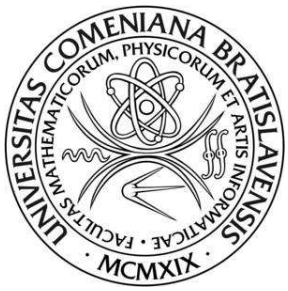
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Univerzita Komenského, Bratislava

Katedra algebry, geometrie a didaktiky matematiky



Aproximácia implicitne definovaných plôch v E^3

DIPLOMOVÁ PRÁCA

Bratislava, 2009 Leonóra Kočišová

Aproximácia implicitne definovaných plôch v E^3

DIPLOMOVÁ PRÁCA

Leonóra Kočišová

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

UNIVERZITA KOMENSKÉHO, BRATISLAVA

Katedra algebry, geometrie a didaktiky matematiky

matematika - počítačová grafika

RNDr. Pavel Chalmovianský, PhD.

Bratislava, 2009

Čestné vyhlásenie:
Čestne vyhlasujem, že diplomovú prácu som vypracovala samostatne pod odborným vedením školiteľa s použitím uvedenej literatúry.

. . . . . .

Bratislava, apríl 2009 Leonóra Kočišová

Abstrakt

KOČIŠOVÁ Leonóra: *Aproximácia implicitne definovaných plôch* [diplomová práca]. Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky. Katedra algebry, geometrie a didaktiky matematiky. Školiteľ: RNDr. Pavel Chalmovianský, PhD., Bratislava. FMFI UK, 2009, 50 strán.

Táto práca sa zaoberá implicitne definovaných plochami od zadania až po vizualizaciu. Popisujeme rôzne spôsoby získania bodov implicitne definovaných plôch pomocou metód numerickej matematiky na výpočet polynómu. Navrhli sme a implementovali algoritmus na načítavanie, a spracovanie vzorcov implicitných funkcií, nájdenie obálky a vizualizáciu implicitných plôch.

Kľúčové slová: numerické metódy, obálka (bouding box), editor vzorcov, algoritmus na trianguláciu.

Obsah

Zoznam obrázkov

1 Úvod

Technické modelovanie je prístup, pri ktorom na základe vlastností konkrétneho objektu tento objekt vymodelujeme. Objekty popisujeme matematickými rovnicami. Z matematickej analýzy poznáme 2 typy rovníc: explicitné a implicitné.

V súčasnosti sa využíva najmä explicitná forma, konkrétne parametrická reprezentácia, pretože objekty popísané parametricky vieme najjednoduchšie vymodelovať. Problém je v tom, že nie všetky objekty reálneho sveta sa dajú parametricky opísať. Preto na ich opis používame implicitnú formu. Nepoznáme však univerzálny algoritmus, ktorý by objekt opísaný implicitne vedel vymodelovať. Preto používame rôzne aproximácie.

V tejto práci voľne nadväzujeme na diplomovú prácu Ondreja Jaborníka, ktorý sa zaoberal problematikou triangulácie implicitne definovaných plôch, t.j. generovaním trojuholníkovej siete povrchu a implicitne definovaného objektu. Popisuje rôzne spôsoby triangulácie, výpočet normály a detekcie ostrých hrán. Používa pritom známe metódy na prehľadávanie a generovanie povrchu a objemu daného objektu, konkrétne metódou *Marching Cubs* generuje objem objektu a následne metódou *Marching Triangules* vytvára povrch objektu. Táto triangulácia je univerzálne použiteľná aproximácia povrchu daného objektu, nemusí však byť najlepšia, dokonca sa môže zacyklit'.
Naším cieľom je spravit' program na zadávanie a editáciu implicitných funkcií, keďže predchádzajúce práce využívajú vkompilované funkcie. Chceme v programe umožnit' užívateľovi zadávat' a vizualizovat' implicitne zadané objekty s čo najmenším množstvom parametrov, a tiež vygenerovat' trojuholníkovú siet', ktorá bude aproximovat' povrch implicitnej funkcie s čo najmenšou spotrebou pamäte a zároveň čo najrýchlejšie.
Prvej kapitole popíšeme prehľad teórie. V druhej kapitole predstavíme použité technológie, ktoré využijeme na implementáciu. V tretej kapitole popisujeme

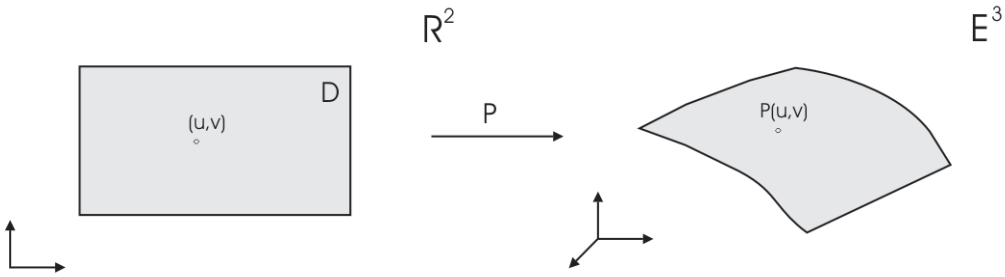
postup práce, v štvrtej kapitole analyzujeme použité algoritmy. V piatej kapitole aanalyzujeme pamäťovú náročnosť a rýchlosť použitých algoritmov vizualizujeme výsledky implicitnej funkcií. V šiestej kapitole predstavujeme dosiahnuté výsledky a možnosti budúcej práce.

2 Plochy a spôsoby ich zádavania

V tejto kapitole zavedieme základné definície a tvrdenia, ktoré budeme využívať v nasledujúcich kapitolách. Oboznámime sa s parametrickymi plochami, implicitnou funkciou. Väčšinu materiálov sme čerpali z [1], [2].

2.1 Parametrické plochy

V geometrií a v počítačovej grafike sa plochy najčastejšie zadávajú *parametricky*, využitím *bodovej funkcie dvoch (číselných) premenných*, a to pomocou zobrazenia $P : D \to E^3$, kde D je oblasť v číselnej rovine R^2.
Klasický zápis: $P = P(u, v)$, $(u, v) \in D$,
V súradniciach: $P(u, v) = (x(u, v), y(u, v), z(u, v)), (u, v) \in D$,



Obrázok 1: Parametrická reprezentácia bodu $P(u, v)$ na ploche.

kde $x = x(u, v)$, $y = y(u, v)$, $z = z(u, v)$ sú číselné funkcie premenných $(u, v) \in D$ pozri obr.1. Premenné (u, v) nazývame *parametrami* bodu $P(u, v)$. Pod oblasťou v číselnej rovine rozumieme neprázdnu súvislú množinu D, ktorej uzáver je uzáverom jej vnútra: $\bar{D} = \bar{intD}$ (pozri [1]).
Parciálne derivácie bodových, vektorových a číselných funkcií skrátene označujeme pomocou dolných indexov, napr.

$$P_u = \frac{\partial P}{\partial u} = (x_u, y_u, z_u), \quad P_v = \frac{\partial P}{\partial v}, P_{uu} = \frac{\partial^2 P}{\partial u^2}, \quad P_{uv} = \frac{\partial^2 P}{\partial u \partial v}, \quad P_{uv} = \frac{\partial^2 P}{\partial v^2}$$

Od parameterického vyjadrenia plochy

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

kde $(u, v) \in D$, sa vyžaduje, aby funkcia $P(u, v)$ bola *hladká* a *regulárna.*

Funkcia $P = P(u, v)$ je hladká, ak pre ňu existujú parciálne derivácie všetkých rádov, t.j. existujú parciálne derivácie všetkých rádov funkcií $x(u, v)$, $y(u, v)$, a $z(u, v)$.

Funkcia $P = P(u, v)$ je regulárna, ak vektory P_u a P_v sú lineárne nezávislé pre všetky hodnoty premenných u a v. To je ekvivalentné s podmienkou $P_u \times P_v \neq \overrightarrow{0}$ všade, resp. hodnost' *Jacobiho matice*

$$\frac{\partial(x, y, z)}{\partial(u, v)} = \begin{pmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \end{pmatrix}$$

je 2.
Plocha sa niekedy určuje rovnicou $F(x, y, z) = 0$ pre neznáme súradnice x, y a z, kde , ktorá spĺňa *podmienku hladkosti* a aj *regulárnost* t.j.
grad $F(x, y, z) \neq 0$ (odsek 2.3).

Parciálna derivácia bodovej funkcie $p(u, v)$ je vektorová funkcia

$$\frac{\partial p(u, v)}{\partial u} = \left(\frac{\partial x(u, v)}{\partial u}, \frac{\partial y(u, v)}{\partial u}, \frac{\partial z(u, v)}{\partial u}, 0 \right),$$

prípadne

$$\frac{\partial p(u, v)}{\partial v} = \left(\frac{\partial x(u, v)}{\partial v}, \frac{\partial y(u, v)}{\partial v}, \frac{\partial z(u, v)}{\partial v}, 0 \right)$$

. Pre $u = a$, $v = b$ určuje vektor dotyčnice parametrickej $u-$čiary, príp. $v-$čiary v bode $P(a, b)$, ktorý označujeme $\vec{p_u}(a, b)$, príp. $\vec{p_v}(a, b)$.

