxelu od jednotlivých objektov, pričom výslednú hodnotu získame ako maximum (respektíve minimum — v závislosti od zvolenej znamienkovej konvencie) z týchto dvoch vzdialeností. To neplatí len v blízkosti prieniku povrchov, kde treba vzdialenosť počítať komplikovanejšie. Situácia je naznačená na obrázku 1.4. Objekty A, B majú porade vonkajší povrch A_0, B_0 a vnútorný povrch A_1, B_1 (hustota sa lineárne mení medzi týmito povrchmi, inde je konštantná). Priesečníkom vonkajších povrchov je množina I, vo všeobecnosti priestorová krivka. Našou snahou je vytvoriť zjednotenie tak, aby vznikol vnútorný povrch U_1. Problematický (*nekonzistentný*) je bod P, ktorý sa nachádza v blízkosti I. V prvej fáze sa konštruuje množina I a označujú sa nekonzistentné voxely, v druhej fáze sa voxelom priraďuje hodnota,



Obrázok 1.4: *Problémová oblasť pri konštrukcii zjednotenia objektov*

pričom pre nekonzistentné voxely sa na výpočet použije vzdialenosť voxelu od I.

Jedným z cieľov tejto diplomovej práce je navrhnúť spôsob, ako CSG operácie realizovať pre dva navzorkované objemy (bez informácie o ich analytickom popise) len pomocou jedného prechodu všetkými voxelmi bez nutnosti konštruovať množinu I.

Kapitola 2

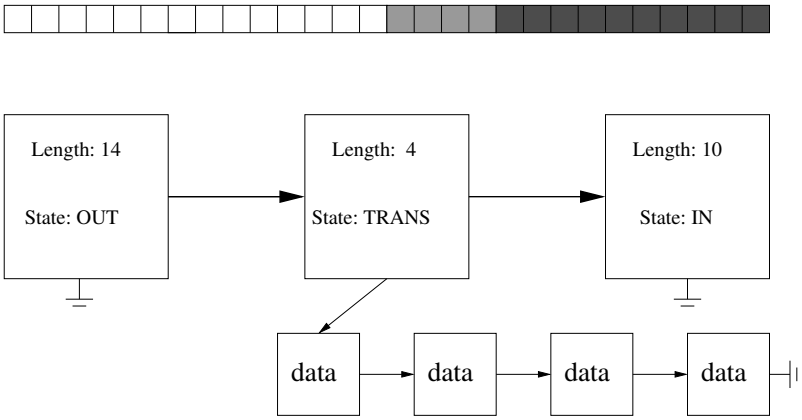
Objemová reprezentácia voxelizovaných dát

2.1 RL kompresia

Jednou z najčastešie citovaných nevýhod objemovej grafiky je jej vysoká spotreba pamäte. Hoci neustály pokrok hardvéru túto prekážku postupne odstraňuje, pamäte nikdy nie je nazvyš, nakoľko s rastúcov výkonnosťou techniky rastú i naše požiadavky na to, čo pomocou nej môžeme dosiahnuť. Preto boli navrhnuté rôzne spôsoby, ako znížiť pamäťové nároky objemovej reprezentácie.

V úvodnej kapitole boli spomenuté adaptívne vzdialenostné polia [6], ktoré využívajú hierarchické delenie priestoru. Nevýhodou hierarchicky uložených dát sú isté komplikácie pri manipulácii s nimi. V ďalších riadkoch prezentujeme metódu, ktorá volí iný prístup. Základom je typ kompresie známy pod názvom *run-length (RL) encoding*, ktorý sme modifikovali tak, aby umožňoval efektívne reprezentovať limitované vzdialenostné polia.

Naša implementácia *RL kompresie* vychádza z faktu, že informácia o objekte sa nachádza



Obrázok 2.1: *Reprezentácia komprimovaného riadku*

len vo voxeloch v blízkom okolí povrchu. Typický riadok v mriežke pozostáva z niekoľkých dlhých sekvencií vonkajších alebo vnútorných voxelov striedaných s krátkymi sekvenciami voxelov, ktoré ležia v prechodovej oblasti. Toto pozorovanie nás vedie k myšlienke rozdeliť riadok na *segmenty* troch typov: *vonkajšie*, *vnútorné* a *prechodové*. Na kódovanie prvých dvoch typov segmentov si stačí pamätať ich dĺžku a stav (*OUT, IN*). Prechodové segmenty majú ešte naviac smerník na spájaný zoznam voxelov, ktoré pokrývajú. Princíp kompresie je ilustrovaný obrázkom 2.1. Celá mriežka je uložená ako dvojrozmerné pole komprimovaných riadkov. Podobnú metódu na reprezentovanie objemu už použili autori článkov [13, 14], avšak nie pre vzdialenostné polia.

2.2 Typy voxelov

Pôvodná metóda limitovaných vzdialenostných polí si vystačí s voxelom nesúcim iba informáciu o hustote, ktorá je daná vzdialenosťou bodu od povrchu. Keď pri vizualizácii potrebujeme poznať gradient hustoty kvôli určeniu normály, zvyčajne ho vypočítame na základe znalosti hustôt okolitých voxelov (najčastejšie pomocou zmienených stredových diferencií). V prípade, že voxelizujeme geometrické objekty, kde gradient poznáme z ich analytického popisu, ponúka sa nám možnosť ukladať do voxelu tento gradient spolu s hustotou. Jednak tým pri zobrazovaní ušetríme čas, jednak tým môžeme dosiahnuť väčšiu presnosť, nakoľko budeme používať skutočný gradient namiesto odhadnutého. Prirodzene, zvýšime tým pamäťové nároky, ale vďaka RL komresii tento nárast nebude až taký kritický.

Presnejšie povedané, nepotrebujeme si pamätať vlastný gradient, ale len jeho smer, ktorý získame normalizáciou daného vektora. Priestorový vektor bežne reprezentujeme trojzložkovo (x, y, z), ale na zapamätanie smeru nám stačia dve zložky (α, β), ktoré získame napríklad známou parametrizáciou guľového povrchu:

$$\begin{aligned} x &= \cos\alpha \cos\beta \\ y &= \sin\alpha \cos\beta \\ z &= \sin\beta. \end{aligned}$$

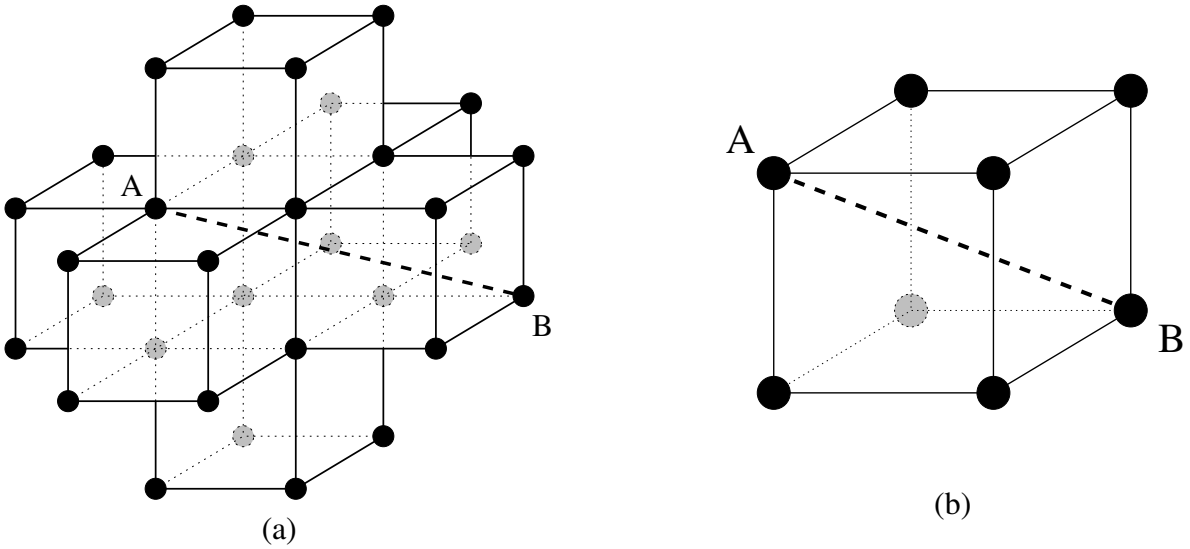
Tým ušetríme časť pamäte, aj keď na druhej strane prevod do sférických súradníc a naspäť nás bude stáť istý čas.

Ďalšou dôležitou otázkou je, koľko bajtov použiť na ukladanie hustoty a gradientu. Pre potreby testovania sme implementovali všetky kombinácie jedno-, dvoj- a štvorbajtovej hustoty s jedným, dvoma a štyrmi bajtami pre každú zložku gradientu. Pritom voxely môžu byť troch druhov:

vxtPlainVoxel: Ukladá sa len hustota.

vxtGradVoxel: Ukladá sa hustota aj gradient (trojzložkovo).

vxtSphGradVoxel: Ukladá sa hustota a gradient pomocou sférických súradníc (dvojzložkovo).



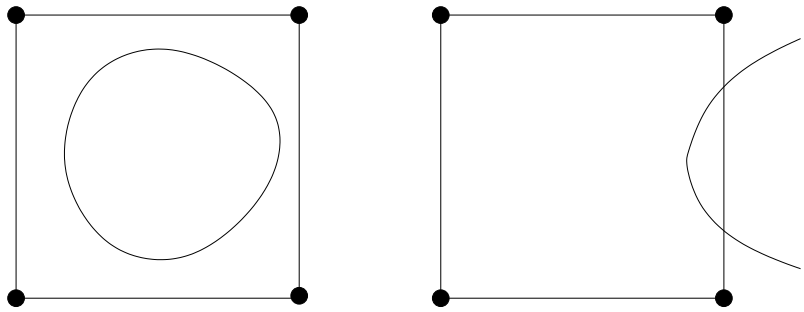
Obrázok 2.2: *Konfigurácia voxelov potrebná na výpočet normály (a) voxel bez gradientu, (b) voxel s gradientom*

V prvom prípade dostávame 3 možnosti a vo zvyšných prípadoch po 9 možností podľa rôznej presnosti pamätania hustoty a gradientu, celkovo máme teda 21 rôznych voxelov. Pravda, niektoré možnosti nie sú veľmi zmysluplné, napríklad kombinácia veľmi presného gradientu s málo presnou hustotou. Ešte treba poznamenať, že pri vlastnej implementácii zaberajú niektoré voxely viac miesta, ako je teoreticky vypočítaná hodnota, nakoľko pri zložených štruktúrach dochádza k zarovnávaniu na násobky dvoch alebo dokonca štyroch bajtov.

S výberom voxelu úzko súvisí aj voľba šírky prechodovej oblasti. V prípade voxelov bez gradientu totiž potrebujeme na výpočet povrchovej normály širšie okolie bodu ako pri voxeloch s gradientom. Konfigurácia voxelov, pomocou ktorých získame normálu, je znázornená na obrázku 2.2. Ak sa povrchový bod nachádza v tesnej blízkosti bodu A, najvzdialenejší potrebný voxel (B) je vo vzdialenosti $\sqrt{6}$ pre bezgradientový voxel, inak je to len $\sqrt{3}$. Tieto čísla sú preto smerodajné pre stanovenie polomeru prechodovej oblasti. Hodnota $\sqrt{6}$ nemusí byť striktne dodržaná. Ak bude polomer o niečo menší, chyba sa dramaticky neprejaví, pretože voxely na kraji prechodovej oblasti majú pri výpočte normály malú váhu. My sa však budeme napriek tomu držať teoreticky odôvodnených hodnôt, teda $\sqrt{6}$ pre `vxtPlainVoxel` a $\sqrt{3}$ pre `vxtGradVoxel` a `vxtSphGradVoxel`.

2.3 Voxelizácia

RL kompresiu sme testovali pre implicitne definované povrchy, pričom na ich voxelizáciu sme použili dve rôzne metódy. Prvá (prevzatá z [15]) je založená na oktantovom delení priestoru na základe testu homogenity, druhá napĺňa mriežku po vrstvách a využíva efektívny prechod



Obrázok 2.3: *Nedokonalosť testu homogenity*

riadkom. Na výpočet vzdialenosti bodu P od povrchu využívame vzťah

$$d(P) = \frac{f(P)}{\| \bigtriangledown f(P) \|},$$

kde f je funkcia definujúca implicitný povrch $f(P) = 0$ a $\| \bigtriangledown f(P) \|$ je veľkosť jej gradientu. Táto lineárna aproximácia je v blízkosti povrchu v poriadku za predpokladu, že veľkosť gradientu sa nemení príliš prudko.

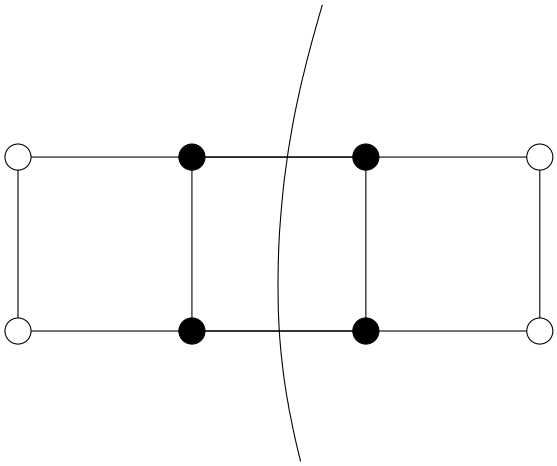
Pri voxelizácii pomocou oktantového delenia priestoru sa objem najskôr rozdelí na dostatočne malé bloky voxelov (napríklad $8 \times 8 \times 8$). Vo vrcholoch každého bloku sa vypočítajú hustoty. Ak sú tieto všetky rovnaké, blok nepretína prechodovú oblasť v okolí povrchu, preto všetky voxely v bloku vyplníme konštantnou hodnotou hustoty. V opačnom prípade blok rekurzívne delíme, pokým nie je kritérium homogenity splnené alebo neskončíme na úrovni blokov $2 \times 2 \times 2$. Prirodzene, test na homogenitu bloku nie je úplne dokonalý. Na obrázku 2.3 sú zobrazené situácie, kedy tento test zlyhá. Je to v prípade, keď povrch prechádza len okrajom bloku, alebo ak celý objekt leží vo vnútri bloku. Rozmery najväčšieho bloku preto musia byť zvolené vhodne s ohľadom na očakávané vlastnosti voxelizovaného objektu.

Jadrom druhej metódy je voxelizácia jedného riadku. Od začiatku riadku postupne dopĺňame hodnoty voxelom, cez ktoré prechádzame. Ak sa daný voxel nachádza v prechodovej oblasti, postúpime v ďalšom kroku do susedného voxelu. Ak je mimo nej, snažíme sa odhadnúť jeho vzdialenosť od povrchu a príslušný počet voxelov preskočiť. Problematickým miestom algoritmu je odhad vzdialenosti. Vyššie uvedený vzťah je dosť presný iba v tesnom okolí povrchu. Nám však stačí len hrubý dolný odhad. Ak preskočíme príliš ďaleko (nasledovný voxel má inú hustotu ako predošlý), skok skorigujeme tým, že sa posunieme len o polovičnú vzdialenosť.

Voxelizácia riadku je súčasťou zložitejšieho mechanizmu, ktorý napĺňa mriežku pomocou *zametacích rovín*. Pôvodne bolo jeho cieľom zisťovať počas voxelizácie, či je voxel v prechodovej oblasti naozaj potrebný pre rekonštrukciu povrchu (na obrázku 2.4 je príklad nepotrebných voxelov). Zametacími rozumieme dve susedné rovnobežné roviny v mriežke, ktoré počas voxelizácie postupne prejdú celým objemom. Počas tohto procesu sa striedajú tri fázy:

1. Naplnenie zametacej roviny pomocou riadkovej voxelizácie.

2. Poznačenie užitočných voxelov.



Obrázok 2.4: *Príklad nepotrebných voxelov v prechodovej oblasti*
Biele voxely ležia približne 1.5 *(menej ako* $\sqrt{3}$*) VU od povrchu, ale na jeho kódovanie nie sú potrebné (pri reprezentácii* `vxtGradVoxel` *a* `vxtSphGradVoxel`*).*

3. Prevod údajov zo zametacej roviny do mriežky.

Počas druhej fázy sa pre prechodové voxely kontroluje, či sú súčasťou aspoň jednej *bunky*, ktorú pretína povrch (bunka je najmenšia kocka s vrcholmi vo voxeloch). Preto sú potrebné dve zametacie roviny.

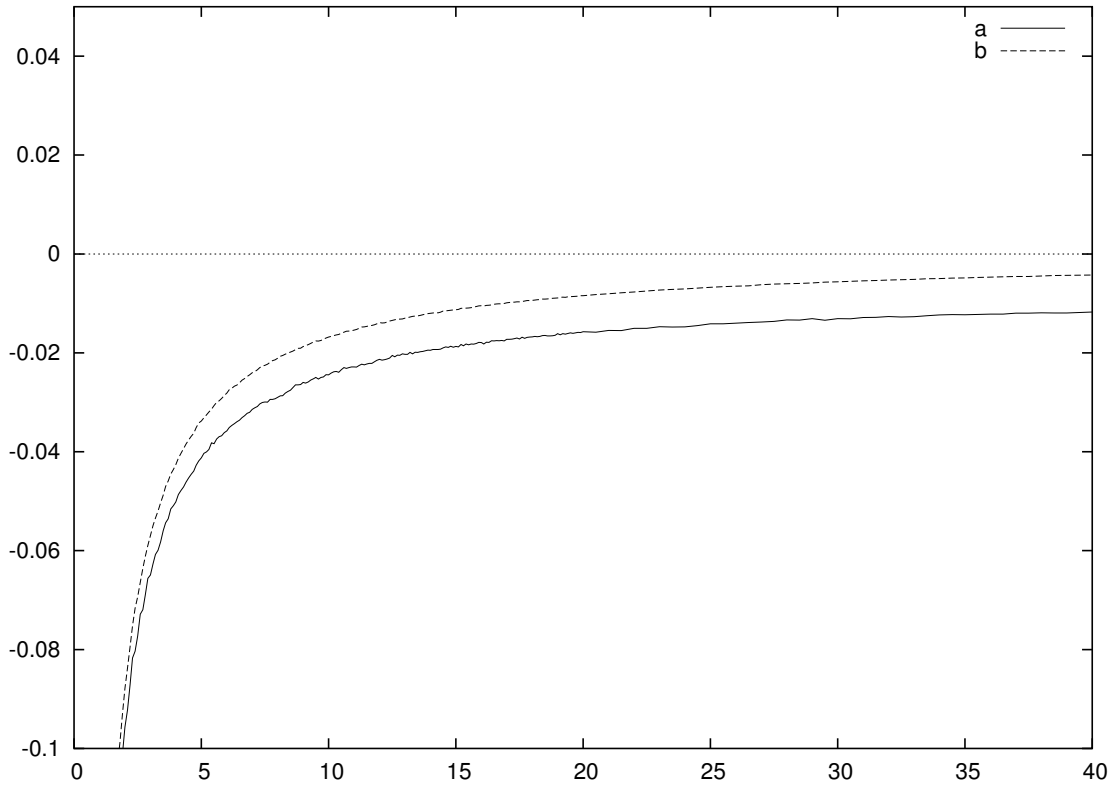
Cieľom tejto metódy bola určitá úspora pamäte vynechaním neužitočných voxelov. Bohužiaľ, neskôr sa ukázalo, že na realizáciu sofistikovanejších typov CSG operácií je potrebná aj táto „zahodená“ informácia. Preto treba druhú fázu procesu vypustiť a vystačíme si vlastne len s efektívnym prechodom riadkov.

Z pozorovania vyplýva, že obe popísané metódy majú podobnú časovú náročnosť. Pri testovaní popísanom na ďalších stranách bola použitá voxelizácia pomocou zametacích rovín.

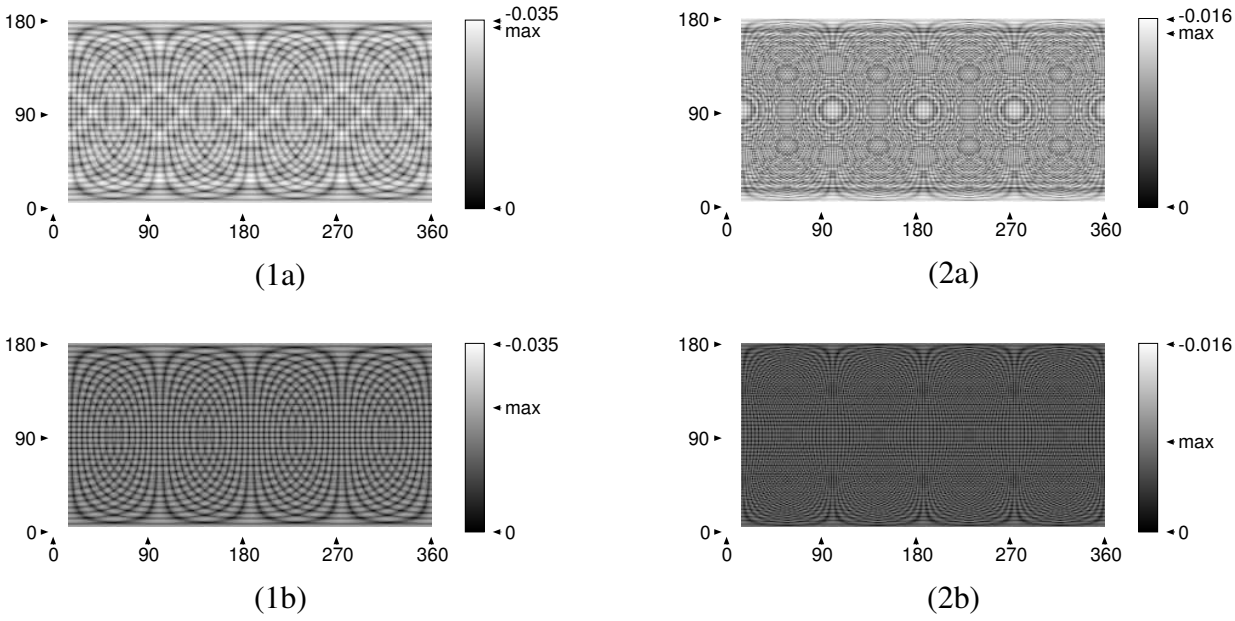
2.4 Výsledky testov

2.4.1 Presnosť rekonštrukcie

Na testovanie presnosti rekonštrukcie povrchu a normály sme použili postup navrhnutý v článku [4]. Základom je guľa, z ktorej stredu vystrelíme lúč a hľadáme jeho priesečník s povrchom. V nájdenom bode potom určíme normálu a zistíme rozdiel medzi skutočným a vypočítaným povrchom a odchýlku normály od správneho smeru. Na výpočet povrchu a normály používame trilineárnu interpoláciu okolitých ôsmich voxelov. V prípade voxelov bez gradientu odhadneme normálu v týchto voxeloch pomocou stredových diferencií. Lúče vysielame rovnomerne vo všetkých smeroch. Aby sme minimalizovali vplyv výberu stredu gule vzhľadom na mriežku, experiment opakujeme pre 125 ($5 \times 5 \times 5$) rovnomerne rozložených polôh tohto stredu vo vnútri priestorovej bunky. Zaujíma nás, ako sa spomínané chyby menia v závislosti od krivosti povrchu, preto meníme polomer gule od $1\,VU$ až po $40\,VU$.



Obrázok 2.5: *Závislosť chyby určovania polohy od krivosti povrchu*
Na osi x je polomer gule (vo VU), na osi y je chyba (vo VU). Presnosť voxelov: (a) 1 bajt na zapamätanie hustoty, (b) 2 bajty na zapamätanie hustoty.



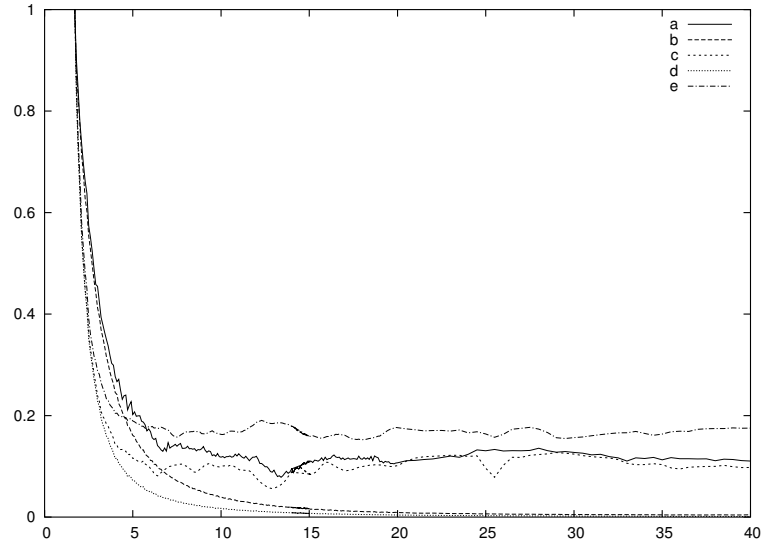
Obrázok 2.6: *Rozloženie chyby určovania polohy*
Polomer gule: (1) R=11VU, (2) R=33VU. Presnosť voxelov: (a) 1 bajt na zapamätanie hustoty, (b) 2 bajty na zapamätanie hustoty. Intenzita charakterizuje chybu polohy v danom smere, kde smer je vyjadrený v sférických súradniciach.

Test ukázal, že na reprezentáciu hustoty plne postačujú 2 bajty, nakoľko štvorbajtová hustota neprináša väčšiu presnosť. Na obrázku 2.5 vidíme porovnanie chyby pre 1-bajtovú a 2-bajtovú hustotu. V oboch prípadoch je vypočítaný povrch oproti skutočnému posunutý dovnútra gule. Pri polomere $40VU$ je odchýlka pri použití dvojbajtovej hustoty približne trikrát menšia ako pri jednobajtovej.

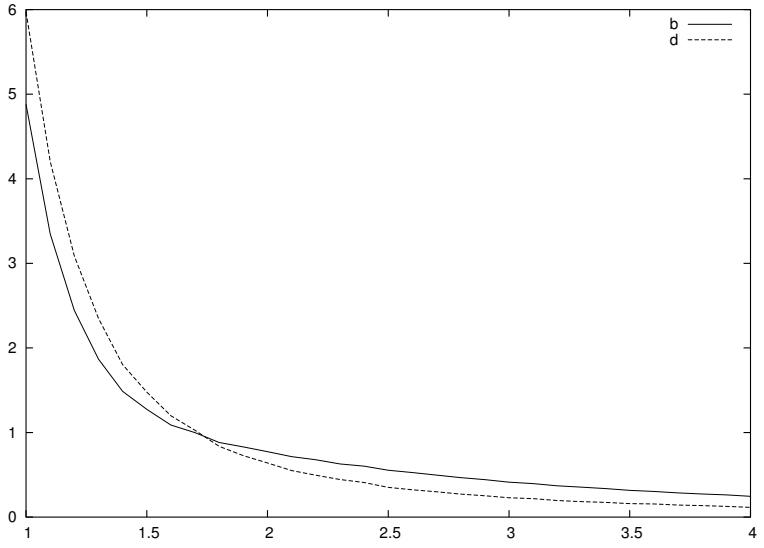
Obrázok 2.6 vizualizuje rozloženie chyby v rôznych smeroch pre guľu so stredom v prostriedku mriežkovej bunky. Vidíme, že chybový vzor je podľa očakávania symetrický a so zväčšujúcim polomerom sa zjemňuje.

Ako bolo vyššie spomenuté, teoreticky máme na výber až 21 rôznych možností, ako zostaviť voxel. Podrobne ich všetky testovať by bolo zložité a výsledky by neboli veľmi prehľadné. Našťastie experiment je usporiadaný tak, že presnosť detekcie povrchu má minimálny vplyv na presnosť výpočtu normály. Preto pri voxeloch so zapamätaným gradientom si stačí všímať len počet bajtov na reprezentáciu gradientu (nie hustoty). Výsledky ukázali, podobne ako pri hustote, že na pamätanie gradientu stačia 2 bajty na každú zložku, pretože so 4 bajtami dosiahneme prakticky rovnakú chybu. Dôvod je ten, že táto chyba je ovplyvnená oveľa viac diskretizáciou priestoru ako kvantizáciou veličín.

Graf 2.7 (1) zobrazuje závislosť chyby normály od krivosti povrchu pre päť vybraných typov voxelov. Ukazuje sa, že s dvojbajtovou presnosťou dostaneme rádovo menšiu odchýlku ako s jednobajtovou, a to pre voxely s gradientom i bez neho. Z 2.7 (2) vidíme, že vďaka zapamätanému gradientu dosahujeme vyššiu presnosť. Pre guľu s polomerom $4VU$ je táto presnosť

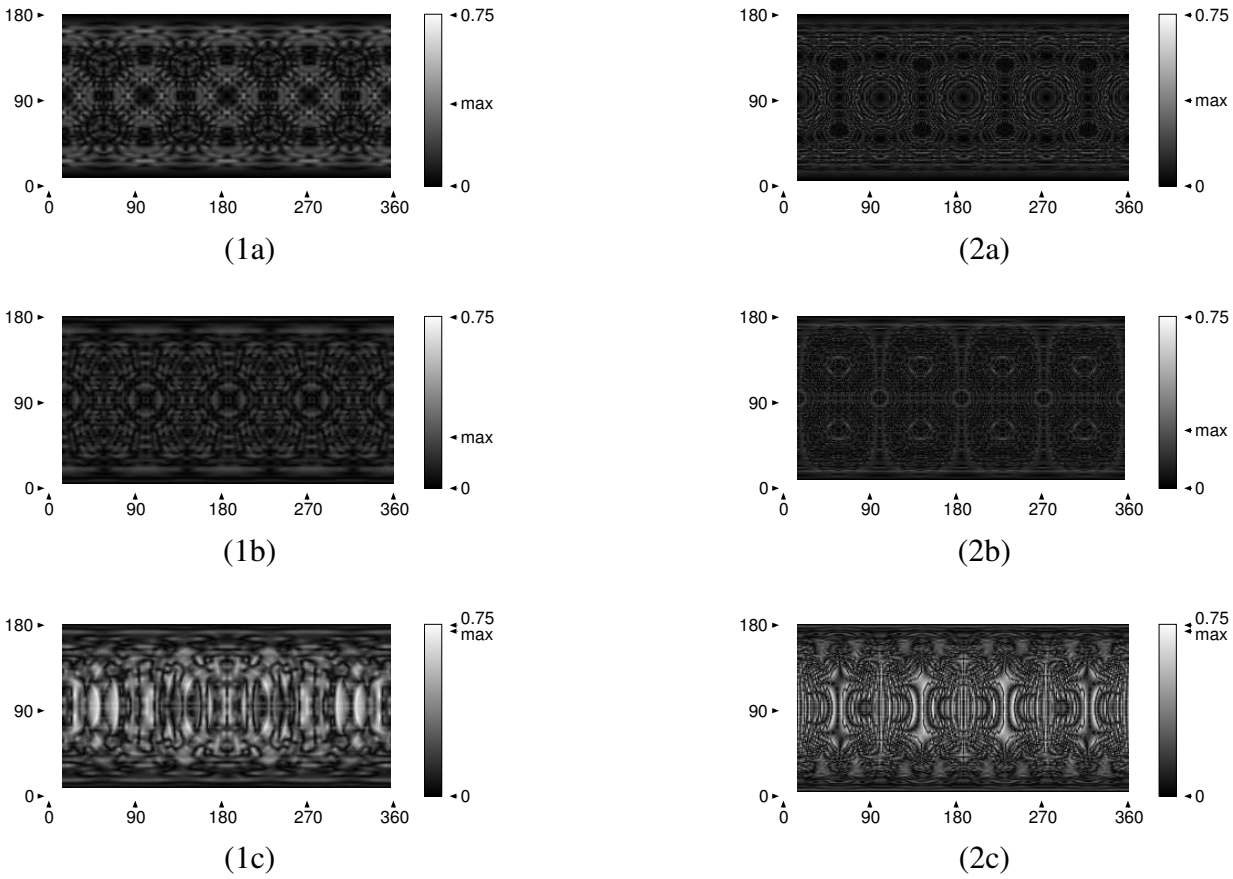


(1)



(2)

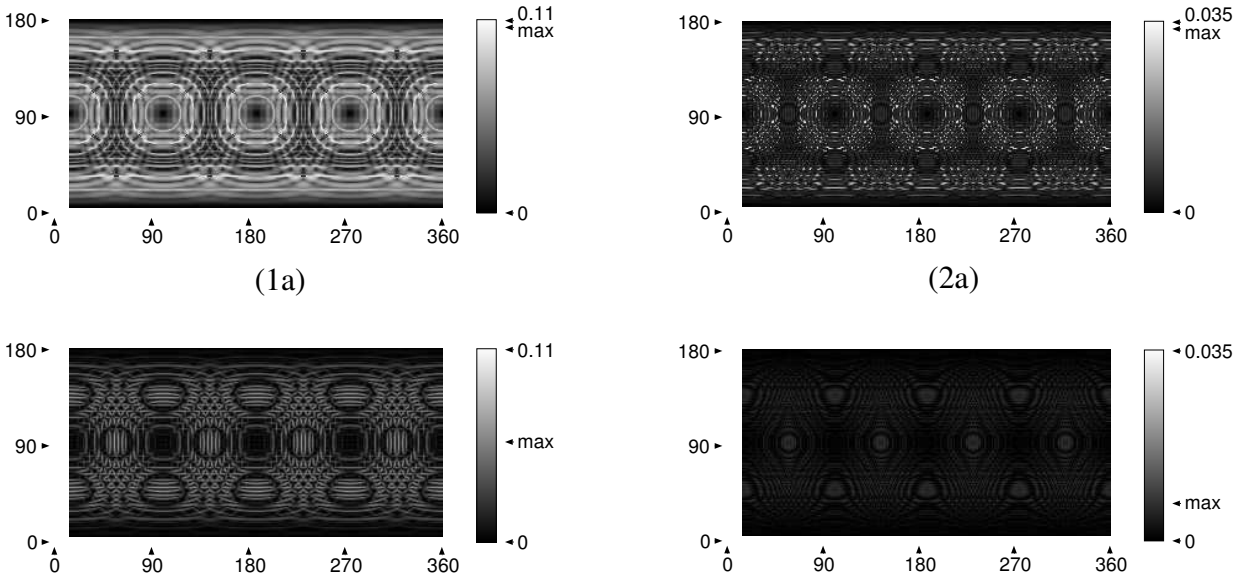
Obrázok 2.7: *Závislosť chyby určovania normály od krivosti povrchu*
Na osi x je polomer gule (vo VU), na osi y je odchýlka od skutočnej normály (v stupňoch). Presnosť voxelov: voxel bez gradientu s 1- a 2-bajtovou presnosťou hustoty (a, b); voxel s gradientom s 1- a 2-bajtovou presnosťou hustoty aj jednotlivých zložiek gradientu (c, d); voxel s komprimovaným gradientom s 1-bajtovou presnosťou (e). Na spodnom grafe (2) je detailnejší pohľad na rovnakú závislosť pre voxely s dvojbajtovou presnosťou (b, d).