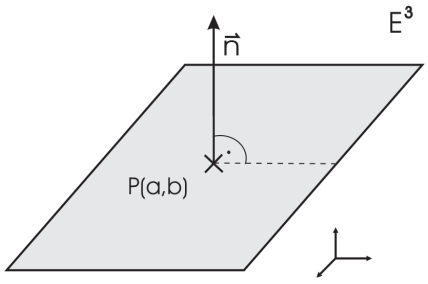
Bod plochy, v ktorom je niektorá z parciálnych derivácií bodovej funkcie plochy nulovým vektorom, prípadne sú vektory v tomto bode lineárne závislé, nazývame *singulárny bod.*

V regulárnom bode $P(a, b)$ sú vektory dotyčníc parametrických čiar nenulové

a lineárne nezávislé a definujú jedinú dotykovú rovinu $\tau(a, b)$ plochy. Dotyková rovina τ v regulárnom bode $P(a, b)$ plochy obsahuje dotyčnice všetkých čiar, ktoré ležia na ploche a prechádzajú bodom dotyku. Vektor

$$\vec{n}(a, b) = p_u(a, b) \times p_v(a, b)$$

nazývame *vektorom normály plochy* v regulárnom bode $P(a, b)$. Je kolmý na dotykovú rovinu a pre kladne orientovanú plochu je orientovaný polpriestoru od dotykovej roviny ako plocha.



Obrázok 2: Normála plochy v bode $P(a, b)$

Priamka určená vektorom normály a prechádzajúca bodom $P(a, b)$ sa nazýva *normála plochy* pozri obr.2. Normála plochy je kolmá na dotyčnice všetkých čiar plochy prechádzajúcich bodom $P(a, b)$, ktoré sú priamkami dotykovej roviny $\tau(a, b)$.

2.2 Implicitne zadané plochy

Implicitná funkcia je určená rovnicou, ktorá každému bodu v priestore priraďuje určitú hodnotu. Táto hodnota môže byť hustota objektu alebo intenzita sily v danom mieste v priestore.
Rovnice, ktoré opisujú implicitne definovanú plochu, vystupujú implicitné funkcie a ich hodnota závisí na polohe bodu *(v Euklidovskom trojrozmernom priestore E^3)*

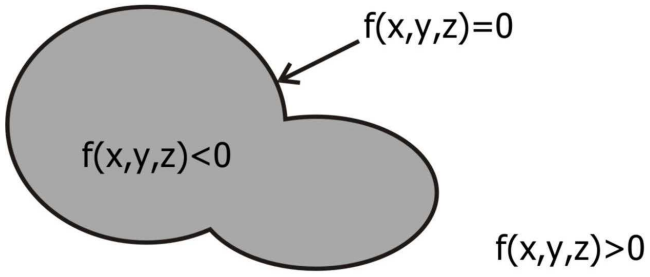
Nech $F(x, y)$ je spojitá funkcia. Rovnica $F(x, y) = 0$ môže definovať funkciu

$y = y(x)$, ktorá spĺňa rovnicu, teda platí $F(x, y(x)) = 0$. Hovoríme, že funkcia $y = y(x)$ je určená implicitne.

Veta.: Daná je rovnica $F(x, y) = 0$ a pre bod $A = [x_0, y_0]$ platí $F(x_0, y_0) = 0$. Nech funkcia $F(x, y)$ je diferencovateľná v bode A a nech platí $F'_y(x_0, y_0) \neq 0$.
Potom existuje jediná spojitá funkcia $y = y(x)$ určená implicitne rovnicou $F(x, y) = 0$ v istom okolí bodu x_0, pričom $y(x_0) = y_0$.
Derivácia tejto implicitne určenej funkcie v bode x_0 je daná vzorcom

$$y'(x_0) = \frac{F'_x(x_0, y_0)}{F'_y(x_0, y_0)}.$$

Podrobnejší výklad, príklady a tiež výpočet parciálnych derivácií implicitne danej funkcie určenej rovnicou sú v [2]



Obrázok 3: Povrch definovaný implicitne pomocou funkcie

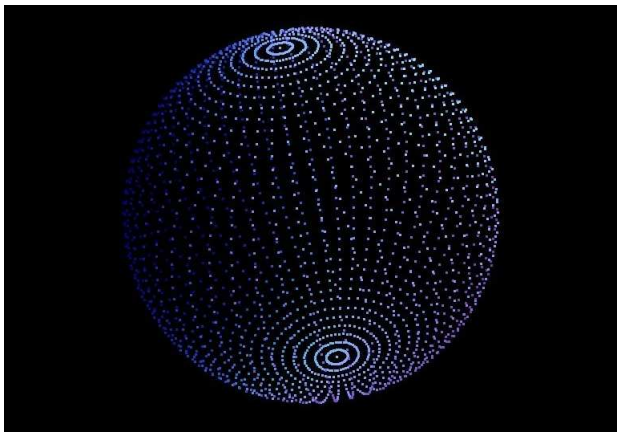
- pre ($f(x, y, z) < 0$ sa bod nachádza vo vnútri)
- pre ($f(x, y, z) = 0$ sa bod nachádza na ploche)
- pre ($f(x, y, z) > 0$ sa bod nachádza mimo plochy)

2.3 Povrchová reprezentácia

Povrchová reprezentácia pracuje len s povrchom objektu, ktorý môže byť daný analyticky alebo aproximovaný pomocou bodov, úsečiek či polygónov. Povrch telesa môže byť zadaný, *analyticky* presne, alebo pomocou vhodnej *aproximácie* môžeme nahradiť zakrivené časti povrchu rovinnými plôškami v tvare mnohouholníka. Analytické vyjadrenie je presnejšie a jednoduchšie na uchovávanie, aproximácia je však rýchlejšia. Pre podrobnejšie informácie pozrite [3] [6].

2.3.1 Reprezentácia pomocou vrcholov

Najjednoduchšie môžeme reprezentovať povrch telesa pomocou množiny bodov, ktoré sa na povrchu nachádzajú. Body okrem svojich súradníc môžu obsahovať aj nejakú dodatočnú informáciu, napr. farbu, či normálu v danom bode pozri obr. 4

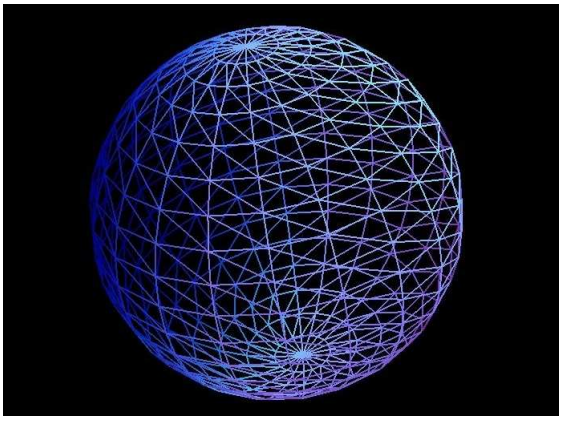


Obrázok 4: Reprezentácia pomocou vrcholov [5].

2.3.2 Reprezentácia pomocou hrán

Plocha sa dá reprezentovať okrem vrcholov aj pomocou hrán. Ak je povrch telesa daný vrcholmi a hranami, ktoré tieto vrcholy spájajú, reprezentácia sa

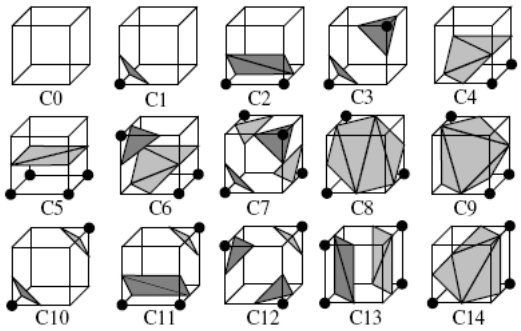
nazýva drôtený model *wireframe*. Vrcholy aj hrany môžu aj v tomto prípade obsahovať nejakú ďalšiu informáciu okrem súradníc pozri obr.5.



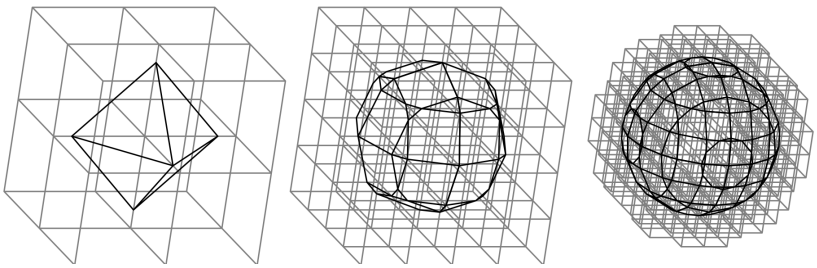
Obrázok 5: Reprezentácia pomocou hrán [5].

2.4 Marching cubs

Tento algoritmus sa používa na zostrojenie plochy. Výstupom je sieť trojuholníkov, ktoré sú aproximáciou plochy pre danú hodnotu. Princíp metódy spočíva v tom, že prerozdelíme $3D$ priestor na kocky. Sleduje v rámci kocky či jej vrcholy ležia vo vnútri danej plochy alebo vonku. Na základe toho určí trojuholníky, ktoré aproximujú tú časť plochy, ktorá danú kocku pretína. Na obrázku 6. je 15 základných konfigurácií ako môžu kocky pretínať danú plochu, ostatné z nich vzniknú symetriou, rotáciou, prípadne inverziou. Viac sa môžeme dočítať v [12] [13] [14]



Obrázok 6: Marching cubs 15 základných konfigurácií ako môžu kocky pretínať danú plochu

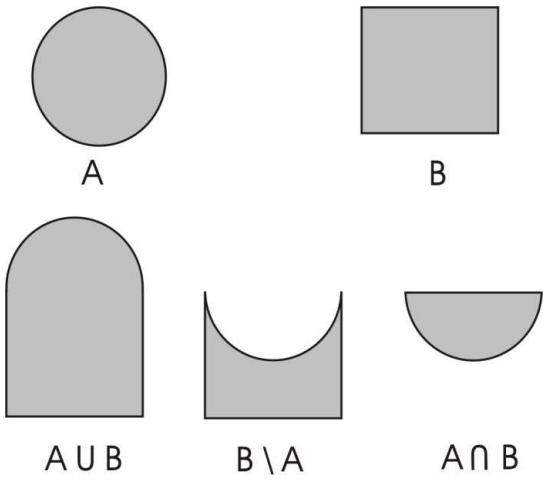


Obrázok 7: Prerozdelenie [12.]