Obrázok 2.8: *Rozloženie chyby určovania normály pre objemy s 1-bajtovou presnosť ou na zapamätanie hustoty a gradientu*
Polomer gule: (1) R=11VU, (2) R=33VU. Typy voxelov: (a) bez gradientu, (b) s gradientom, (c) s komprimovaným gradientom. Intenzita charakterizuje chybu normály v danom smere (čísla sú v stupňoch), kde smer je vyjadrený v sférických súradniciach.

dvakrát a pre polomer $40VU$ až štyrikrát väčšia. Ďalší zaujímavý poznatok je ohľadom voxelu s jednobajtovým komprimovaným gradientom. Tento dosahuje horšie výsledky ako voxel bez gradientu. Vysvetlenie nájdeme na obrázku 2.8, kde vidíme, že chyba je koncentrovaná okolo roviny xy, zatiaľčo v smere osi z je minimálna. Dôvodom sú použité sférické súradnice, ktoré povrch gule nepokryjú rovnomerne, takže zvislé smery máme zachytené presnejšie ako vodorovné. Tento jav sa však pri viacbajtovej reprezentácii gradientu neprejaví, voxel s dvojbajtovým komprimovaným gradientom dosiahol prakticky rovnaké výsledky ako voxel s nekomprimovaným gradientom.

Záver z tejto analýzy hovorí, že z hľadiska presnosti a pamäťovej náročnosti je najvýhodnejšie používať voxel s dvojbajtovou hustotou a dvojbajtovým komprimovaným gradientom.



Obrázok 2.9: *Rozloženie chyby určovania normály pre objemy s 2-bajtovou presnosť ou na zapamätanie hustoty a gradientu*
Polomer gule: (1) R=11VU, (2) R=33VU. Typy voxelov: (a) bez gradientu, (b) s gradientom. Intenzita charakterizuje chybu normály v danom smere (čísla sú v stupňoch), kde smer je vyjadrený v sférických súradniciach.

2.4.2 Spotreba pamäte

Teoreticky môžeme pamäťovú zložitosť RL kompresie odhadnúť jednoduchou úvahou. Je daná počtom voxelov v prechodovej oblasti v a počtom riadkových segmentov s. Zrejme v lineárne závisí od veľkosti povrchu a s vypočítame ako $r+2p$, kde r je počet riadkov v mriežke a p je počet priesečníkov povrchu s riadkom, nakoľko s každým priesečníkom v riadku pribudnú dva segmenty. Aj p je zhora ohraničené veľkosťou povrchu a $r = n^2$, preto pre pamäťovú zložitosť S_{RL} platí

$$S_{RL}(k,n) = \theta(n^2 + kn^2),$$

kde k je faktor charakterizujúci veľkosť povrchu a n je rozlíšenie mriežky (predpokladáme rozmery $n \times n \times n$). Pre danú scénu teda veľkosť pamäte kvadraticky závisí od rozlíšenia na rozdiel od objemu bez kompresie, kde je táto závislosť kubická. Prirodzene, RL kompresia je vhodná pre scény s dostatočne malým faktorom k.

Uskutočnené experimenty tieto teoretické závery potvrdzujú. Testovali sme viaceré objekty, konkrétne prázdny objem, kocku, guľu, zjednotenie gule s kockou, pravidelný štvorsten a osemsten, *onion* — implicitný povrch daný funkciou

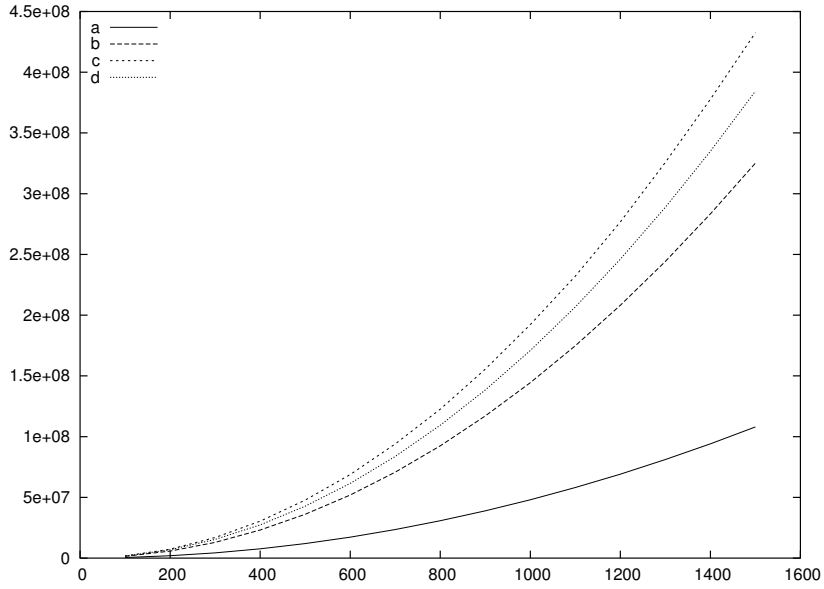
$$f(x,y,z) = \sqrt{y^2+z^2} - 0,4\,\frac{\cos 2\pi x\ +\ 1}{2} \cdot \left(0,3 \cdot \left|\cos\left(10\pi x + 4\,\mathrm{arctg}\,\frac{z}{y+\frac{1}{1000}}\right)\right| + 0,7\right),$$

supergul'u — danú funkciou

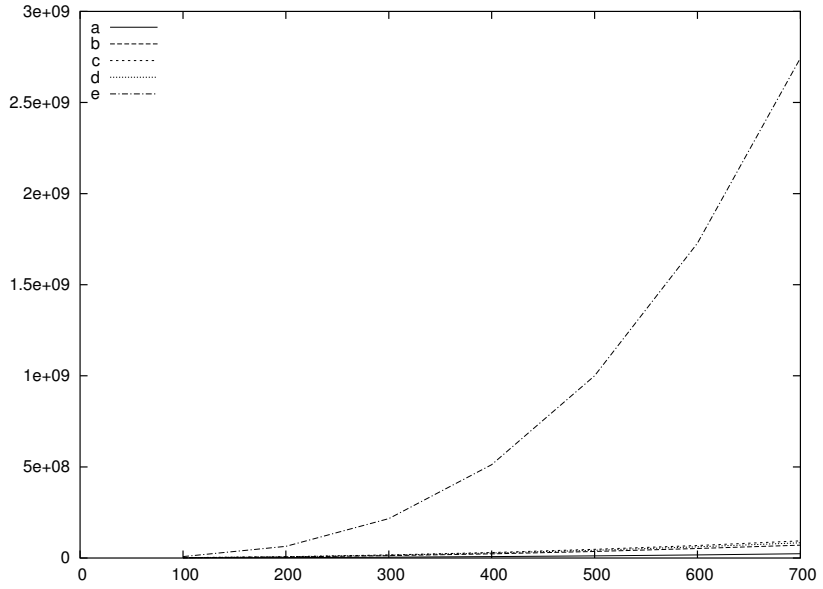
$$f(x,y,z) = \left(|x|^{\frac{2}{p}} + |y|^{\frac{2}{p}}\right)^{\frac{p}{q}} + |z|^{\frac{2}{q}} - r^{\frac{2}{q}}$$

(pre $p = 0,3$; $q = 0,7$; $r = 0,5$), *sphereflake* — zjednotenie 91 gúľ rôznych veľkostí (pozri obrázok 2.12). Všetky mali závislosť pamäte od rozlíšenia veľmi podobnú, niektoré z nich sú znázornené v grafe 2.10 (1). Tá istá závislosť je pre lepšiu názornosť zobrazená na obrázku 2.10 (2) s pridaním nekomprimovaného objemu. Ako vidno z ďalšieho grafu (2.11), pri mriežke 1500^3 potrebujeme pre všetky testované objekty len necelé 2% pamäte z veľkosti nekomprimovaného objemu.

Obrázok 2.13 ilustruje, ako závisí spotreba pamäte od veľkosti voxela. Vidíme, že hoci je voxel s gradientom až 4-krát väčší ako bez gradientu, nároky na pamäť nie sú ani dvakrát väčšie (pre testované objekty s malým faktorom k). Dôvod je ten, že určitá časť pamäte slúži na reprezentáciu riadkových segmentov a smerníkov medzi voxelmi — na túto pamäť nemá vplyv veľkosť voxelov. Rozdiel medzi voxelom s plným a komprimovaným gradientom preto nie je veľmi výrazný.

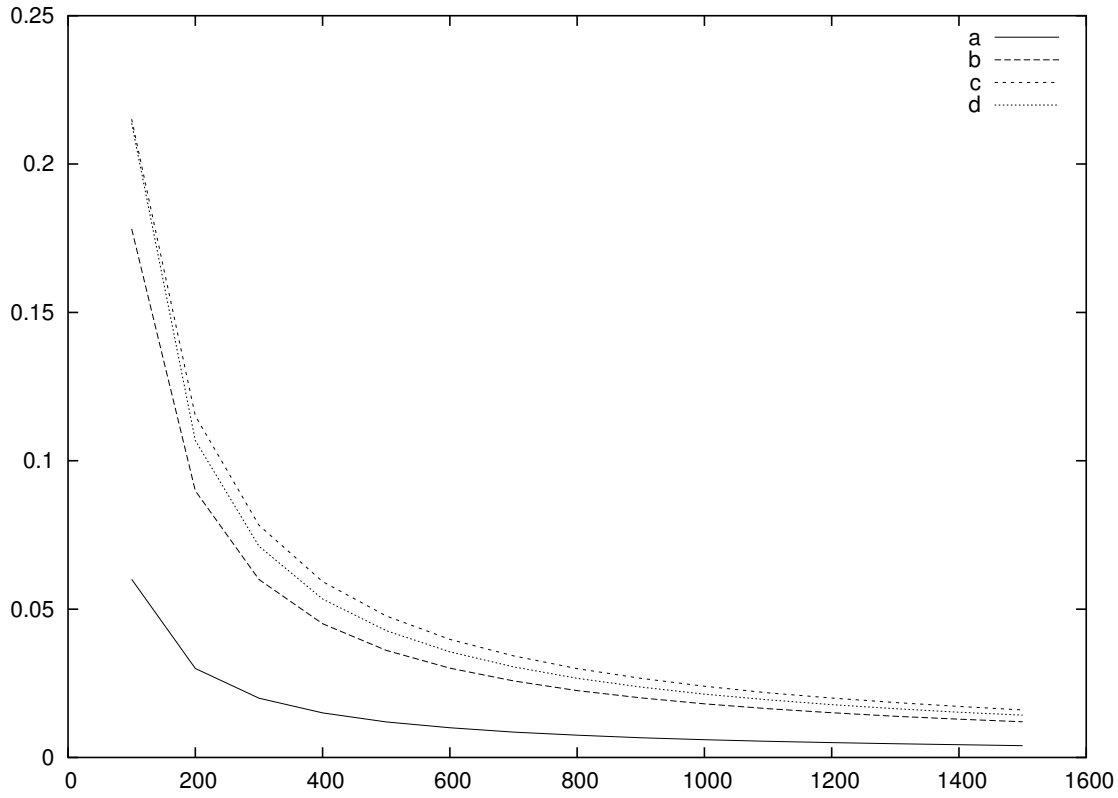


(1)

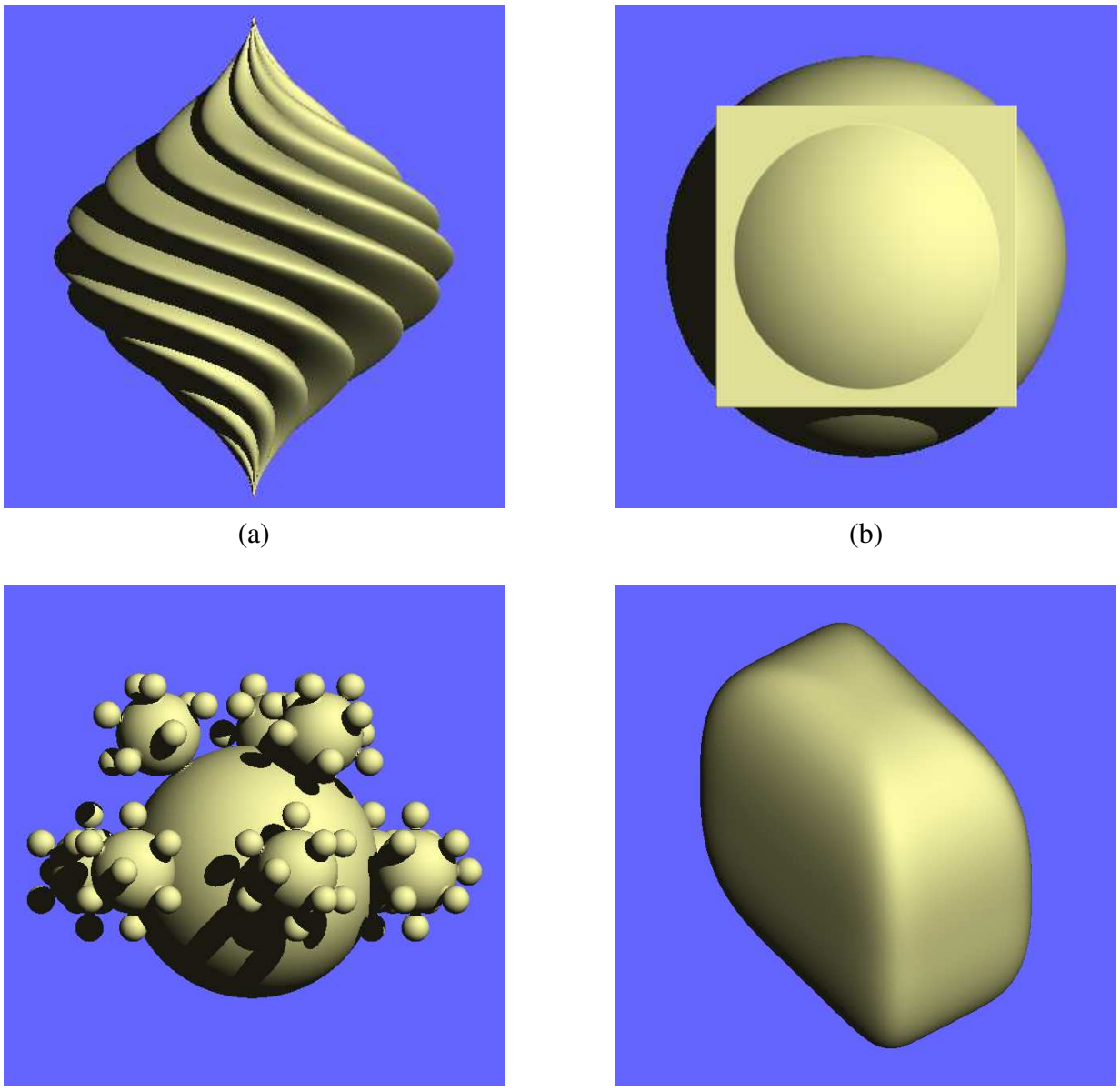


(2)

Obrázok 2.10: *Závislosť spotreby pamäte od rozlíšenia pre rôzne objekty*
1: (a) prázdny objem, (b) guľa, (c) onion, (d) zjednotenie gule s kockou. 2: rovnaká závislosť pre porovnanie s nekomprimovaným objemom (e). Použitý bol voxel s dvojbajtovou hustotou a nekomprimovaným dvojbajtovým gradientom.



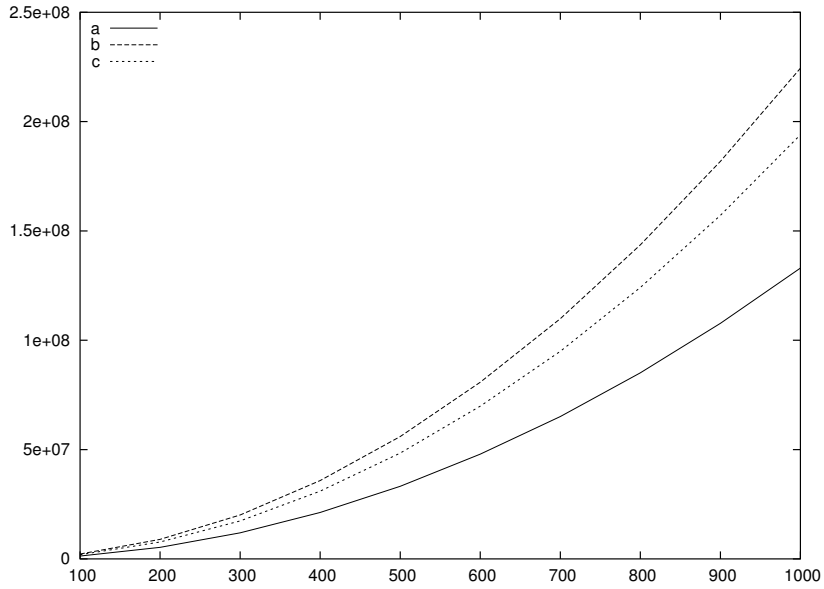
Obrázok 2.11: *Relatívna spotreba pamäte pre rôzne objekty*
Závislosť vyjadruje pomer medzi komprimovaným a nekomprimovaným objemom. Objekty: (a) prázdny objem, (b) guľa, (c) onion, (d) zjednotenie gule s kockou.



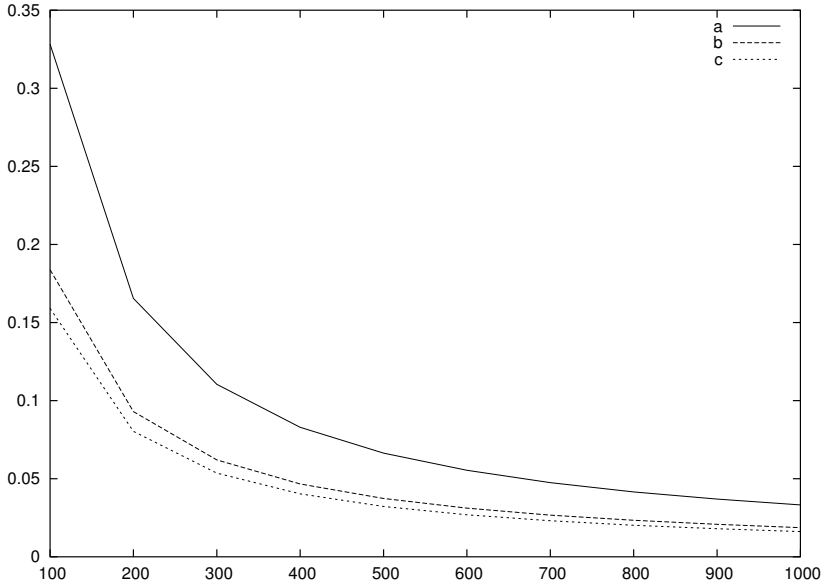
(a) (b)

(c) (d)

Obrázok 2.12: *Príklady voxelizovaných objektov*
(a) onion, (b) zjednotenie gule s kockou, (c) sphereflake, (d) superguľa

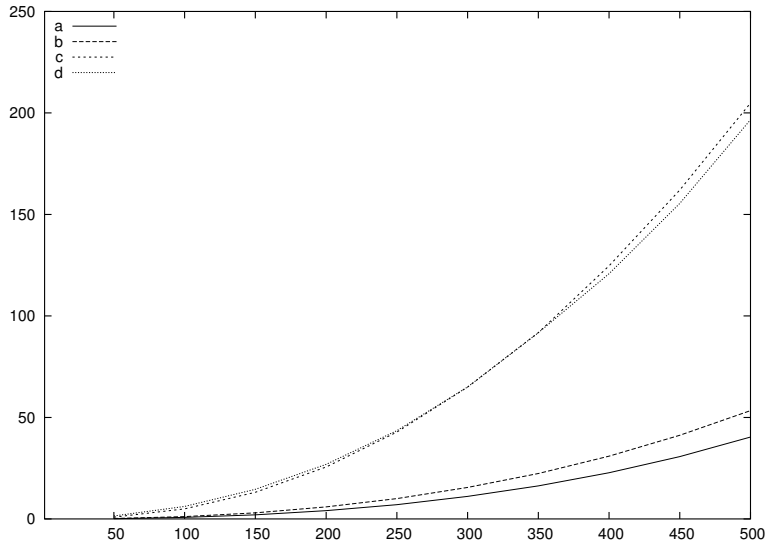


(1)



(2)

Obrázok 2.13: *Závislosť spotreby pamäte od rozlíšenia pre rôzne typy voxelov (1) absolútna spotreba pamäte, (2) relatívna spotreba v porovnaní s nekomprimovaným objemom. Typy voxelov: (a) voxel bez gradientu, (b) voxel s gradientom, (c) voxel s komprimovaným gradientom. Testovací objekt bola guľa a na reprezentáciu hustoty aj gradientu bola použitá štvorbajtová presnosť.*



(1)

Obrázok 2.14: *Závislosť času voxelizácie od rozlíšenia pre rôzne typy objektov*
Čas je vyjadrený v sekundách. Objekty: (a) pravidelný osemsten, (b) guľa, (c) onion, (d) super-guľa.

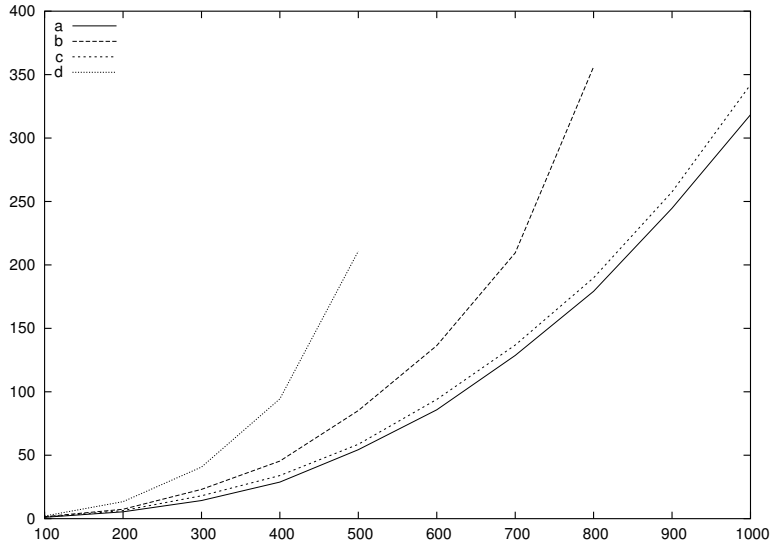
2.4.3 Časová náročnosť voxelizácie

Čas voxelizácie môžeme rozdeliť na dve základné zložky:

1. čas strávený výpočtom implicitnej funkcie
2. čas potrebný na uloženie informácie do mriežky

Prvá zložka zrejme nezávisí od reprezentácie objemu, ale len od náročnosti výpočtu implicitnej funkcie. Na obrázku 2.14 vidíme, že objekty popísané komplikovanejšou funkciou sú voxelizované niekoľkonásobne pomalšie.

Pri jednoduchšie definovaných objektoch sa výraznejšie prejaví aj druhá zložka. RL kompresia má nevýhodu, že k voxelom nemáme priamy prístup — rýchlosť prístupu je závislá od zložitosti povrchu. Na druhej strane môžeme využiť riadkovú kompresiu na rýchle vypĺňanie homogénnych oblastí — celý vonkajší alebo vnútorný riadkový segment sa dá uložiť naraz. Z grafu 2.15 vidno, že druhý vplyv je silnejší, vďaka čomu je voxelizácia pomocou RL kompresie rýchlejšia ako bez nej. Ďalej si môžeme všimnúť, že ukladanie gradientu spôsobí takmer zanedbateľné spomalenie.



Obrázok 2.15: *Závislosť času voxelizácie od rozlíšenia pre komprimovaný a nekomprimovaný objem*
Čas je vyjadrený v sekundách. Typy objemov: (a) komprimovaný bez gradientu, (b) nekomprimovaný bez gradientu, (c) komprimovaný s gradientom, (d) nekomprimovaný bez gradientu. Testovací objekt: guľa.

Kapitola 3

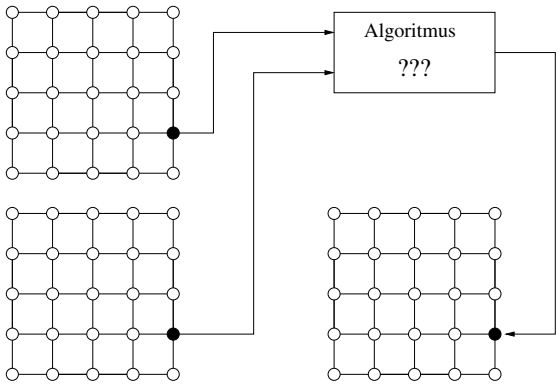
CSG operácie s voxelizovanými modelmi

Ako sme už uviedli v úvodnej kapitole, CSG operácie so vzdialenostnými poľami sa nedajú prevádzať triviálnym spôsobom, ak chceme dostať korektný výsledok. Kameňom úrazu sú hrany, ktoré v zásade nie sú reprezentovateľné a pri CSG operáciách bežne vznikajú. Preto pri snahe vyhnúť sa artefaktom potrebujeme hrany zaobliť.

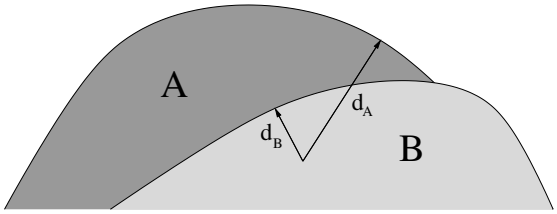
Našim cieľom bolo vyvinúť CSG operácie, ktoré budú hrany vyhladzovať bez potreby rekonštruovať spojité objekty z ich diskrétnej reprezentácie. Ideálne by bolo nájsť algoritmus, ktorý na vstupe dostane dva voxelizované objekty a vytvorí žiadaný výsledok jedným prechodom voxelmi, pričom na určenie hodnoty voxelu v novom objeme sa použije vždy len informácia z dvojice korešpondujúcich voxelov zo vstupných objemov (schéma je na obrázku 3.1).

Nami vytvorené CSG operácie vychádzajú z tejto schémy, aj keď sa jej nedržia úplne striktne. Pri sofistikovanejších metódach totiž potrebujeme malej množine voxelov zo vstupných objemov dodať určitú chýbajúcu informáciu na základe hodnôt voxelov z ich blízkeho okolia. Hlavný princíp pracovať len na úrovni voxelov sa nám však podarilo zachovať.

Každý voxel má definovanú hustotu $d \in \langle 0, 1 \rangle$. Mimo objektu je hustota nulová, vo vnútri je jednotková a na povrchu má hodnotu $0,5$. Voxely v prechodovej oblasti majú naviac zapamätaný normalizovaný gradient $\vec{n}$. Algoritmus vo všeobecnosti pracuje s hodnotami d_a, $\vec{n}_a$, d_b, $\vec{n}_b$ vstupných voxelov a ich analýzou získa výsledné hodnoty d, $\vec{n}$.



Obrázok 3.1: *Schéma ideálnej realizácie CSG operácie na úrovni voxelov*



Obrázok 3.2: *Ilustrácia k minmaxovému kritériu*
A, B sú vstupné objekty. d_A je vzdialenosť od povrchu A a zároveň od $A \cup B$, d_B je vzdialenosť od povrchu B a zároveň od $A \cap B$.

Skôr, ako sa začneme venovať konštrukcii CSG operácií, je dobré si uvedomiť, že medzi zjednotním, prienikom a rozdielom platia nasledovné vzťahy

$$\begin{aligned} A \cap B &= (A^c \cup B^c)^c & \qquad A \cup B &= (A^c \cap B^c)^c \\ A \setminus B &= (A^c \cup B)^c & \qquad A \setminus B &= A \cap B^c, \end{aligned}$$

kde A^c (doplnok k objektu A) získame na voxelovej úrovni jednoducho:

$$\begin{aligned} d^c &= 1 - d \\ \vec{n}^c &= -\vec{n} \end{aligned}$$

To znamená, že všetky operácie môžeme vykonať jednotným spôsobom. Stačí vedieť vytvoriť zjednotenie (respektíve prienik) a ostatné operácie naň prevedieme. Ďalej budeme na ilustráciu používať ľubovoľnú CSG operáciu podľa toho, ktorá bude v danej situácii názornejšia.

3.1 Jednoduchá CSG operácia

Prvú sme implementovali jednoduchú CSG operáciu, ktorá výsledný voxel vytvorí priamo prebratím celej informácie (respektíve jej negáciou) z jedného zo vstupných voxelov. O tom, ktorý z nich to bude, rozhodne „minmaxové“ kritérium:

$$\begin{aligned} d_{A \cup B} &= \max(d_a, d_b) \\ d_{A \cap B} &= \min(d_a, d_b) \\ d_{A \setminus B} &= \min(d_a, 1 - d_b) \end{aligned}$$

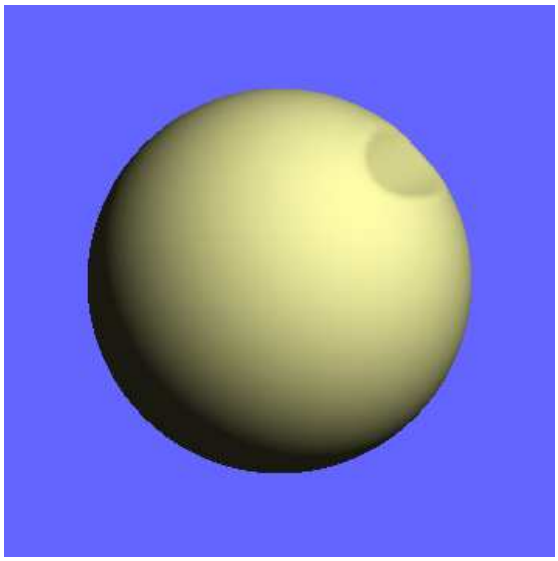
Gradient preberieme z toho istého voxelu ako hustotu, v prípade rozdielu ho niekedy potrebujeme otočiť (ak je výsledná hustota $1 - d_b$).

Minmaxové kritérium už bolo v minulosti viackrát aplikované (napríklad v [15][6][12]), motiváciu k nemu poskytuje obrázok 3.2. V článku [15] bol navrhnutý alternatívny spôsob na výpočet hustoty rozdielu objektov:

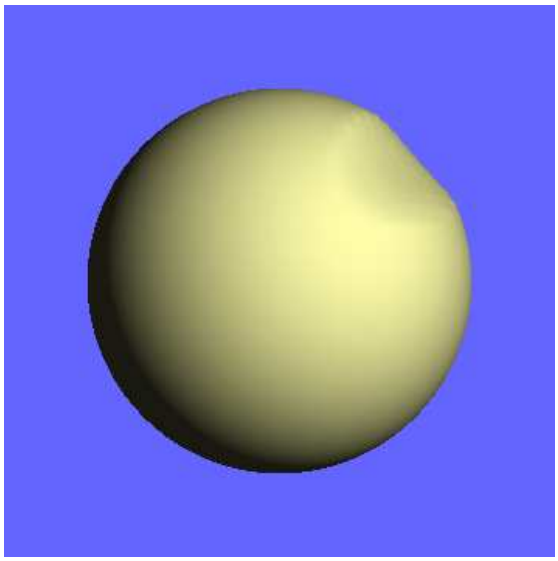
$$d_{A \setminus B} = \max(0, d_a - d_b)$$

Táto realizácia rozdielu vedie k rozšíreniu hrany. Ostré hrany potom vyzerajú lepšie (sú menej zubaté), ale pri tupých hranách dochádza k vizuálnemu zhoršeniu. Na obrázkoch 3.3 a 3.4

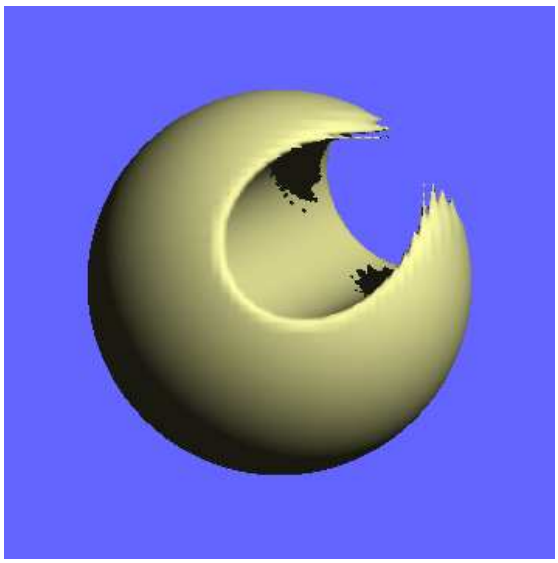
môžeme obe metódy porovnať — ťažko stanoviť, ktorá je lepšia. My sme sa rozhodli kvôli uniformite používať prvú metódu, pretože zachováva platnosť vzťahov medzi zjednotením, prienikom a rozdielom.