2.5 Konštruktívna geometria telies - CSG

Pri konštruktívnej geometrií telies *(Constructive Solid Geometry, CSG)* uvažujeme niekoľko základných telies, z ktorých potom skladáme výsledný objekt. Na to používame transformácie a boolovské operácie. Základné telesá nazývané aj ako CSG primitívy môžu byť objekty ako napr. kváder a guľa. Pomocou boolovských operacií(zjednotenie, prienik a rozdiel) môžeme vytvárať nové objekty pozri obr.8. Tie reprezentujeme v stromovej štruktúre (*CSG strom*). [6]
Booleovské operácie sú definované:

- $(f \cup g)(\mathrm{p}) := \max(\mathrm{f(p)},\mathrm{g(p)})$
- $(f \cap g)(\mathrm{p}) := \min(\mathrm{f(p)},\mathrm{g(p)})$
- $(f \setminus g)(\mathrm{p}) := \min(\mathrm{f(p)},\mathrm{-g(p)})$



Obrázok 8: Boolovské operácie

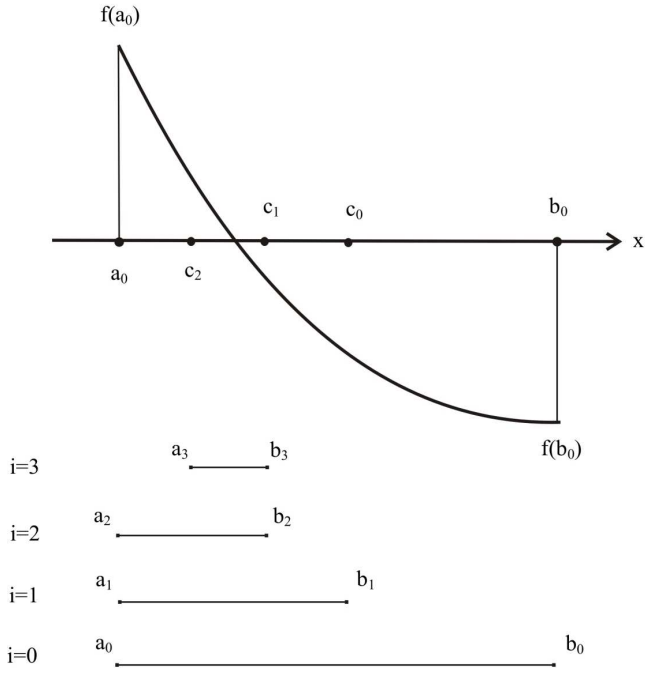
2.6 Numerické metódy

V tejto časti sa budeme zaoberať numerickým riešeniam rovníc s jednou neznámou tvaru $f(x) = 0$, kde x je číslo a nazýva sa koreň. Riešiť danú

rovnicu znamená nájsť jej koreň, pričom musí platiť, že počiatočná aproximácia je dostatočne blízko k hľadanému koreňu a iteračná metóda vždy konverguje. Popíšeme numerické metódy, ktoré spĺňajú vyššie uvedené predpoklady a niektoré z nich použijeme v práci.

2.6.1 Metóda bisekcie

Metóda bisekcie je založená na skutočnosti, že spojitá funkcia mení znamienko v okolí koreňa. Ku konvergencii stačí, aby funkcia bola spojitá, a derivácia nemusí vôbec existovať. Ak interval obsahuje viac ako jeden koreň, bisekčná



Obrázok 9: Bisekčná metóda na itervale (a_0, b_0)

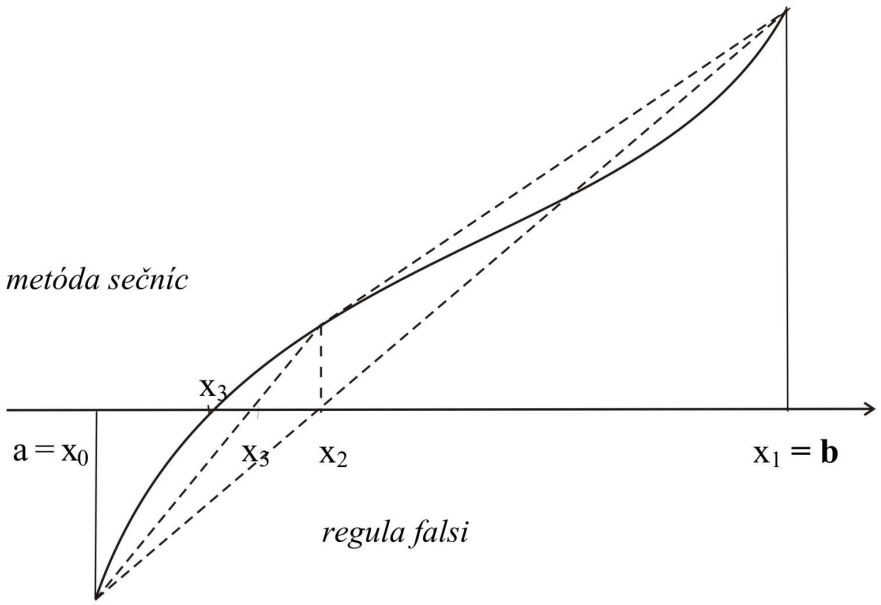
metóda vie nájsť aspoň jeden z nich. Predpokladajme že chceme riešiť rovnicu $f(x) = 0$, kde f je spojitá funkcia.

Bisekčná metóda delí interval medzi dvoma bodmi $f(a_0)$ a $f(b_0)$, ktoré majú opačné znamienka na polovicu. Z novo vzniknutých intervalov si zvolíme

ten, v ktorom majú hodnoty krajiných bodov opačné znamienka a pokračuje, kým nenarazí na koreň.

2.6.2 Metóda sečníc

Teraz uvedieme dve najjednoduchšie numerické metódy, ktoré umožnia nájsť koreň rovnice. Predpokladajme že f je spojitá funkcia na intervale $\langle a, b\rangle$ a nech $f(a)f(b) < 0$. Vieme vypočítať hodnotu $f(x)$ pre $x \in \langle a, b\rangle$ s dostatočnou presnosťou. Potom metóda inverznej interpolácie sa dá modifikovať



Obrázok 10: Metóda sečníc

tak, že k zvolenému bodu $a = x_0 < x_1 ... < x_i = b$ pribudne ďalší bod x_{i+1}, pre ktorý sa hodnota interpolačného polynómu inverznej funkcie rovná nule a celý postup opakujeme. Ak nechceme zvýšiť stupeň interpolačného polynómu vynecháme jeden bod napr. x_0 Ak je inverzná interpolácia lineárna (vtedy uvažujeme len posledné dva body. x_{i-1} a x_i.), hovoríme o *metóde*

sečníc (pozri obr.10) Vtedy platí

$$x_{i+1} = F_i(x_i, x_{i-1}) = x_i - \frac{x_i - x_{i-1}}{y_i - y_{i-1}} \cdot y_i;$$

kde $y_j = f(x_j)$.
Ak je inverzná interpolácia lineárna, funkcia f nemá v intervale (a, b) inflexný bod (t.j. bod, v ktorom sa funkcia mení z konvexnej na konkávnu alebo opačne), je spojitá a platí $f(a)f(b) < 0$, potom vieme najsť jednoduchý koreň metódu *regula falsi*:

$$x_{i+1} = \frac{y_i}{y_i - y_0} \cdot x_0 + \frac{y_0}{y_0 - y_i} \cdot x_i; (i = 1, 2, ...)$$

kde x_0 a x_i sú hranice intervalu $\langle a, b \rangle$.

2.6.3 Newtonova - Raphsonova metóda

Rád ľubovoľnej jednobodovej iteračnej metódy danej iteračnou funkciou $F(x)$ je prirodzené číslo. Podrobnejšie, $F(x)$ je rádu r vtedy a len vtedy, ak platí:

$$F(\lambda) = \lambda, F^{(j)}(\lambda) = 0, \quad pre \quad 1 \leq j < r, \quad F^{(r)}(\lambda) \neq 0.$$

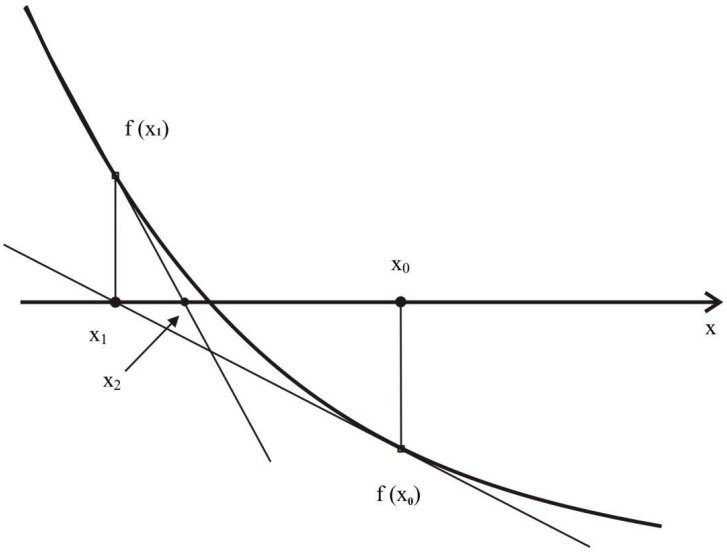
Najčastejšie používaná je Newtonova-Raphsonova metóda:

$$x_{i+1} = F(x_i) = x_i - f(x_i)/f'(x_i).$$

Funkciu f nahradíme jej dotyčnicou v bode x_i a nájdeme koreň dotyčnice, ak existuje. Pretože je pre $f(\lambda) = 0$, $F(\lambda) = \lambda$, $F'(\lambda) = 0$,
$F''(\lambda) = -f''(\lambda)/f'(\lambda)$, je táto metóda pre jednoduchý koreň aspoň druhého rádu. Pre p-násobný koreň má Newtonova-Raphsonova metóda tvar:

$$x_{i+1} = x_i - p.f(x_i)/f'(x_i).$$

a je tiež aspoň druhého rádu.



Obrázok 11: Newtonova-Raphsonova metóda

Uvedieme príklad, kde porovnávame spomínané metódy.

Príklad 9.

Hľadáme koreň rovnice $5x^3 + 12x^2 - 17x - 24 = 0$ ležiaci v intervale $(1, 2)$. V tabuľke sú prvé aproximácie koreňa nájdené uvedenými metódami. Výpočet sa prerušil, keď bolo $|x_{i+1} - x_i| < 5.10^{-3}$.

Metóda	**Sečníc**	**Regula falsi**	**Delenie intervalu napoly=Bisekcia**	**Newtonova -Raphsonova**
i	x_i	x_i	x_i=$(a_i + b_i)/2$	x_i
0	1	1	1,5	2
1	1,4444	1,4444	1,75	1,6703
2	1,5422	1,5667	1,625	1,6028
3	1,606013	1,5932	1,5625	1,6001
4	1,59998	1,5986	1,59375	1,6000003
5		1,5997	1,609375	

Presná hodnota koreňa je $\lambda = 1, 6$.

3 Použité technológie

3.1 C#

V angličtine *Si-sharp* je objektovo-orientovaný programovací jazyk vyvinutý spoločnost'ou *Microsoft*. Za základ pre nový jazyk *C#* si *Microsoft* zobral *C++* a jazyk *Java*.
C# bolo navrhované s úmyslom vyvážiť silu jazyka *C++* s možnost'ou rýchleho programovania "rapid application development", ktoré ponúkali jazyky ako napríklad *Visual Basic a Delphi*.

Vlastnosti jazyka
Tento jazyk bol priamo navrhnutý tak, aby umožňoval využitie všetkých vlastností, ktoré poskytuje CLI*(Common Language Infrastructure)*, to znamená, že kompilátor jazyka *C#* nemusí mat' za cieľovú podpornú platformu priamo CLI, respektíve vôbec nemusí generovať medziprekladový jazyk MSIL *(Microsoft Intermediate Language)* ani žiaden iný formát. Teoreticky je možné vytvoriť kompilátor jazyka *C#,* ktorý bude prekladať priamo do strojového kódu ako tradičné kompilátory jazyka *C++*, *Fortran* a podobne. Triedy sú väčšinou organizované do menných priestorov namespace. Na rozdiel od *C* a *C++* umožňuje lepšiu čitateľnosť zdrojového kódu. Vďaka tomu sa stal obľúbením programovacím jazykom. [9]

3.2 DirectX

DirectX je programátorské rozhranie *(Application Programming Interface)* pre multimediálne programy a pre počítačové hry fungujúce na platforme *Windows*. *DirectX* nie je len obyčajné rozhranie, ale je to zbierka rôznorodých komponentov, ktorá ponúka podporu pre audio a rôzne vstupné zariadenia (napr. myš, joystick) a sieťovú komunikáciu. Pokrýva takmer celú multimediálnu oblasť. Poskytuje veľmi kvalitné rozhranie pre prácu s *2D* a *3D grafikou*. Momentálne je aktuálna verzia *DirectX 10.1* pod operačným systémom Windows Vista. V práci som používala *DirectX 9.0c.*
Direct3D je trojrozmerné rozhranie, teda obsahuje mnoho príkazov na trojrozmerné renderovanie, obsahuje však aj niekoľko príkazov na dvojrozmernú grafiku. Toto rozhranie je pravidelne aktualizované, aby podporovalo posledné technológie grafických kariet. Podporuje plnú softvérovú emuláciu vertexov, no nepodporuje žiadnu emuláciu pixelových vlastností, ktoré nie sú podporované hardvérom.
Napríklad, ak je program naprogramovaný na používanie *Direct3D* s pixel shaderom a grafická karta to nepodporuje, *Direct3D* to nebude emulovať.
Aplikačné rozhranie definuje referenčný rasterizér alebo referenčné zariadenie, ktoré emuluje štandardnú grafickú kartu, hoci je to príliš pomalé, ak sa to použije na emuláciu v akejkoľvek aplikácií a pixel shadery sú obyčajne ignorované. Hlavným súperom rozhrania *Direct3D* je *OpenGL*. Viac sa o *DirectX* môžeme dočítať v [10] [11]. Vďaka ľahko prístupným tutoriálom na stránke Riemer XNA [10] máme možnosť vyskúšať *DirectX* pre C#. Základ zdrojového kódu sme použili priamo z tohto tutoriálu.

```
using Microsoft.DirectX; using Microsoft.DirectX.Direct3D;
staticvoid Main() {
  using (WinForm our_directx_form = new WinForm())
  {
      Application.Run(our_directx_form);
  }
```

Príklad: Vytvorenie poľa vrcholov trojuholníkov

```
CustomVertex.TransformedColored[]vertices = new
CustomVertex.TransformedColored[3];
```

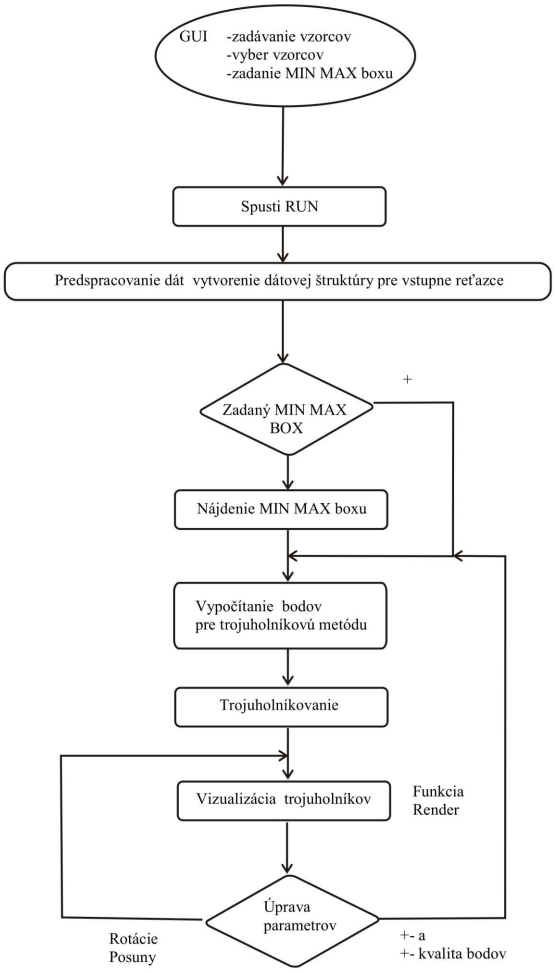
a ich vykreslenie:

```
device.BeginScene();
device.VertexFormat=CustomVertex.TransformedColored.Format;
device.DrawUserPrimitives(PrimitiveType.TriangleList, 1, vertices);
device.EndScene();
```

Viac informácií na [9].

4 Popis práce

4.1 Úvod



Obrázok 12: Algoritmus postupnosti krokov

V tejto kapitole popisujeme celý postup práce. Podrobne opíšeme nami navrhnuté algoritmy a interface medzi nimi, ktoré umožnia spracovať implicitne zadanú funkciu až po trojuholníkovú vizualizáciu funkcie danej implic-

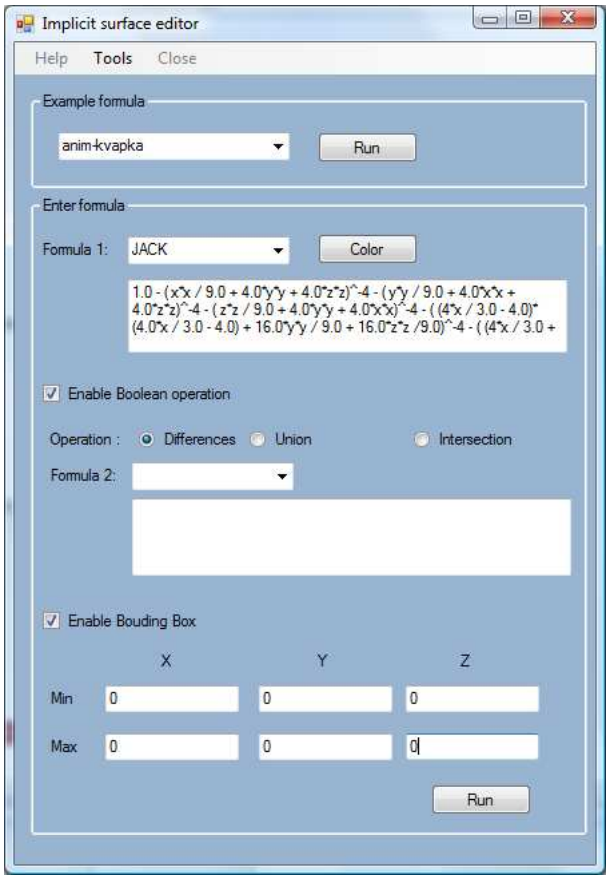
itne. Postupnosť krokov, znázorňená na vývojovom diagrame pozri obr. 12, nám umožní vytvoriť prehľad o prepojení jednotlivých funkcií, ktoré postupne detailnejšie vysvetlíme v jednotlivých podkapitolách. Práca sa delí na tieto hlavné časti:

1. program Implicit surface editor na načítavanie a zadávanie implicitných funkcií
2. rýchle vypočítanie implicitnej funkcie
3. obálka IF(bouding box)
4. triangulácia implicitnej funkcie
5. vizualizácia dát

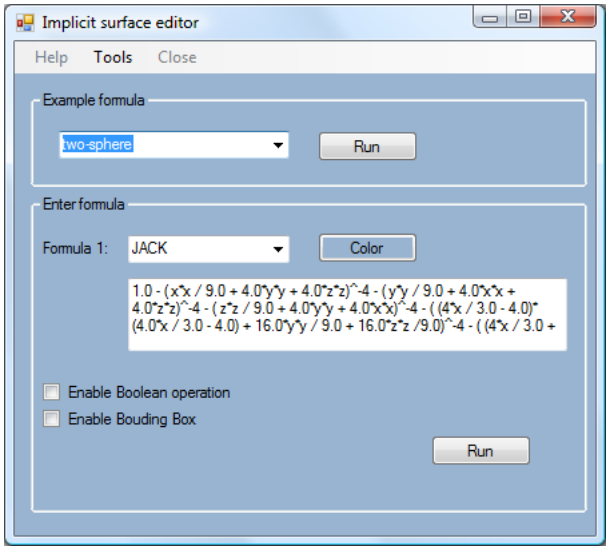
4.2 Popis programu Implicit surface editor

Program Implicit surface editor slúži na zadávanie implicitne definovaných plôch a objektov pre program VizualizáciaIF. Program sa skladá z troch častí pozri obr. 13, 14.

1. V hornej časti nám umožňuje vybrať špeciálne predpripravené skupiny implicitných funkcii, ktoré spolu tvoria objekt. Každému objektu zodpovedá jeden textový súbor, ktorý je možné editovať v programe (napr.: v programe Notepad).
2. V strednej časti zadávame (zvolíme) vzorec funkcie danej implicitne. V prípade, že zvolíme dva objekty tak možeme použiť medzi nimi jednu z troch booleovských operacií: rozdiel, zjednotenie a prienik týchto objektov. Takto nám vzniká veľká škála testovacích dát pre náš program. Program podporuje tieto typy operacií:
   $+, -, *, /, sin, cos, tan, asin, acos, atan$
   Ako premenné si môžme zvoliť':
   x, y, z a a.



Obrázok 13: Implicit surface editor



Obrázok 14: Implicit surface editor

Hodnotu premennej a môžeme potom pomocou kláves O a P meniť a tak môžeme pozorovať zmenu objektu (štandardne je hodnota a rovná 0).

3. V spodnej časti môžme nastaviť obálku IF (bouding box). Umožnuje nám zobraziť len časť objektu definovaného pomocou implicitne danej funkcie.

Po stlačení tlačítka RUN sa uložia informácie do textového súboru pre ďalšie spracovanie programom Vizualizácia IF.

4.3 Popis programu Vizualizácia IF

4.3.1 Rýchle vypočítanie IF zo zadaného vstupu

Na vstupe máme zadanú implicitnú funkciu (IF) vytvorene v programe Implicit surface editor v textovom súbore. Pre rýchly výpočet hodnoty IF v bode x,y,z potrebujeme vytvoriť parser, ktorý spracuje textový súbor a

vytvorí datovú štruktúru, ktorú potom neskôr použijeme na rýchle výpočty IF pomocou funkcie

```
void PredvypocitajDataIF(string retazecIF);
```

Základnou jednotkou pola objektov je objekt

```
 class ParseOperation
{
    public int i1, i2, i3;
    double d1, d2;
}
```

Objekt ParseOperation reprezentuje jednu matematickú operáciu $(+, -, /, *, sin, cos, tan)$ medzi jednou alebo dvoma hodnotami. Akýkoľvek vzorec premeníme na pole dát typu ParseOperation a priebežných výsledkov AV typu double.
Pre operáciu typu A (operácia) B bude hodnota i_1 a $i_3 \geq 0$ ak odkazuje do *pol'a AV*(dočasné pomocné pole kde ukladáme medzi výpočty, hodnotu $i-$teho riadku z list vypočtov uložíme do $i-$teho riadku *AV pola*) na príslušné miesto, ktoré obsahuje výpočet daného riadku. Hodnota i_1 (i_3) je

$= -2$ pre $A(B) = x$
$= -3$ pre $A(B) = y$
$= -4$ pre $A(B) = z$
$= -5$ pre $A(B) = a$ (dynamická hodnota použitá pre animáciu)
$= -1$ ak A je reálne číslo, vtedy $d1 = A$
$= -1$ ak B je reálne číslo, vtedy $d2 = B$

Hodnota i_2 rovná sa

5, pre operáciu *
6, pre operáciu /
7, pre operáciu +
8, pre operáciu -

9, pre operáciu $\wedge$
10, pre operáciu sin
11, pre operáciu cos
12, pre operáciu tan

Príklad:
$(x * 2 - y * z) \Longrightarrow$ spracujeme operáciu $x * 2$ do objektu ParseOperation ako

ParseOperation vzorec1 = new $(-2, 5, -1, 0, 2)$;

pričom výsledok tejto operácie sa vloží do pola AV po prvom kroku dostanem string(AV000 - y*z).

Takto pre zadanú IF vytvoríme pole ListVypoctov. Deklarácia v C# vyzerá nasledovne:

Arraylist ListVypoctov = new ArrayList();

ktorá obsahuje ParseOperation hodnoty. Na výpočet hodnoty funkcie danej implicitne (IF) v bode x, y, z, a použijeme funkciu

double VypocitajIF(double x, double y, double z, double a)

ktorá z predpripraveného poľa ListVypoctov vráti hodnotu IF v danom bode.

Takže príklad $(x * 2 - y * z)$ bude vyzerať v poli ListVypoctov nasledovne:

i	**0**	**1**	**2**	**3**	**4**
0	-2	5	-1	0	2
1	-3	5	-4	0	0
2	0	8	1	0	0

$$\underbrace{\underbrace{x * 2}_{A000} - \underbrace{y * z}_{A001}}_{A002}$$

4.3.2 Nájdenie obálky IF (bouding boxu)

Pre d'alšie spracovanie IF potrebujeme nájst' obálku IF (bouding box). V prípade, že nemáme na vstupe zadanú obalku IF tak väčšina programov používa na nájdenie začiatočného bodu jednu z troch nasledujúcich metód:

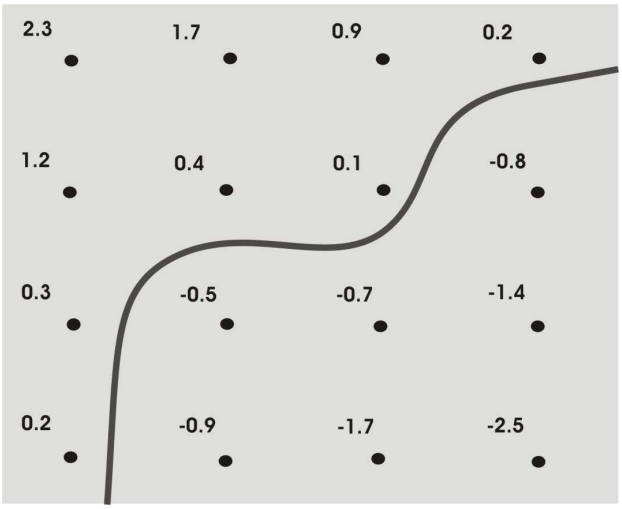
- *Exhaustive search*
- *Algoritmus Random search*
- *Algoritmus Random search*

Ako je popísané v práci [13] nevýhodou týchto metód je skutočnost' že nenájdu oddelené časti objektov (pozri obr. 27), preto spravíme nový algoritmus , ktorý bude prehl'adávat' celý priestor.

Náš algoritmus používa nasledujúce metódy na hl'adanie obálky IF(bouding box):

1. **Hodnota funkcie IF v náhodne vygenerovanom bode.**
2. **Hl'adanie zápornej hodnoty IF** (ak nájdeme zápornú hodnotu použijeme funkciu BisekciaXYZ, ktorá nájde hodnoty implicitnej funkcie v smeroch x, y a z.)
3. **Využitie najmenšej nájdenej kladnej hodnoty** okolo ktorej budeme d'alej hl'adat' (t.j. ak hodnota IF pre $(x_1,\ y_1,\ z_1)$ je 1000, a hodnota pre $(x_2,\ y_2,\ z_2)$ je 10 predpokladame, že $x_2,\ y_2,\ z_2$ je bližšie k IF, takže okolo tohto bodu budeme d'alej hl'adat'). Pomocou tejto metódy nájdeme gul'u o polomere 1 v kocke s intervalom $(-100, 100)$ pre x, y, z t.j. v objeme v 200^3.