(1a)



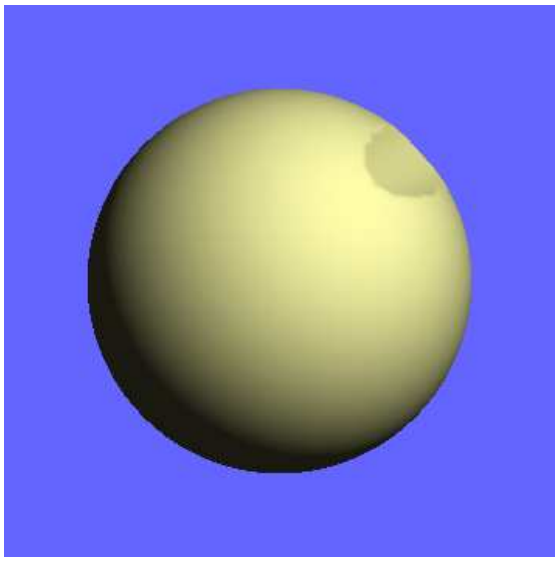
(2a)



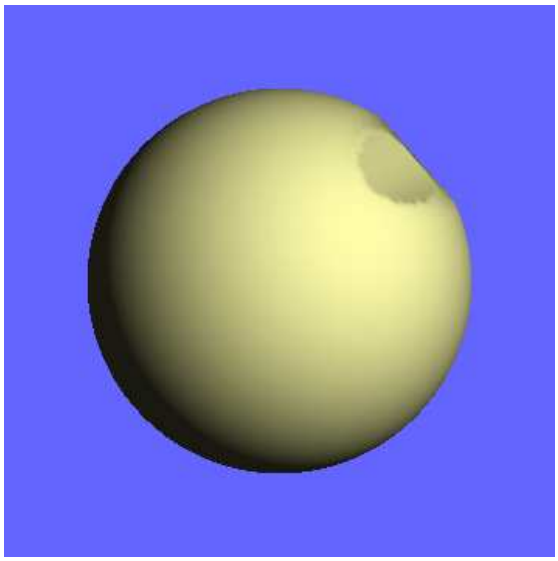
(1b)

(2b)

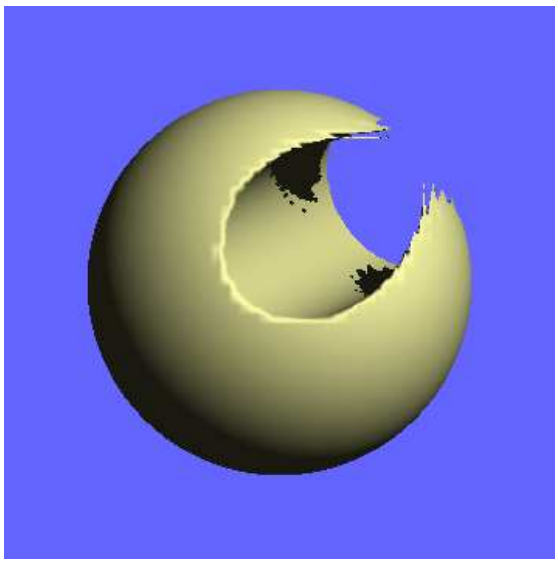
Obrázok 3.3: *Porovnanie dvoch spôsobov realizácie rozdielu objektov pomocou jednoduchého minmaxového kritéria (objem bez gradientu)*
Model: guľa mínus valec. Rôzne výpočty výslednej hustoty:
(1) $d_{A\setminus B} = min(d_a,\ 1 - d_b)$
(2) $d_{A\setminus B} = max(0,\ d_a - d_b)$



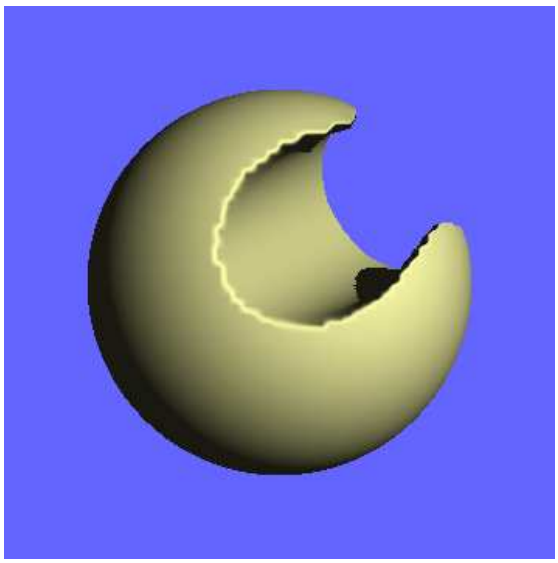
(1a)



(2a)



(1b)



(2b)

Obrázok 3.4: *Porovnanie dvoch spôsobov realizácie rozdielu objektov pomocou jednoduchého minmaxového kritéria (objem s gradientom)*
Model: guľa mínus valec. Rôzne výpočty výslednej hustoty:
(1) $d_{A\setminus B} = min(d_a,\ 1 - d_b)$
(2) $d_{A\setminus B} = max(0,\ d_a - d_b)$

3.2 Vylepšená CSG operácia

Minmaxové kritérium dáva dobré výsledky skoro všade, zlyháva len v blízkosti prieniku daných povrchov. To je práve kritická oblasť, kde vznikajú hrany, ktoré potrebujeme zaobliť tak, aby výsledok bol korektne reprezentovateľný. Obrázok 3.5 ilustruje situáciu v okolí hrany, ktorá vznikne prienikom objektov A, B. Kvôli prehľadnosti sú zobrazené len hranice prechodových oblastí (nazývame ich *vnútorný* a *vonkajší povrch*) — skutočný povrch prechádza stredom medzi nimi. Metódu si popíšeme na rovinnom prípade, ktorý sa ľahko zovšeobecní pre trojrozmerný priestor. Vychádzame z prieniku dvoch polrovín, pričom treba rozlíšiť dve možnosti podľa toho, aký uhol zvierajú:

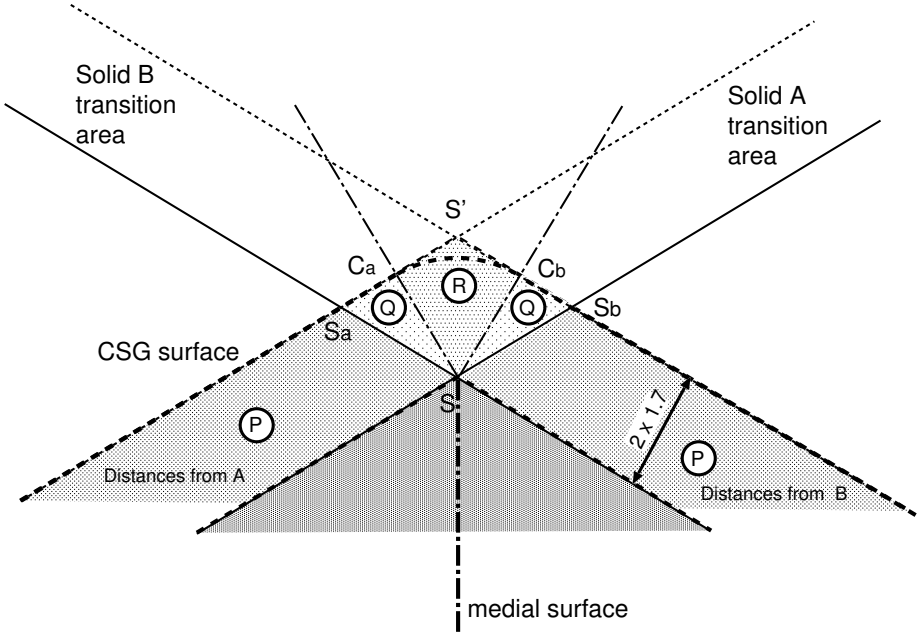
tupý uhol: V oblastiach P, Q získame správny výsledok pomocou minmaxového kritéria. V oblasti R upravíme hustotu a jej gradient tak, aby korešpondovala so vzdialenosťou od bodu S, ktorý je priesečníkom vnútorných povrchov. Vonkajší povrch prieniku vytvorí v oblasti R kružnicový oblúk so stredom S, polomerom $2r$ a koncovými bodmi C_a, C_b, kde r je polomer prechodovej oblasti. Podobne i skutočný povrch bude v R tvorený kružnicovým oblúkom, ale s polomerom r. Týmto presne splníme kritérium otvorenosti a uzavretosti, ktoré v rovinnom prípade znamená, že sme schopní „kotúľať“ kruh s polomerom r po oboch stranách povrchu.

ostrý uhol: Tu je situácia zložitejšia. Potrebovali by sme dať voxelom v oblastiach Q, R informáciu, ktorá by zodpovedala ich vzdialenosti od bodu S (podobne, ako v prípade tupého uhlu). Problém je v tom, že body v Q ležia len v prechodovej oblasti jedného objektu, preto v nich poznáme hustotu a gradient len pre jeden povrch. Na základe hodnôt voxelov nedokážeme odlíšiť oblasti P a Q. Jedine v R môžeme použiť iný postup ako minmaxové kritérium. Aby sme sa čo najviac priblížili ideálnemu riešeniu, voxelom v R priradíme informáciu tak, aby vonkajší povrch vytvoril kružnicový oblúk spájajúci body S_a, S_b. Polomer volíme tak, že sa oblúk v S_a, S_b dotýka vonkajších povrchov objektov A, B. Prirodzene, týmto sa nám nepodarí splniť kritérium otvorenosti a uzavretosti — čím menší uhol povrchy zvierajú, tým väčšiu chybu dostaneme.

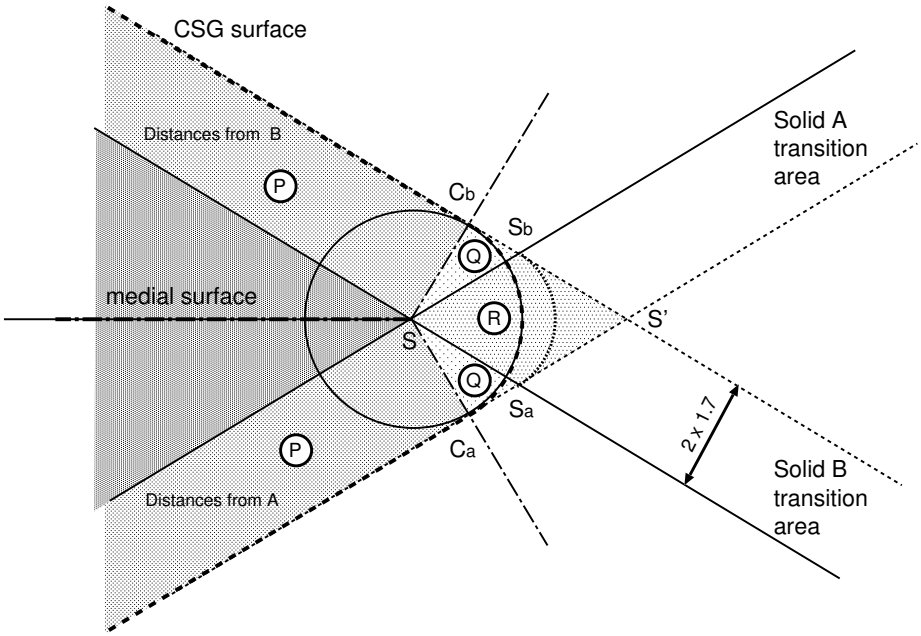
Ktorý z uvedených prípadov nastal, rozpoznáme na základe skalárneho súčinu normalizovaných gradientov.

Týmto sme si vysvetlili filozofiu metódy, teraz si popíšeme konkrétny výpočet hustoty a gradientu v kritickej oblasti. Kľúčovým je riešenie úlohy načrtnutej na obrázku 3.6. Máme daný bod P a lineárne nezávislé vektory $\vec{u}$, $\vec{v}$, ktoré určujú roviny α, β — roviny α, β sú porade kolmé na vektory $\vec{u}$, $\vec{v}$ a od bodu P sú vzdialené $\|\vec{u}\|$, $\|\vec{v}\|$. Ďalej X je ten bod priesečnice rovín, ktorý je najbližšie k bodu P. Našim cieľom je nájsť vektor $\vec{x} = \overrightarrow{PX}$. Využijeme tri fakty:

1. $(\vec{x} - \vec{u}) \perp \vec{u}$
2. $(\vec{x} - \vec{v}) \perp \vec{v}$
3. $\vec{x}$ je lineárnou kombináciou vektorov $\vec{u}$, $\vec{v}$

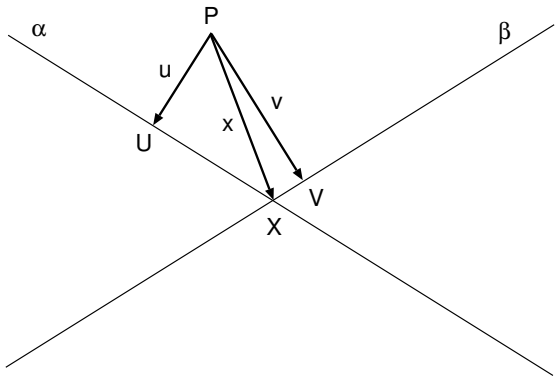


(a)



(b)

Obrázok 3.5: *Schéma prieniku dvoch objektov A, B*
(a) hrana s tupým uhlom , (b) hrana s ostrým uhlom (b)



Obrázok 3.6: *Prienik dvoch rovín*
Roviny α, β *sú dané bodom P a vektormi* $\vec{u}$, $\vec{v}$. *X je najbližší bod priesečnice rovín k bodu P. Vektor* $\vec{x}$ *nezávisí od voľby bodu P, ale len od vektorov* $\vec{u}$, $\vec{v}$.

Na základe týchto poznatkov zostavíme sústavu rovníc a jej riešením získame vzťah

$$\vec{x} = \frac{(\|\vec{u}\|^2 - \vec{u}\vec{v}) \cdot \|\vec{v}\|^2}{\|\vec{u}\|^2\|\vec{v}\|^2 - (\vec{u}\vec{v})^2} \cdot \vec{u} + \frac{(\|\vec{v}\|^2 - \vec{u}\vec{v}) \cdot \|\vec{u}\|^2}{\|\vec{u}\|^2\|\vec{v}\|^2 - (\vec{u}\vec{v})^2} \cdot \vec{v} \qquad (*)$$

Na obrázku 3.7 je naznačená schéma, z ktorej vyplýva výpočet gradientu a hustoty pre voxely ležiace v kritickej oblasti v okolí tupej hrany zjednotenia dvoch objektov. Vo V máme dané hustoty d_a, d_b a normalizované gradienty $\vec{n}_a$, $\vec{n}_b$ vstupných objemov A, B a chceme získať hodnoty d a $\vec{n}$ výsledného zjednotenia. Využijúc, že $\vec{a} = -d_a\vec{n}_a$ a $\vec{b} = -d_b\vec{n}_b$, na základe (*) dostávame:

$$\begin{aligned} \vec{g} &= -K\vec{n}_a - L\vec{n}_b \\ K &= \frac{d_a - d_b\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2} \\ L &= \frac{d_b - d_a\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2} \end{aligned}$$

Pre $K > 0, L > 0$ leží V v oblasti R, preto mu priradíme hustotu a gradient na základe vektora $\vec{g}$:

$$\begin{aligned} d &= \|\vec{g}\| \\ \vec{n} &= -\frac{\vec{g}}{\|\vec{g}\|} \end{aligned}$$

Inak využijeme minmaxové kritérium:

pre $K \leq 0$:

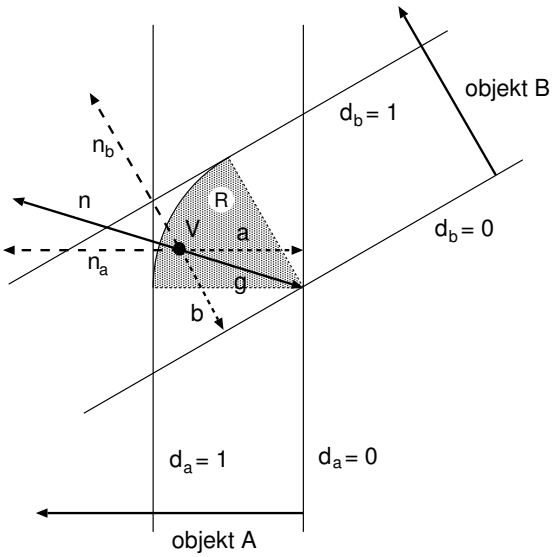
$$\begin{aligned} d &= d_a \\ \vec{n} &= \vec{n}_a \end{aligned}$$

pre $L \leq 0$:

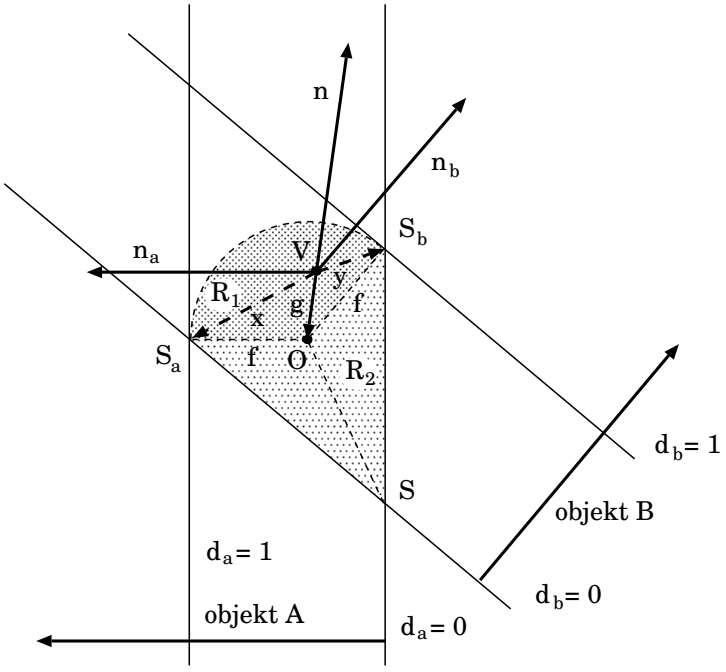
$$\begin{aligned} d &= d_b \\ \vec{n} &= \vec{n}_b \end{aligned}$$

V okolí ostrej hrany je situácia komplikovanejšia (pozri obrázok 3.8). Vektory $\vec{x}$, $\vec{y}$ môžeme vyjadriť na základe (*) nasledovne:

$$\begin{aligned} \vec{x} &= K_x\vec{n}_a - L_x\vec{n}_b & \qquad \vec{y} &= -K_y\vec{n}_a + L_y\vec{n}_b \\ K_x &= \frac{(1-d_a)+d_b\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2} & \qquad L_x &= \frac{d_b+(1-d_a)\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2} \\ K_y &= \frac{d_a+(1-d_b)\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2} & \qquad L_y &= \frac{(1-d_b)+d_a\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2} \end{aligned}$$



Obrázok 3.7: *Kritická oblasť zjednotenia objektov — tupý uhol*
V leží v kritickej oblasti zjednotenia $A \cup B$. Jeho hustotu a gradient odhadneme na základe vektora $\vec{g}$.



Obrázok 3.8: *Kritická oblasť zjednotenia objektov — ostrý uhol*
V leží v kritickej oblasti zjednotenia $A \cup B$. Jeho hustotu odhadneme na základe vektora $\vec{g}$.