V prvom kroku testujeme kocku z intervalom $(-1, 1)$ v každom smere. V nej náhodne vygenerujeme body, v ktorých zist'ujeme či hodnota IF v danom bode je kladná alebo záporná. Ak nájdeme zápornú hodnotu IF dáme ho do pomocného listu Polebodov a hl'adáme d'alej. Ak sme našli dostatočný počet bodov (pre nás je to 10 bodov) zo zápornou hodnotou IF, tak na



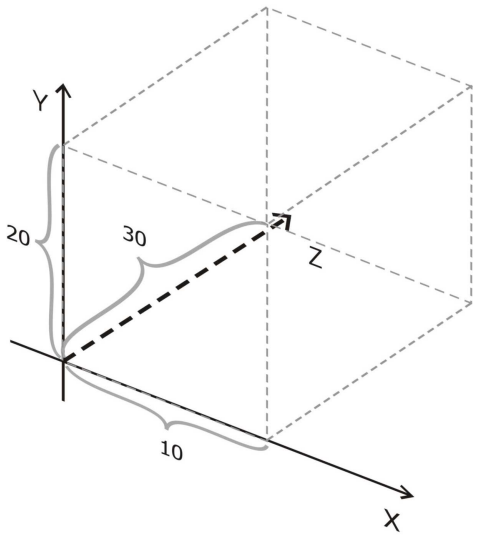
Obrázok 15: Aproximácia implicitnej funkcie.

týchto nájdených bodoch použijeme našu numerickú metódu BisekciaXYZ. A takto získaných bodov vytvoríme obálku (bouding box). Ak sme nenašli dostatok bodov tak postupne zväčšujeme interval hľadania až na hodnotu $(-100, 100)$. Ak ani vtedy nenájdeme dostatok bodov, tak použijeme metódu hľadania najmenšej kladnej hodnoty okolo, ktorej budeme hľadať ďalej. Ak narazí aspoň na jeden prechod (t.j. medzi kladnou a zapornou hodnotou) pri 1400 výpočtoch, oplatí sa zväčšiť prehľadávaci priestor, aby sme mali väčšiu pravdepodobnosť narazit' na ďalší prechod. Pretože z implicitnej funkcie vyplýva, že pri jednom nájdenom bode existuje aj ďalší hľadaný bod.
Kombináciou týchto metód nájdeme obálku (bouding box) zadanej implicitnej funkcie, ak existuje.

4.3.3 BisekciaXYZ

Majme bod $B(x, y, z)$. Ak hodnota IF v bode B je záporná, tak B je vnútri objektu. Vypočítame pre neho 6 vrcholov, t.j. hľadáme na intervaloch $(x,\ 100)$ a $(-100,\ x)$ body, v ktorých hodnota IF bude kladná. To isté urobíme v smere y a z. Takto postupujeme pre všetky body z poľa Pole-

bodov. Z nových bodov, ktoré nájdeme spravíme obálku, t.j. nájdeme všetky $min\ x$, $min\ y$, $min\ z$, a $max\ x$, $max\ y$, $max\ z$. Takto získame hľadanú *obálku IF (bouding box)* pozri obr. 16.



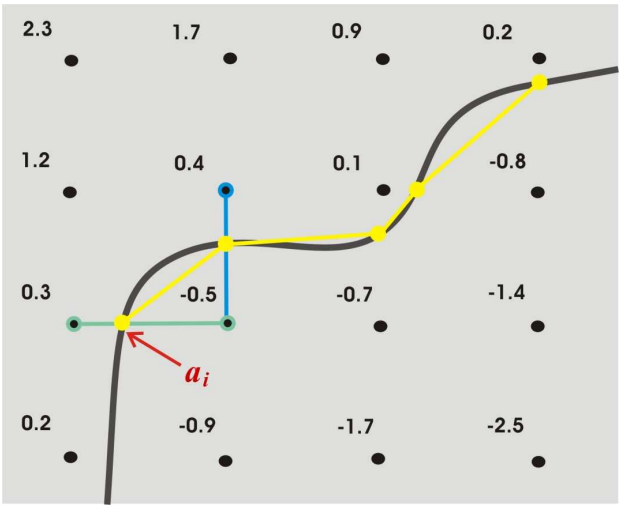
Obrázok 16: Obálka IF (bouding box)

Ak používateľ zadal v programe Implicit surface editor vlastnú obálku IF (bouding box), tak ju použijeme.

4.3.4 Vypočítanie hodnôt implicitnej funkcie na nájdenej obálke IF(bouding box)

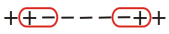
Z obálky IF (bouding box) získame hodnoty v_x, v_y, v_z, čo sú rozmery obálky IF (bouding box) ako kvádra. Ten budeme postupne rezať v smere x (pre krok 3 v smere y). Pre každý rez A_j, kde $j \in < 0, v_x >$ vytvoríme pole bodov vypočítaných z IF, t. j. rez $A_j = (a_0, a_1, ..., a_n)$, kde a_i, pre $i \in < 0, n >$, sú body IF. Body a_i vyrátavame pre rez A_j podľa nasledovných krokov:

1. Pre každý rez A_j hľadáme y_k, kde $k \in < 0, v_y >$, hodnoty z, v ktorých nastane prechod hodnoty IF do kladného zo záporného alebo naopak. Medzi týmito hodnotami je náš hľadaný bod a_i. Na jeho presnejšie nájdenie použijeme numerický algoritmus BisekciaZ (názov je podľa toho, že hľadáme z-tovú hodnotu), ktorý nájde z-tovú hodnotu so zadanou toleranciou. Takto nájdený bod pridáme do pola bodov pre príslušný rez A_j.



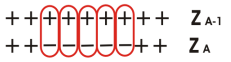
Obrázok 17: Bisekčná metóda nájde bod IF na intervale (zelená úsečka v smere Z, modrá v smere Y). Pospájané body (žlté úsečky) aproximujú IF.

Obrázok pre zadanú x-ovú a y-ovú hodnotu hľadáme z-tovú hodnotu $z \in < 0, v_z >$



2. Podobne ako v prvom kroku, hľadáme prechod hodnoty IF do kladného zo záporného alebo naopak, ale v tomto kroku porovnávame hodnoty medzi y_{k-1} a y_k (respektíve y_{k+1} a y_k) pre zadané x a zvolené z.

   Obrázok pre zadané x a y_{k-1} a y_k hľadáme:

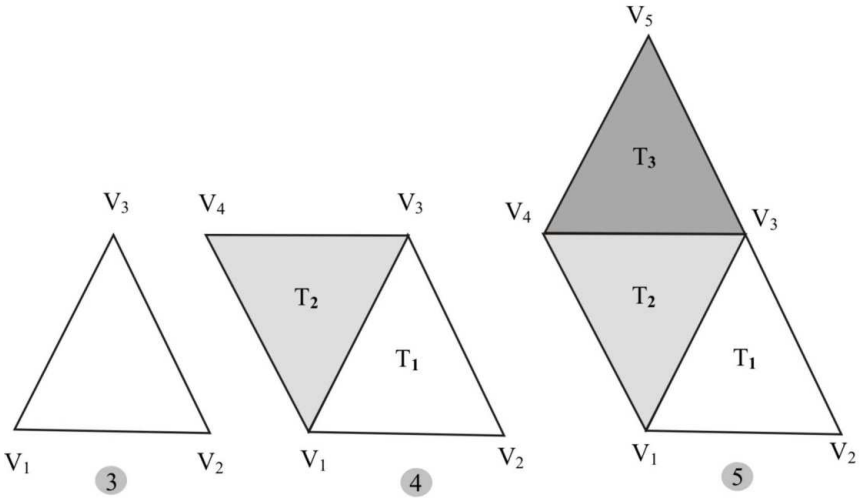


3. Tu budeme postupne rezať v smere y a použijeme predchádzajúce kroky s tým, že miesto rezov x robíme rezy y.

Potom pre každé dva rezy A_i a A_{i+1}, na ktorú použijeme numericky algoritmus BisekciaX. Podobne ako v predchádzajúcich krokoch, hľadáme prechod hodnoty IF do kladného zo záporného alebo naopak, ale v tomto kroku porovnávame hodnoty medzi x_{k-1} a x_k (respektíve x_{k+1} a x_k). Medzirez M_j kde $j \in < 0, v_x >$

4.3.5 Algoritmus na trianguláciu.

A_0 a A_1 čo sú listy, ktoré zahŕňajú všetky vrcholy pre daný rez, kde $A_0 = (a_0, a_1 \ldots\ldots a_n)$ $A_1 = (b_0, b_1 \ldots\ldots .b_m)$.
Náš algoritmus postupuje podobne ako MC a vytvára pomocou rezov mriežkovú sústavu, ktoré nám pripomína MC prerozdelenie na kocky. Tak ako pri sledovaní povrchu pri MC pozorujeme základné konfigurácie vo vrcholoch. Nakoľko využívame obálku implicitnej funkcie na základe ktorej, môžeme určiť vzdialenosť rezov $A_j(step)$. V prípade nedostatočnej kvality vizualizacie môžeme hodnotu *step* meniť. Vytvárať trojuholníky budeme vždy z troch rezov A_i a A_{i+1} a medzirezom M_i. Postupne prechádzame mriežkovú sústavu a po častiach (kockách) s rozmermi y a $y + step$ a z a $z + step$, z týchto troch rezov zoberieme body, ktoré nachádzajú na hrane daného intervalu. Na takto získané hodnoty použijeme nasledovný algoritmus.



Obrázok 18: Hľadáme postupnosť vrcholov vhodnú pre trianguláciu

Majme vrcholy V_0.....V_n a kde n je z intervalu $< 0, 11 >$. Hľadáme postupnosť vrcholov vhodnú pre trianguláciu.

1. Zvolíme si prvý bod.
2. Pre zvolený bod hladíme na susediacich stenách najbližší ďalší bod.
3. Druhý krok opakuje až kým nenájdeme ďalší bod.

Takto postuptupne prechádzame všetky vrcholy a získame vhodnú postupnosť pre trianguláciu.

Špeciálne prípady:

V prípade že algoritmus nenájde aspoň tri vrcholov posunieme sa na ďalšiu kocku.

Ak takýmto postupom nedostaneme rovnaký počet vrchov čo sme mali dispozíci na vstupe jednoducho algoritmu odfiltruje možný problém. Napr. na každej hrane by nachádzal jeden vrchol alebo pozri obr.18. Takto získanú postupnosť vrcholov pospájame trojuholníkmi a postupnosť nám zároveň minimalizuje možné problémy s trojuholníkmi. Vrcholy pospájame medzi sebou tak aby nevznikli medzi trojuholníkmi diery a zároveň sa neprekrývali. pozri obr. 18.

4.4 Popis programu Vizualizácia IF

Samotný program na vizualizáciu slúži na vykreslenie trojuholníkov vstupnej IF. Program možeme ovládať pomocou nasledovných kláves.

A ⇒ pomocou tohto tlačidla priblížime objekt
S ⇒ pomocou tohto tlačidla oddialime objekt
M ⇒ zväčšuje hustotu trojuholníkovej siete
N ⇒ zmenšuje hustotu trojuholníkovej siete
E ⇒ rotuje objekt okolo osi y v danom bode
R ⇒ rotuje objekt okolo osi x v danom bode
F5 ⇒ uloží trojuholník do stl súboru
H ⇒ Help zobrazí a skryje návod

Obrázok 19: Popis programu Vizualizácia IF

5 Analýza algoritmov

5.1 Pamäťová náročnosť

5.1.1 Analýza výpočtu IF

Na výpočet IF (viď. kapitola 4.3.1) potrebuje funkcia PredvypocitajdataIF pole objektov ParseOperation. Spotreba pamäte je ParseOperation * počet všetkých matematických operácií vo vzorci.

5.1.2 Analýza nájdenia obálky IF

Spotreba rovna pamäte algoritmu je rovná velkosti listu PoleBodov, ktorý obsahuje vrcholy, z ktorých potom vytvárame obálku IF $O(n)$.

5.1.3 Analýza algoritmu na vypočítanie hodnôt IF

Rezy A_0 až A_n obsahujú všetky vypočítané vrcholy IF. Preto pamäťová náročnosť je rovná počtu všetkých vypočítaných bodov IF pozri obr. 15.

5.1.4 Analýza algoritmu na trianguláciu

V pamäti vždy potrebujeme maximálne 3 rezy A_i, A_{i+1}a M_i, čo sú listy všetkých vrcholov pre daný rez. Pamäťová náročosť je $O(n^2)$

5.2 Rýchlosť

5.2.1 Analýza rýchlosti výpočtu IF

Neustále spracovanie string-u je príliš pomalé v C# tak že sme na výpočet IF vytvorili vlastnú štruktúru, ktorá sa raz vytvorí pre zadaný IF a na samotný výpočet sa použije naša štruktúra (ParseOperation), ktorá je len o máličko pomalšia ako priame zadanie vzorca do kódu. V našich testoch bola maximálne 10-krát pomalšia.

5.2.2 Analýza rýchlosti najdenia obálky IF

Tento algoritmus funguje na testovaní hodnoty IF a rozhoduje sa, či nájde hodnotu IF menšiu ako 0. Najhoršia zložitosť algoritmu je 6x1400x rýchlosť výpočtu IF v testovanom bode teda $O(n)$. Vtedy však algoritmus obálku IF nenájde.

5.2.3 Analýza rýchlosti algoritmu na vypočítanie hodnôt IF

Rýchlosť algoritmu je v_x*v_y*v_z *$\frac{1}{step^3}$ * rýchlosť výpočtu hodnoty v bode, kde step je vzdialenosť medzi jednotlivými rezmi a_j (viď kapitola 4.3.4). Teda zložitosť je $O(n^3)$ Takto výpočítavame veľa zbytočných hodnôt, ale vzhľadom ktomu, že rýchlosť je relatívne dostačujúca, sme túto časť neoptimalizovali. Tu vidíme priestor na možné urýchlenie.

5.2.4 Analýza rýchlosti algoritmu na trianguláciu

Trojuholníky vytvárame medzi dvoma rezmi, ktoré majú n respektíve m vrcholov. Keďže vrcholy sú usporiadané, testujeme vrcholy len vrámci jednej kocky. Celková zložitosť je potom $O(n^3)$.

5.3 Zhrnutie

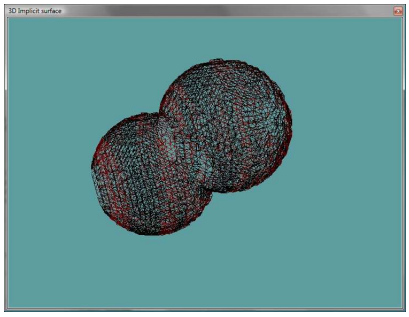
Vzhľadom k tomu, že sme dávali väčšiu prioritu na spotrebu pamäte, tak rýchlosť algoritmov nie je úplne ideálna a dala by sa zlepšiť. Existujú optimalizačné, rezervy rýchlosti algoritmov. Vzhľadom na dosahované výsledky

je však rýchlost postačujúca. Väčšina naších testovaných dát sa vykreslila do 1 sekundy.

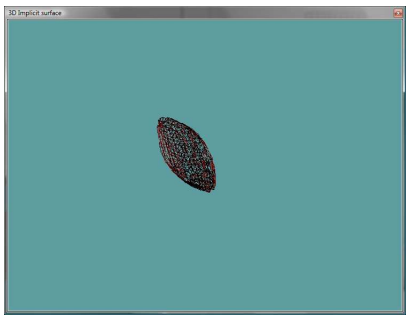
6 Výsledky

6.1 Boolovské operácie

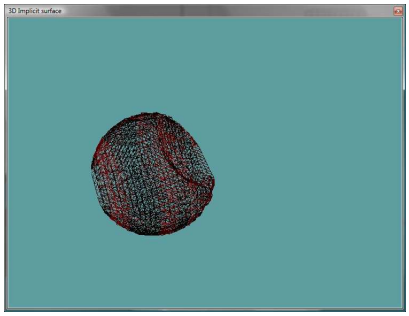
Demonštrujeme príklad zjednotenia, rozdielu a prieniku na dvoch implicitne zadaných guľových plochách.



Obrázok 20: Vizualizácia zjednotenia objektov



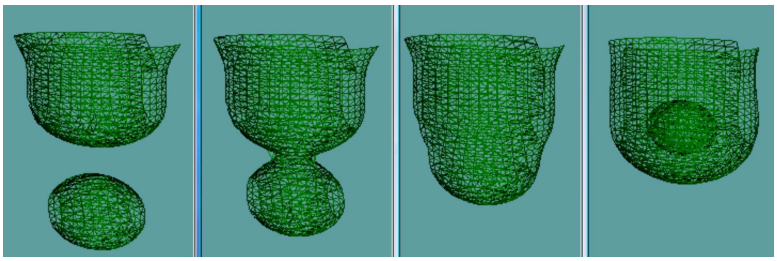
Obrázok 21: Vizualizácia prieniku dvoch objektov



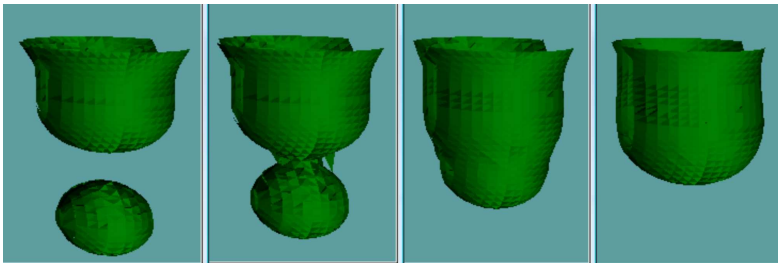
Obrázok 22: Vizualizácia rozdielu dvoch objektov

6.2 Animácia

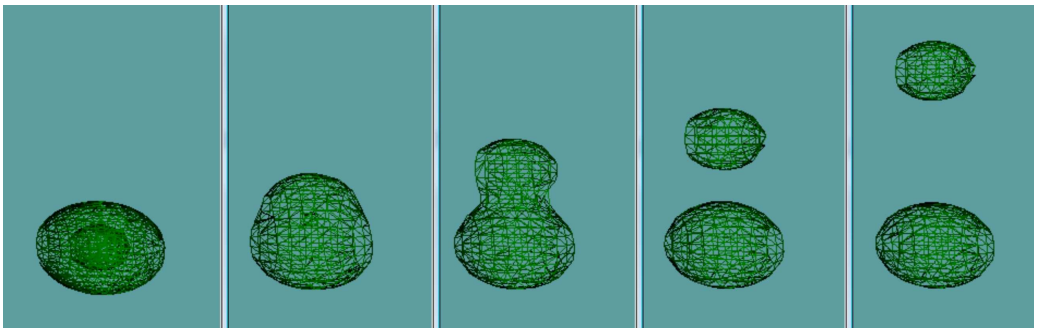
Pridaním novej premennej a (krok plus, mínus 0.04 (klávesy O a P)) do vzorca IF môžeme pozerat', aký má vplyv na vizualizáciu implicitnej funkcie.