Z týchto vzťahov vieme ďalej odvodiť:

$$\begin{aligned} \vec{g} &= -K_y\vec{n_a} - L_x\vec{n_b} \\ f &= K_x + K_y = L_x + L_y = \frac{1}{1-\vec{n_a}\vec{n_b}} \end{aligned}$$

Keďže bod O je od vnútorného povrchu vzdialený o dĺžku f, musí mať hustotu $1-f$. Voxel V je k povrchu o $\|\vec{g}\|$ bližšie, preto mu priradíme hodnotu

$$d = (1-f) + \|\vec{g}\| .$$

Oblasť R_1 je charakterizovaná podmienkou $K_y > 0, L_x > 0$. Voxely mimo nej dostanú hustotu podľa minmaxového kritéria.

Rozhodnúť, aký smer gradientu by mali mať voxely v oblastiach R_1, R_2, nie je triviálne. Ak by sme sa rozhodli pre smer k bodu S, na úsečkách S_aS, S_bS by sa gradient menil nespojite, pretože mimo R_1, R_2 je smer gradientu kolmý na jeden z pôvodných povrchov. Ak by sme v oblasti R_2 ponechali gradient podľa minmaxového kritéria a v R_1 by sme zvolili smer k bodu O, vznikla by nespojitosť na úsečke OS. Preto počítame gradient v oblastiach R_1, R_2 týmto spôsobom:

$$\vec{n} = \frac{d_a\vec{n_a} + d_b\vec{n_b}}{\|d_a\vec{n_a} + d_b\vec{n_b}\|}$$

Tento vzťah je odvodený z prieniku povrchov, ktoré sú na seba kolmé. Takto stanovený gradient nemá v kritickej oblasti presne smer normály na povrch, ale aspoň sa všade mení spojite (okrem bodu S, kde to nevadí). Chyba, ktorá týmto vznikne, je menej závažná, ako nespojitý gradient.

Špeciálne treba ošetriť situácie, keď sú blízko seba rovnobežné povrchy (presnejšie povedané, takmer rovnobežné s určitou toleranciou). Pre rovnako orientované povrchy priamo použijeme minmaxové kritérium. Pre opačne orientované povrchy vychádzame s limitného prípadu ostrého uhla, preto zjednotenie realizujeme na základe hustôt nasledovne:

Ak $d_a + d_b \leq 1$, potom máme:

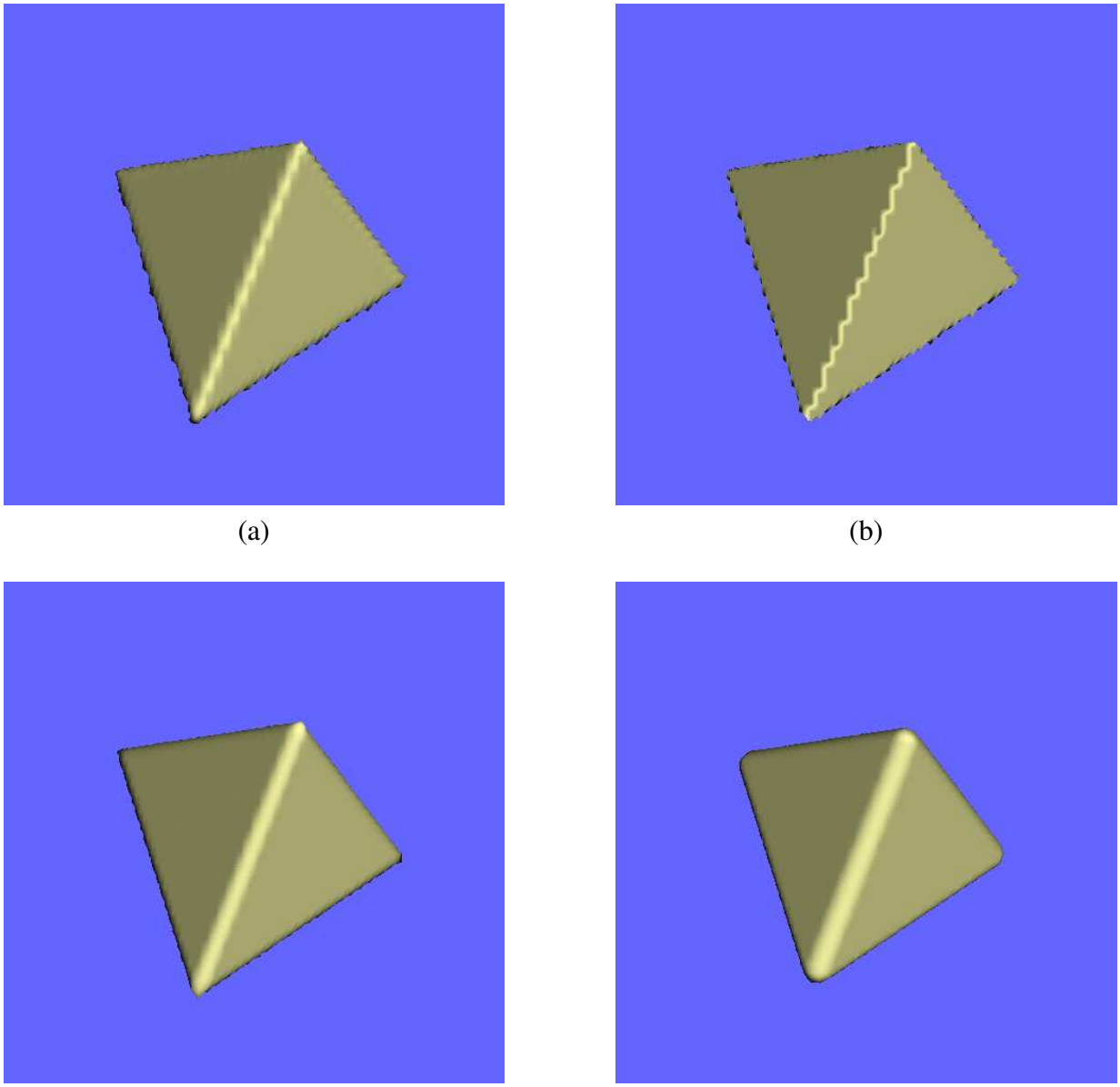
pre $d_a \geq d_b$: $\vec{n} = \vec{n_a}$, $d = d_a$ — pre $d_a < d_b$: $\vec{n} = \vec{n_b}$, $d = d_b$

Ak $d_a + d_b > 1$, potom stanovíme: $d = 1$

Vylepšená CSG operácia korektne vyhladzuje tupé hrany a na ostrých hranách do určitej miery koriguje artefakty. Na obrázku 3.9 vidno pravidelný štvorsten, na ktorom touto metódou dosahujeme viditeľne lepšie výsledky ako jednoduchou CSG operáciou, hoci steny zvierajú ostrý uhol (asi $75,5°$). Na veľmi ostrých hranách (obrázok 3.10(c)) sa však nedostatky prejavia viac, dochádza k podobnému „zúbkateniu“ ako pri jednoduchej metóde. Preto sme pristúpili k návrhu dokonalejšej techniky, ktorú prezentujeme na ďalších stranách.

3.3 Pokročilá CSG operácia

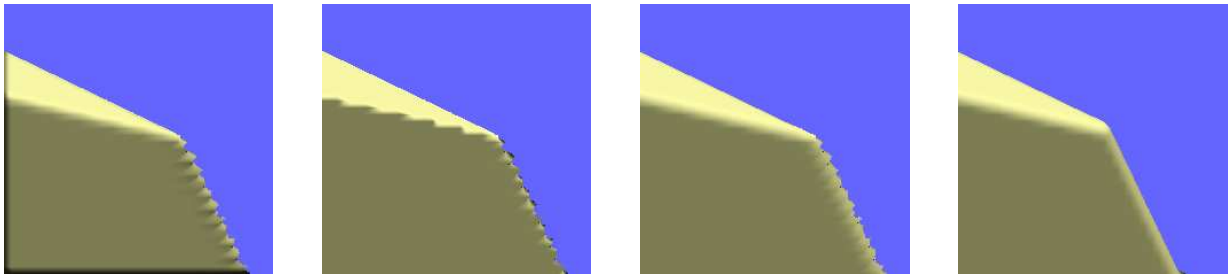
Pokročilá metóda sa snaží odstrániť artefakty aj na ostrých hranách. Hlavným problémom je spomínaná oblasť (Q na obrázku 3.5(b)), v ktorej máme len informáciu o jednom povrchu, ale



(a) (b)

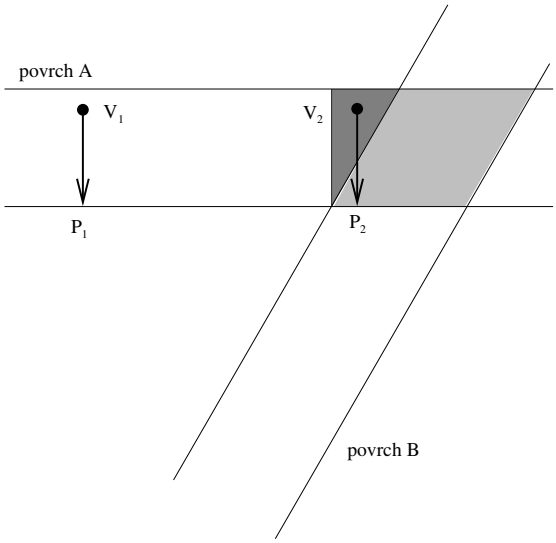
(c) (d)

Obrázok 3.9: *Artefakty na hranách pravidelného štvorstenu pri rôznych výpočtoch CSG operácií Objem bez gradientu: (a) jednoduchá operácia. Objem s gradientom: (b) jednoduchá operácia, (c) vylepšená operácia, (d) pokročilá operácia.*



(a) (b) (c) (d)

Obrázok 3.10: *Artefakty na ostrých hranách pri rôznych výpočtoch CSG operácií*
Objem bez gradientu: (a) jednoduchá operácia. Objem s gradientom: (b) jednoduchá operácia, (c) vylepšená operácia, (d) pokročilá operácia.



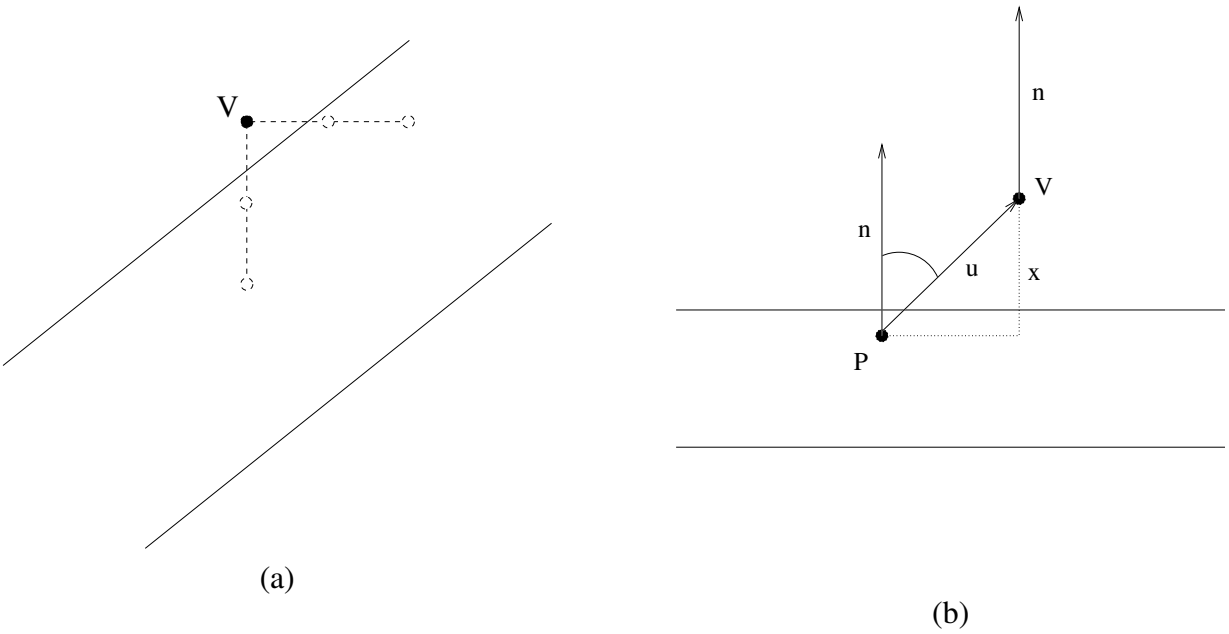
Obrázok 3.11: *Test na rozlíšenie voxelov, ktoré ležia v kritickej oblasti*

na správne zaoblenie hrany potrebujeme aj údaje o druhom povrchu. Keď ich budeme mať, môžeme výpočet určený doteraz len pre tupé hrany rozšíriť na ľubovoľný typ hrany. Za týmto účelom musíme pridať do algoritmu nasledovné kroky:

1. Pre voxely, ktoré ležia len v okolí jedného povrchu, treba rozlíšiť, či patria do oblasti P alebo Q.

2. Voxelom v oblasti Q musíme pred vykonaním CSG operácie doplniť chýbajúcu informáciu o vzťahu k druhému povrchu.

Hoci hustota pamätaná vo voxeloch má len hodnoty z intervalu $\langle 0, 1 \rangle$, pri realizácii CSG operácií pracujeme aj s hodnotami mimo tohto intervalu. Pritom tieto hodnoty vypočítame vždy tesne pred realizáciou CSG operácie s danou dvojicou voxelov, preto túto informáciu nepotrebujeme do voxelov ukladať.



(a) (b)

Obrázok 3.12: *Doplnenie informácie do voxelu v kritickej oblasti (a) pomocou susedných voxelov, (b) pomocou predtým nájdeného bodu P*

Test, ktorým rozpoznávame voxely v oblasti Q, je ilustrovaný obrázkom 3.11. Vo voxeli V máme informáciu o povrchu A, na základe ktorej vieme určiť súradnice bodu P (päta kolmice na hranicu prechodovej oblasti). Ak bod P leží v prechodovej oblasti objektu B, potom voxel V sa nachádza v kritickej oblasti Q, inak je mimo nej.

Pravda, testovať polohu bodu P vzhľadom na povrch B nie je úplne triviálne. P je totiž bod spojitého priestoru, ktorý leží v niektorej mriežkovej bunke. Vo vrcholoch bunky sú voxely, z ktorých niektoré môžu byť vo vnútri a niektoré mimo prechodovej oblasti B. Pomocou tejto neúplnej informácie potrebujeme odhadnúť hustotu v bode P. S týmto faktom sa vysporiadavame nasledovne:

- Ak sú všetky voxely danej bunky mimo prechodovej oblasti B, potom aj bod P je mimo.
- Ak sú všetky voxely danej bunky vo vnútri prechodovej oblasti B, potom aj bod P je vo vnútri.
- Inak skontrolujeme všetkých 26 susedných buniek (susediacich stenou, hranou alebo vrcholom) a z nich si vyberieme tie, ktoré majú všetky voxely v prechodovej oblasti B. Z každej takejto bunky získame trilineárnou interpoláciou odhad hustoty v bode P. Bunkám priradíme váhy na základe ich vzdialeností od bodu P a vytvoríme váhovaný priemer z odhadnutých hustôt. Výslednú hustotu priradíme bodu P a podľa nej určíme, či voxel V je v kritickej oblasti.

V druhom kroku sa snažíme voxelu V dodať hodnoty hustoty a gradientu týkajúce sa povrchu B. Rozlišujeme dva prípady:

(i) Ak aspoň v jednom smere ležia najbližšie dva voxely v prechodovej oblasti, využijeme informáciu z týchto voxelov (obrázok 3.12(a)). Ak napríklad chceme vypočítať hustotu a gradient vo voxeli $V_{i,j,k}$ pomocou známych voxelov $V_{i+1,j,k}$, $V_{i+2,j,k}$, stanovíme:

$$\begin{aligned} d_{i,j,k} &= 2d_{i+1,j,k} - d_{i+2,j,k} \\ \vec{n}_{i,j,k} &= 2\vec{n}_{i+1,j,k} - \vec{n}_{i+2,j,k} \end{aligned}$$

V prípade, že môžeme takýto odhad spraviť pre dvojice voxelov vo viacerých smeroch, výsledok vytvoríme spriemerovaním týchto odhadov.

(ii) Keď sa v okolí V nenachádza žiadna potrebná dvojica voxelov, použijeme informácie zistené v bode P (obrázok 3.12(b)). Gradient $\vec{n}$ bodu P určíme rovnakým postupom ako jeho hustotu. Na základe vektorov $\vec{n}$, $\vec{u} = \overrightarrow{PV}$ vypočítame rozdiel hustôt x v bodoch V, P. Voxel V potom získa hodnoty

$$\begin{aligned} d_V &= d_P + x \\ \vec{n}_V &= \vec{n}_P \, . \end{aligned}$$

Opäť treba osobitne riešiť výskyt rovnobežných povrchov. Pri zjednotení použijeme v prípade rovnako orientovaných povrchov minmaxové kritérium. Ak sú blízko seba opačne orientované povrchy, medzeru medzi nimi nie sme schopní reprezentovať, preto ju vyplníme vnútornou hustotou. Analogicky postupujeme pri realizácii prieniku a rozdielu.

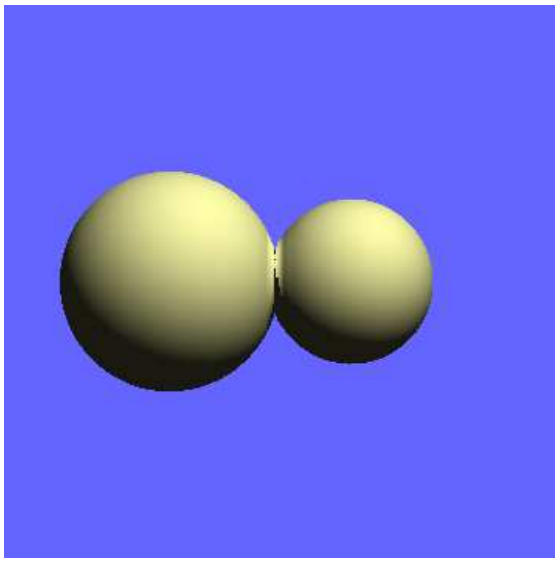
Ako vidno na viacerých obrázkoch (3.9, 3.10, 3.14, 3.16), touto metódou sa nám podarilo vyriešiť problém prakticky všetkých hrán. V prípade ostrých hrán to síce vedie k ich viditeľnému zaobleniu (obrázky 3.14, 3.16)), čo je však v súlade s kritériom otvorenosti a uzavretosti (časť 1.4.2, [12]). Pri zjednotení dvoch takmer sa dotýkajúcich objektov vzniká trochu iný typ artefaktov ako pri použití predošlých metód (obrázok 3.13). Problém je v tom, že úzka medzera medzi objektami nie je reprezentovateľná a pokročilá CSG operácia ju zaplní tak, aby bol medzi povrchmi hladký prechod.

Hoci pri popise metódy sme predpokladali voxely so zapamätaným gradientom, túto techniku je možné použiť aj pre bezgradientové voxely a v prípade potreby gradient dopočítať (stredovými diferenciami). Ako však vidno na obrázku 3.15, týmto spôsobom získame na ostrých hranách horšie výsledky, ako pri použití gradientového voxelu. To potvrdzuje náš predpoklad, že ukladať si gradient je výhodné napriek zvýšeným požiadavkám na pamäť.

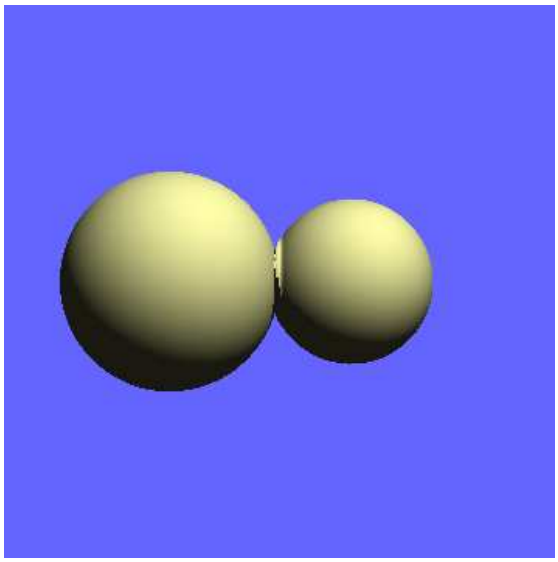
Z pozorovania vyplýva, že pokročilá CSG operácia dosahuje veľmi uspokojivé výsledky. Okrem zvláštnych prípadov, kde sa stále objavujú artefakty (obrázok 3.17). S týmto problémom sme sa pokúsili vysporiadať pomocou špeciálnej CSG operácie.

3.4 Špeciálna CSG operácia

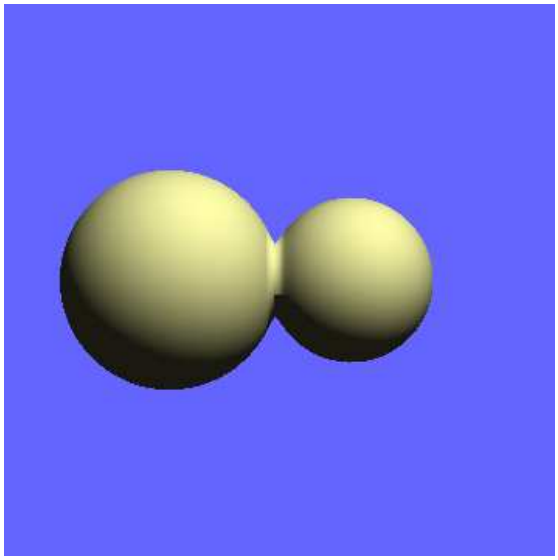
Špeciálna CSG operácia vychádza s pokročilej CSG operácie, pričom sa snaží odhaľovať regióny, kde by mohli nastať problémy. Pokročilá CSG operácia pracuje presne len v situáciách, keď sa pretínajú lokálne rovinné povrchy. Čím viac sú povrchy zakrivené, tým k väčšej chybe dochádza. Vizuálne sa táto chyba začne prejavovať, ak sa zakrivené povrchy stretajú pod



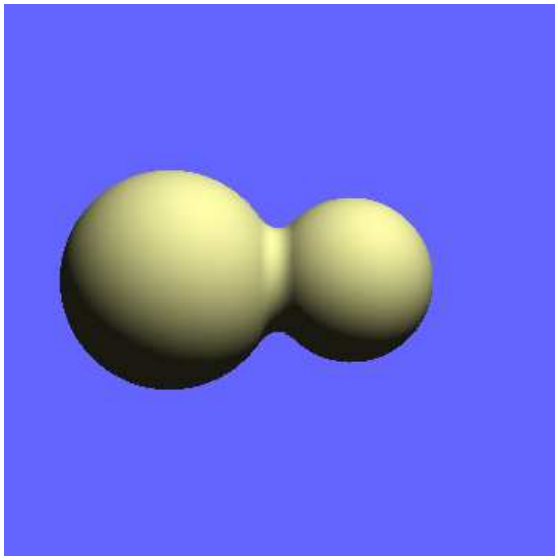
(a)



(b)

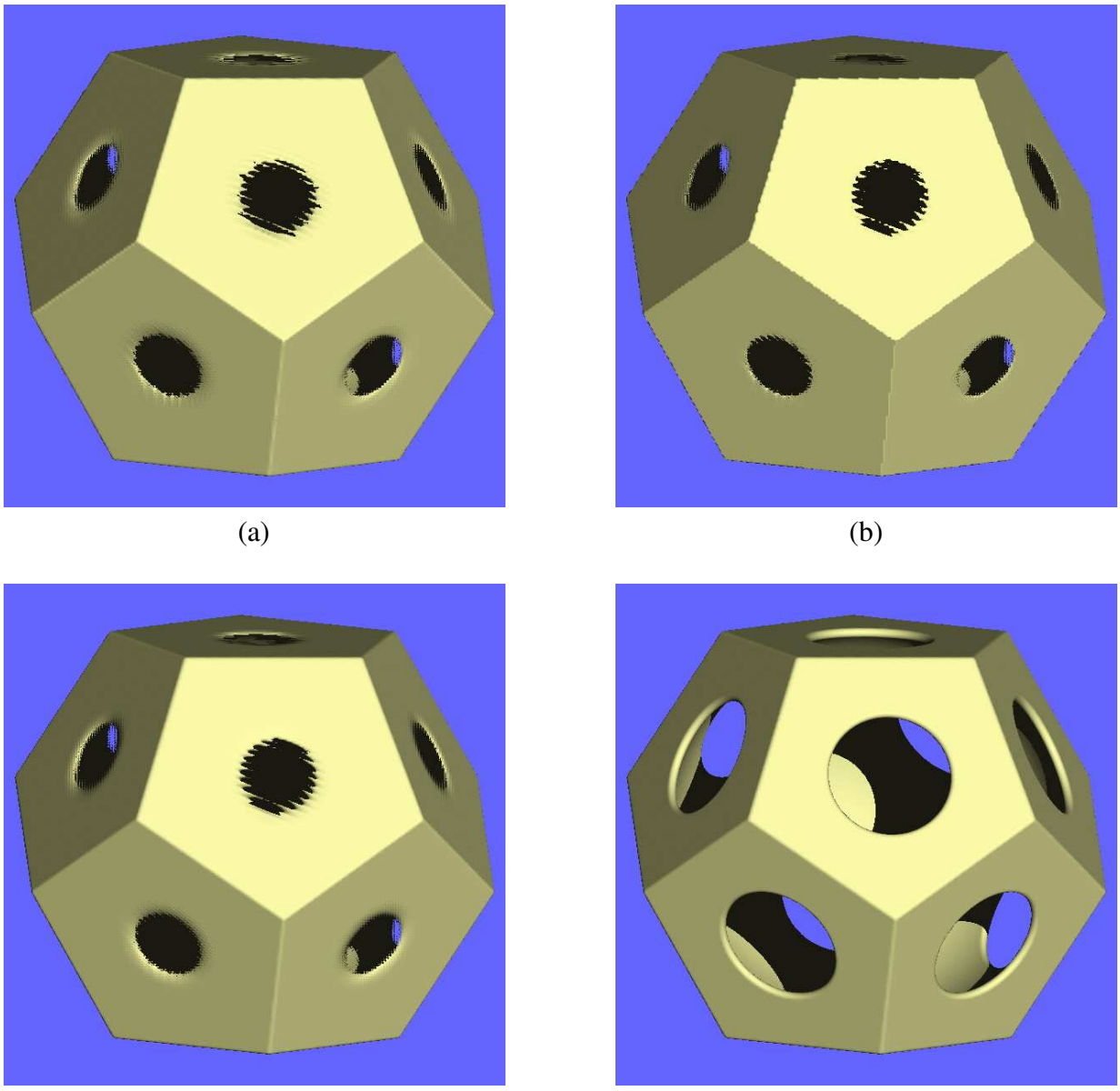


(c)



(d)

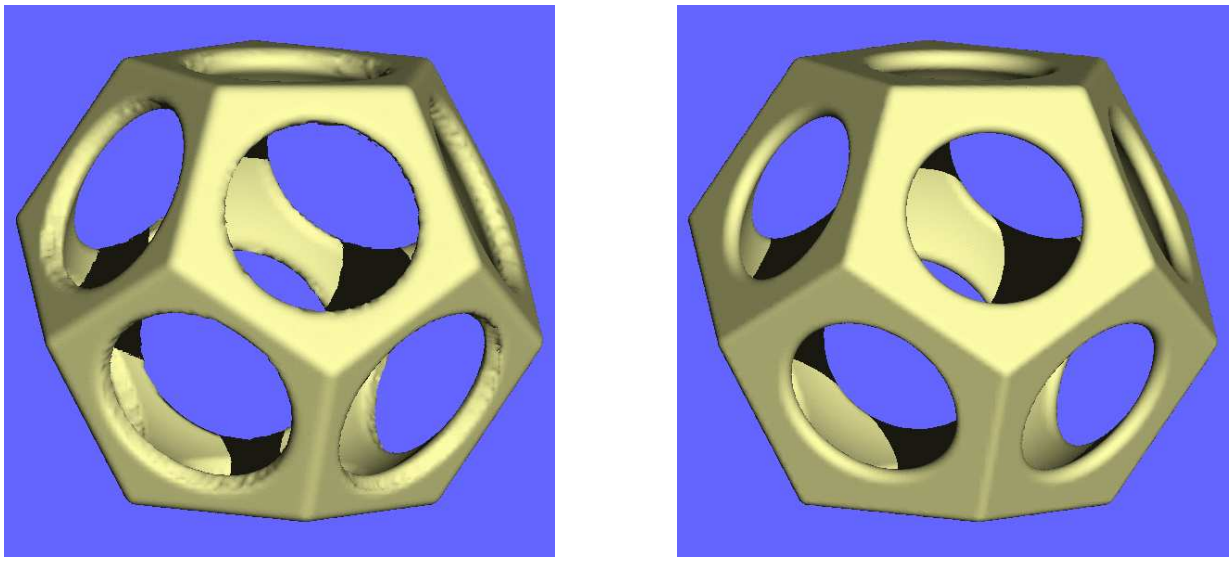
Obrázok 3.13: *Artefakty v okolí dvoch dotýkajúcich sa povrchov pri rôznych výpočtoch CSG operácií*
Objem bez gradientu: (a) jednoduchá operácia. Objem s gradientom: (b) jednoduchá operácia, (c) vylepšená operácia, (d) pokročilá operácia.



(a) (b)

(c) (d)

Obrázok 3.14: *Artefakty na hranách pri rôznych výpočtoch CSG operácií (model: pravidelný 12-sten mínus guľa)*
Objem bez gradientu: (a) jednoduchá operácia. Objem s gradientom: (b) jednoduchá operácia, (c) vylepšená operácia, (d) pokročilá operácia.



(a) (b)

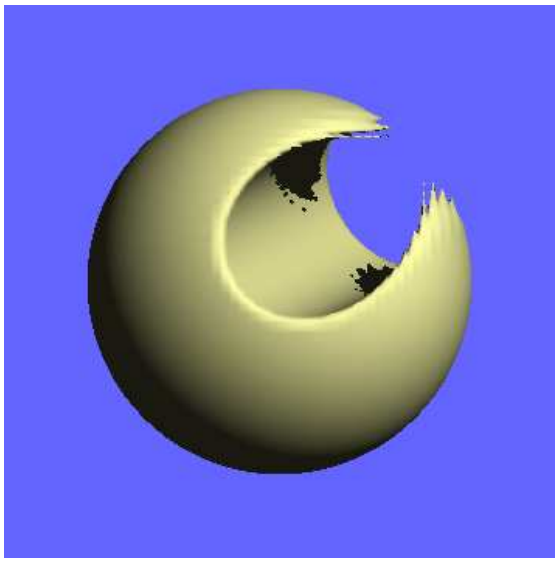
Obrázok 3.15: *Porovnanie pokročilej CSG operácie pre rôzne typy voxelov (a) voxel bez gradientu, (b) voxel s gradientom. Model: pravidelný 12-sten mínus guľa.*

veľmi malým uhlom. Vtedy je totiž výpočet priesečníku kriviek (v priestore povrchov) značne nepresný. Špeciálna CSG operácia sa tento problém snaží riešiť nasledovne:

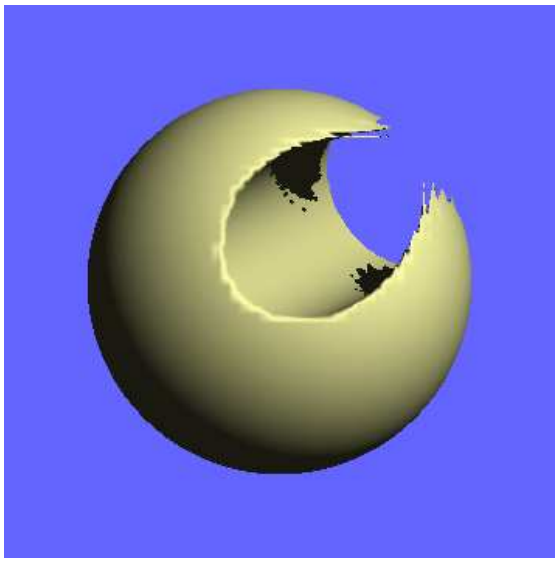
1. Počas vykonávania pokročilej CSG operácie si označujeme voxely, ktoré vznikli kombináciou vstupných voxelov nesúcich „takmer“ opačné gradienty. Experimentálne sme ich stanovili ako tie, ktorých uhol sa od priameho líši menej ako o 7°. Tieto voxely sú nazvané *nebezpečné*.

2. Po uskutočnení CSG operácie kontrolujeme nebezpečné voxely. Ich hodnoty porovnávame so susednými (iba s bezpečnými) voxelmi a ak sa od nich výrazne líšia, upravíme ich tak, aby boli v súlade s „bezpečným okolím“. Podrobnosti sú dosť zložité vzhľadom na rôzne možné situácie, preto ich nebudeme presne popisovať.

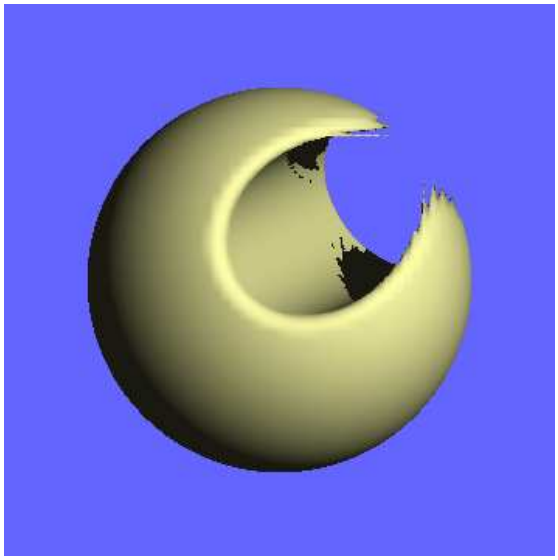
Týmto sa nám podarilo dosiahnuť isté vizuálne zlepšenie, aj keď je zrejmé, že nedostaneme úplne korektný výsledok (pozri obrázok 3.17). Narážame tu totiž na principiálny problém, ktorý je ilustrovaný obrázkom 3.18. Máme dva objekty A, B. Pokým sú od seba dosť ďaleko, ich povrchy sú oddelené. Keď ich navzájom priblížime tak, že sa budú výrazne pretínať, ich povrchy sa spoja. Je zrejmé, že medzi týmito dvoma polohami nemôže existovať spojitý prechod. Preto bude pri ľubovoľnom rozlíšení existovať kritická vzdialenosť, kedy bude zjednotenie týchto dvoch objektov nereprezentovateľné. Pritom tento problém nastane pri približovaní ľubovoľných dvoch povrchov. Našťastie kritická je len vzdialenosť objektov vo veľmi úzkom pásme hodnôt, mimo tohto pásma sa CSG operácia realizuje korektne.



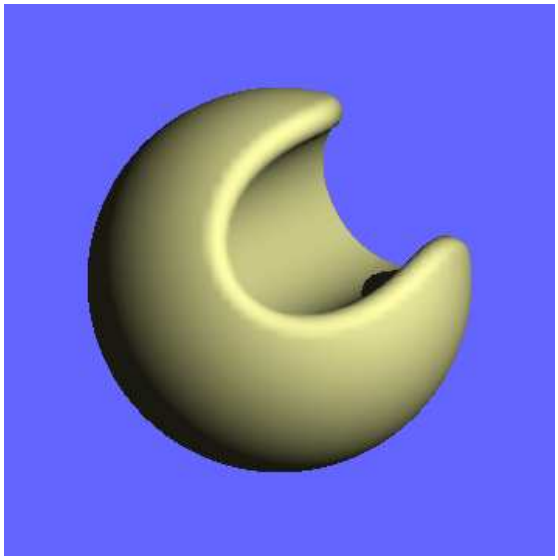
(a)



(b)

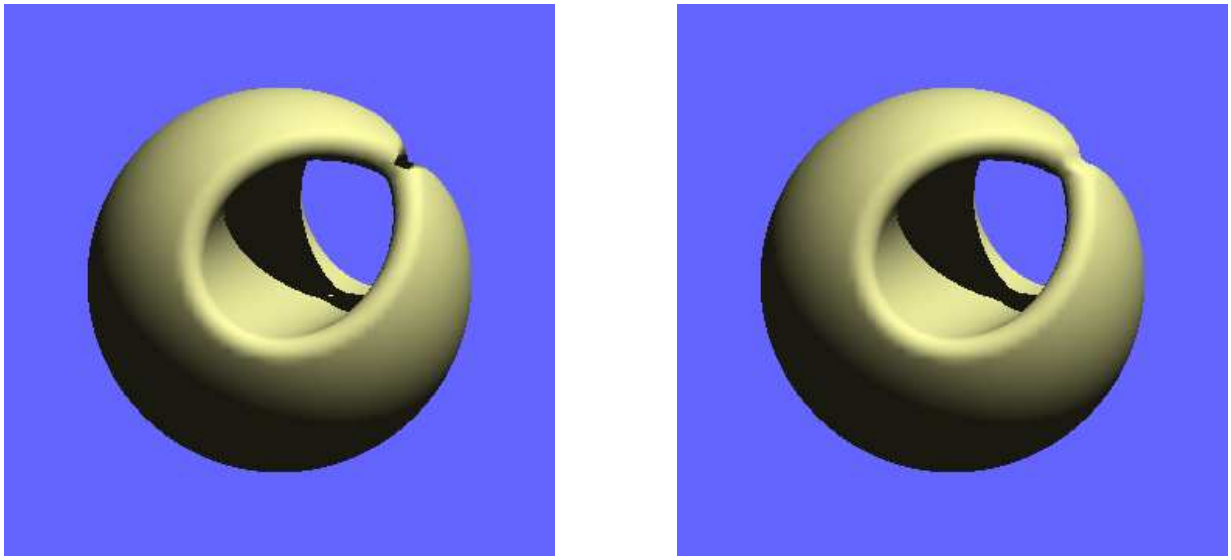


(c)



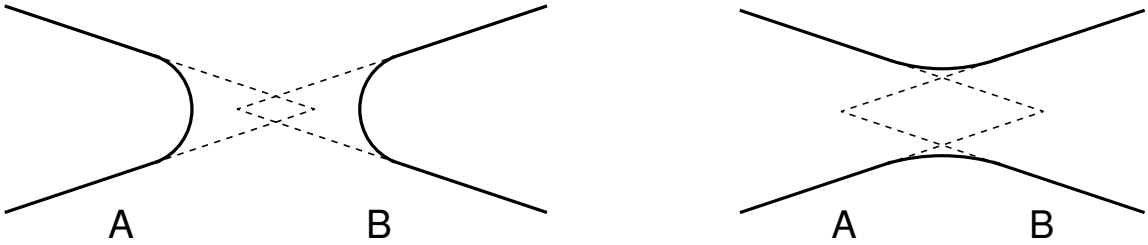
(d)

Obrázok 3.16: *Artefakty na hranách pri rôznych výpočtoch CSG operácií (model: guľa mínus valec)*
Objem bez gradientu: (a) jednoduchá operácia. Objem s gradientom: (b) jednoduchá operácia, (c) vylepšená operácia, (d) pokročilá operácia.



(a) (b)

Obrázok 3.17: *Problémy pokročilej CSG operácie (model: guľa mínus valec)*
Chyby vznikajú pri povrchoch, ktoré sa takmer dotýkajú. Typy CSG operácií: (a) pokročilá, (b) špeciálna.



(a) (b)

Obrázok 3.18: *Problém približujúcich sa objektov*
Snažíme sa reprezentovať zjednotenie dvoch objektov A, B. Medzi polohami (a), (b) neexistuje spojitý prechod, preto v kritickej vzdialenosti vznikajú artefakty.

Kapitola 4

Implementácia

4.1 Knižnica vxt

Východiskom pre túto diplomovú prácu bola existujúca knižnica `vxt`, napísaná v jazyku `C++`. Jej jadrom sú dve hierarchie tried odvodených od nadtried:

`vxtObject3D` Reprezentuje rôzne typy objektov a s nimi spojené voxelizačné techniky.

`vxtGrid3D` Reprezentuje rôzne druhy priestorových mriežok.

Voxelizácia prebieha tak, že objekt zavolá metódu `Voxelize()` s mriežkou ako parametrom. Trieda `vxtObject3D` sa delí na dve podtriedy:

`vxtPrimitive` Reprezentuje jednoduché objekty, ktoré slúžia ako základné jednotky.

`vxtAggregate` Reprezentuje zložené objekty vytvorené z jednoduchých pomocou CSG operácií.

CSG operácie sa realizujú na dvoch úrovniach:

1. Objekt zavolá metódu `Voxelize()` s udaním mriežky a typu CSG operácie a do mriežky sa zapíše informácia vytvorená kombináciou pôvodnej hodnoty v mriežke s novou hodnotou získanou voxelizáciou objektu.

2. Mriežka M zavolá metódu `Merge()` s udaním druhej mriežky N a typu CSG operácie. Výsledok sa zapíše do M.

Trieda `vxtGrid3D` má podtriedy `vxtVolume<T>`, kde `T` je šablónový parameter určujúci presnosť, s akou je reprezentovaná hustota.

Podrobnejší popis knižnice sa nachádza v článku [15].

4.2 Knižnica vxtRL

Nová knižnica `vxtRL` vznikla z pôvodnej `vxt` vykonaním nasledovných úprav:

- Hierarchia pod triedou `vxtGrid3D` bola zásadne pretvorená tak, aby umožňovala jednotnú reprezentáciu komprimovaných aj nekomprimovaných objemov s rôznymi typmi voxelov.
- Pre triedu `vxtImplicitSolid` (podtrieda `vxtPrimitive`) bola vytvorená alternatívna voxelizačná metóda (využívajúca zametacie roviny).
- Pod `vxtImplicitSolid` bola pridaná podtrieda `SolidCylinder` na reprezentáciu telesa ohraničeného rotačnou valcovou plochou.
- V triede `vxtAggregate` bola upravená voxelizačná metóda, aby vyhovovala aj realizácii zložitejších CSG operácií.

Základ hierarchie na reprezentáciu objemov tvoria nasledovné triedy:

`vxtRLVolume<T>`

- Slúži na reprezentáciu komprimovaného objemu pomocou modifikovanej metódy run-length encoding (časť 2.1).
- Pracuje s abstraktným voxelom (šablónový parameter `T`). Ten potrebuje poznať len operácie ==, != a mať preddefinovanú štandardnú hodnotu, ktorá je priradená vonkajším a vnútorným segmentom.
- Umožňuje prístup k dátam po voxeloch, blokoch, riadkoch alebo vrstvách.
- Využíva pomocnú triedu `vxtRLRow<T>`.

`vxtRLRow<T>`

- Slúži na zapamätanie komprimovaného riadku.
- `T` je typ voxelu.
- Obsahuje metódy na konštrukciu, aktualizáciu a rušenie riadku.
- Prístup k dátam je na úrovni voxelov, segmentov, alebo celého riadku.
- Pre testovacie účely sú implementované zisťovacie metódy o veľkosti obsadenej pamäte a o štruktúre riadku.

`vxtVoxel<T>`

- Trieda na reprezentáciu všeobecného voxelu (časť 2.2). Dedia z nej tri podtriedy:

  `vxtPlainVoxel<T>` Pamätá si len hustotu.

  `vxtGradVoxel<T,G>` Pamätá si hustotu aj gradient (trojzložkovo).

vxtSphGradVoxel<T,G> Pamätá si hustotu a gradient pomocou sférických súradníc.

`T`, `G` sú dátové typy použité na reprezentovanie hustoty a gradientu. Môžu nadobúdať hodnoty:

f3dUChar 1 bajt

f3dUInt16 2 bajty

f3dFloat 4 bajty

- Trieda obsahuje metódy na čítanie a zapisovanie z/do voxelov.

vxtGrid3D

- Abstraktná trieda na reprezentáciu mriežky.
- dedí z nej trieda `vxtVolume<V,T,G>`.

vxtVolume<V,T,G>

- Trieda slúži na reprezentáciu ľubovoľného objemu.
- Šablónové parametre `V`, `T`, `G` sú porade pre typ voxelu, hustoty a gradientu.
- Pracuje s hustotou z intervalu $\langle 0, 1\rangle$ a gradientom z $\langle -1, 1\rangle^3$.
- Metódy na čítanie a zapisovanie z/do voxelov pracujú na tejto úrovni (bez ohľadu na typ kompresie, typ voxelu a presnosť dát).
- Obsahuje metódy na interpoláciu uložených diskrétnych dát pre potreby vizualizácie.
- Implementované sú jednoduché metódy na zobrazovanie objemu a tiež na jeho ukladanie a načítanie do/zo súboru (vo formáte `f3d`).
- Využíva triedu `vxtAnyVolume<V,T,G>`.

vxtAnyVolume<V,T,G>

- Trieda tvorí rozhranie medzi `vxtVolume<V,T,G>` a svojimi podtriedami `vxtUncompVolume<V,T,G>` a `vxtCompRLVolume<V,T,G>`.
- Zabezpečuje konverziu hustoty (gradientu) z $\langle 0, 1\rangle$ ($\langle -1, 1\rangle^3$) do príslušných rozsahov podľa typov `T`, `G`.
- Implementuje CSG operácie na úrovni voxelov (kapitola 3).

vxtUncompVolume<V,T,G>

- Pamätá si celý objem bez kompresie. Využíva pritom triedu `f3dgrid` z knižnice `f3d`.
- Metódy na čítanie a zapisovanie voxelov realizuje pomocou rozkladu voxelu na jednotlivé zložky.
- Obsahuje metódy na ukladanie a čítanie dát do/zo súboru.

vxtCompRLVolume<V,T,G>

- Na zapamätanie objemu používa triedu `vxtRLVolume< V<T,G> >`
- Na ukladanie a čítanie dát do/zo súboru využíva metódy knižnice `f3d`.

Časť programu, ktorý využíva knižnicu `vxtRL`, môže vyzerať napríklad takto:

```
1. vxtGrid3D *volume = createGrid2(200, 200, 200,
   VXT_RL_COMPRESS, VXT_SPH_G_VOXEL, f3dUInt16Type,
   f3dUInt16Type);

2. SolidSphere sphere(0.4);

3. sphere.Voxelize(*volume, VXT_WRITE);

4. volume->viewX("image", 3, Vector3D(1,1,1));

5. volume->save("sphere");
```

V prvom kroku vytvoríme komprimovaný objem `volume` s rozmermi $200 \times 200 \times 200$, ktorý bude pracovať s voxelom s dvojzložkovým gradientom a na reprezentáciu hustoty a jednotlivých zložiek gradientu použije dva bajty. V druhom kroku definujeme objekt guľa s relatívnym polomerom $0,4$ (v mriežke to bude $0,4 \cdot 100$ VU) so stredom v strede scény. V ďalšom kroku guľu voxelizujeme do pripraveného objemu. Potom vytvoríme obrázok `image.ppm`, ktorý vznikne projekciou v smere osi x, pričom scéna bude osvetlená zo smeru $(1, 1, 1)$. Nakoniec objem uložíme do súborov `sphere.f3d` (hustota) a `sphere_G.f3d` (gradient).

Ako vidno z príkladu, manipulácia s mriežkami všetkých druhov je jednotná, definícia konkrétneho typu sa vykoná pomocou parametrov funkcie `createGrid2(...)`. Je možný aj altenatívny prístup:

```
vxtVolume<vxtSphGradVoxel, f3dUInt16, f3dUInt16> volume(200, 200,
200, VXT_RL_COMPRESS);
```

Dosiahneme rovnaký výsledok, ako v prvom kroku, len `volume` bude priamo premenná typu trieda a nie smerník na triedu.

Kapitola 5

Záver

V predloženej diplomovej práci sme navrhli nový spôsob, ako objemovo reprezentovať voxelizované geometrické objekty. Základnou myšlienkou je pridať k limitovaným vzdialenostným poliam normalizovaný gradient a využiť ho na presnejšiu rekonštrukciu objektov a na realizáciu sofistikovanejších CSG operácií. Z analýz vyplýva, že pre hustotu a jednotlivé zložky gradientu je najvhodnejšie používať dvojbajtové premenné. Istú časť pamäte môžeme pre gradient ušetriť jeho reprezentáciou pomocou dvoch hodnôt v sférických súradniciach. S cieľom zredukovať pamäťové nároky sme navrhli modifikovanú RL kompresiu, ktorá pre určitú triedu scén dosahuje významný kompresný pomer. Ukázali sme, že táto kompresia urýchľuje aj samotnú voxelizáciu.

Ďalej sme navrhli a implementovali novú metódu realizácie CSG operácií medzi voxelizovanými objektami na úrovni voxelov bez potreby rekonštruovať ich analytický popis. Táto metóda je v súlade s kritériom reprezentovateľnosti voxelizovaných objektov, vďaka čomu odstraňuje nežiaduce artefakty na rozdiel od jednoduchších metód, ktoré boli používané doposiaľ.

Implementácia týchto algoritmov je súčasťou knižnice `vxtRL`, ktorá vznikla rozšírením balíka `vxt`. V práci sa dá pokračovať rôznymi smermi. RL kompresiu zatiaľ nie je možné používať pre farebné dáta, preto by ju bolo vhodné za týmto účelom zovšeobecniť. Súčasná knižnica obsahuje len jednoduché techniky na zobrazovanie uložených dát, tieto by mohli byť rozšírené o komplexnejšie algoritmy.

Literatúra

[1] A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *IEEE Computer*, 26(7):51–64, July 1993.

[2] R. Yagel, D. Cohen, and A. Kaufman. Discrete ray tracing. *IEEE Computer Graphics and Applications*, 12(5):19–28, September 1992.

[3] S.W. Wang and A. Kaufman. Volume sampled voxelization of geometric primitives. In *Visualization '93*, pages 78–84, San Jose, CA, October 1993.

[4] M. Šrámek and A. Kaufman. Object voxelization by filtering. In *IEEE Symposium on Volume Visualization*, pages 111–118, 1998.

[5] S.F.F. Gibson. Using distance maps for accurate surface reconstruction in sampled volumes. In *IEEE Symposium on Volume Visualization*, pages 23–30, 1998.

[6] S. F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 249–254. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[7] R. N. Perry S. F. Frisken. Kizamu: A system for sculpting digital images. In Eugene Fiume, editor, *Siggraph 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 47–56. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2001.

[8] U. Tiede, K.-H. Höhne, M. Bomans, A. Pommert, M. Riemer, and G. Wiebecke. Investigation of medical 3D-rendering algorithms. *IEEE Computer Graphics and Applications*, 10(3):41–53, 1990.

[9] M. E. Goss. An adjustable gradient filter for volume visualization image enhancement. In *Proceedings of Graphics Interface '94*, pages 67–74, May 1994.

[10] J. A. Bærentzen, M. Šrámek, and N. J. Christensen. A morphological approach to voxelization of solids. In *The 8-th International Conference in Central Europe on Computer Graphics, Visualization and Digital Interactive Media 2000*, pages 44–51, Pilsen, Czech republic, 2000.

[11] M. Šrámek, L. I. Dimitrov, and J. A. Baerentzen. Correction of voxelization artifacts by revoxelization. In Klaus Mueller and Arie Kaufman, editors, *Volume Graphics'01, Proceedings of the Joint IEEE TVCG and Eurographics Workshop*, pages 265–275, Stony Brook, NY, June 21–22, 2001.

[12] J. A. Baerentzen and N. J. Chistensen. A technique for volumentric CSG based on morphology. In Klaus Mueller, editor, *Volume Graphics'01*, pages 71–79, Stony Brook, NY, June 2002.

[13] C. Montani and R. Scopigno. Rendering volumetric data using sticks representation scheme. In *Proceedings of the 1990 workshop on Volume visualization*, pages 87–93. ACM Press, 1990.

[14] N. Shareef and R. Yagel. Rapid Previewing via Volume-based Solid Modeling. In *The Third Symposium on Solid Modeling and Applications, SOLID MODELING '95*, pages 281–292, Salt Lake City, Utah, May 1995.

[15] M. Šrámek and A. Kaufman. `vxt`: a c++ class library for object voxelization. In Min Chen, Arie E. Kaufman, and Roni Yagel, editors, *Volume Graphics*, pages 119–134. Springer Verlag, London, 2000.