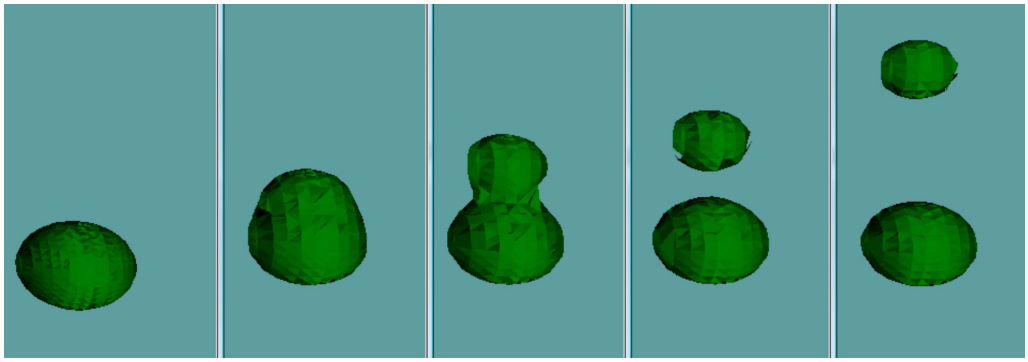
Obrázok 23: Animácia zmeny IF pomocou dynamickej premennej (hranová reprezentácia). $IF = ((x^2) + (y^3 * (a-1)) + z^2 - 1) * (x^2 + ((y - (a-1) * 4) * (y - (a-1) * 4)) + z^2 - 0.5) - 0.15$ ak $a = 0.36$, $a = 0.44$, $a = 0.6$, $a = 0.8$



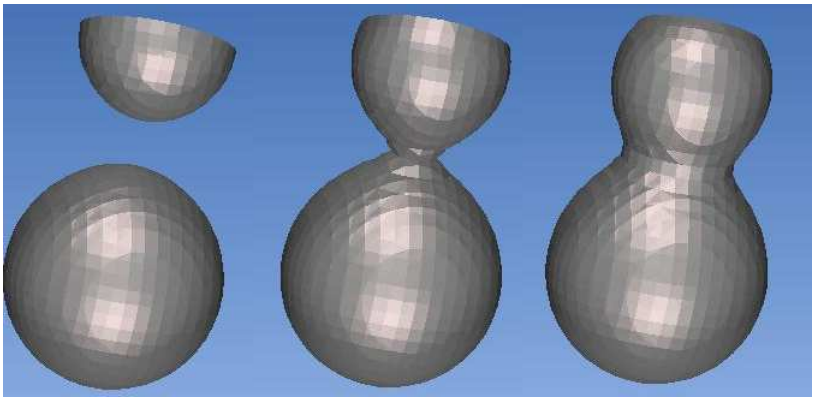
Obrázok 24: Animácia zmeny IF pomocou dynamickej premennej (trojuholníková reprezentácia). $a = 0.36$, $a = 0.44$, $a = 0.6$, $a = 0.8$



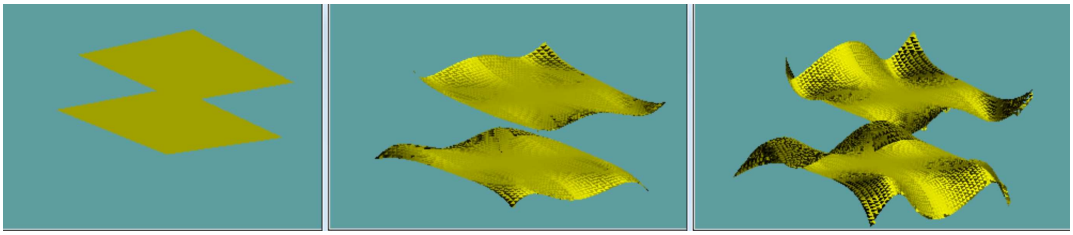
Obrázok 25: Animácia zmeny IF pomocou dynamickej premennej (hranová reprezentácia). $IF = (x^2 + y^2 + z^2 - 1) * (x^2 + ((y - a^2) * (y - a^2)) + z^2 - 0.5) - 0.15a = 0$, $a = 1$, $a = 1.36$, $a = 1.56$, $a = 2$



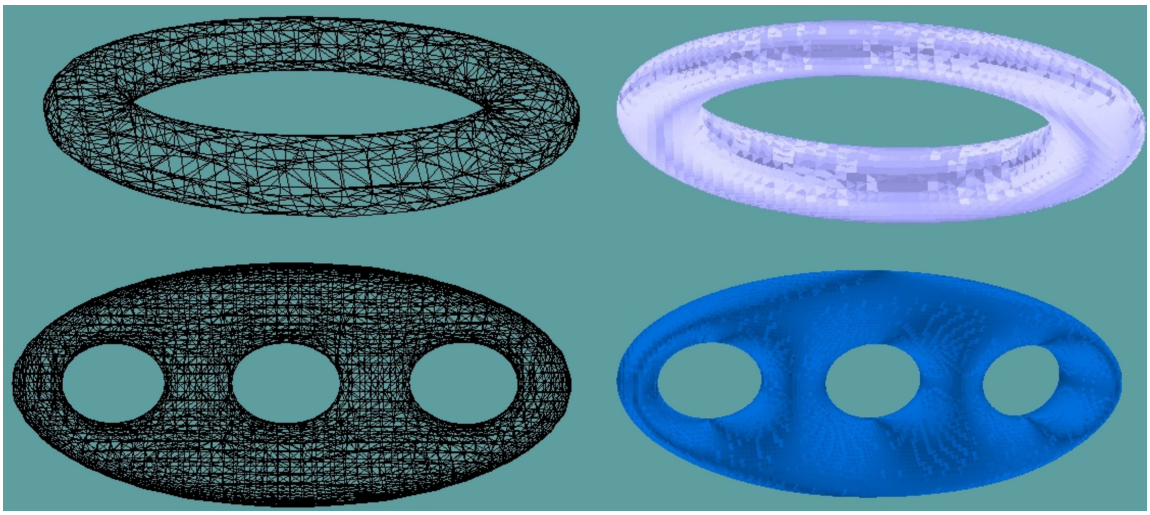
Obrázok 26: Animácia zmeny IF pomocou dynamickej premennej (trojuholníková reprezentácia). $a = 0$, $a = 1$, $a = 1.36$, $a = 1.56$, $a = 2$



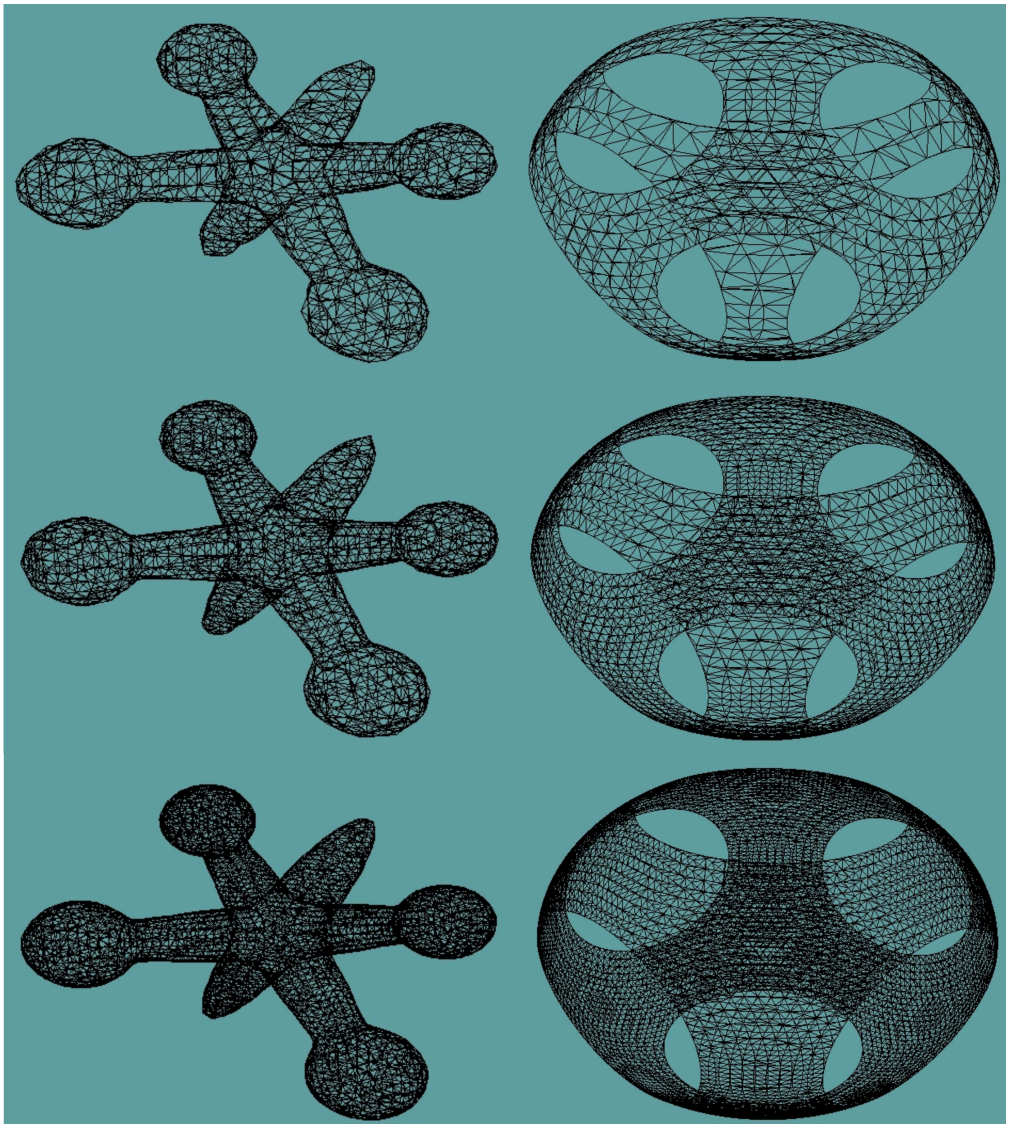
Obrázok 27: zmeny hodnoty a a zobranených v programe STL viewer.



Obrázok 28: Animácia zmeny IF pomocou dynamickej premennej (trojuholníková reprezentácia dvoch dosiek). $(y^2 * a) * (4x^2 * (1 - x^2) - y^2) + z^2 - 1/4$ $a = 0$, $a = 0.2$, $a = 0.64$



Obrázok 29: Vizualizácia objektv Torus a Genus3)



Obrázok 30: Vizualizácia objektu Jack, počet trojuholníkov zhora dole postupne 1560, 2824, 2824 a Wiffle cube(vpravo) 2638, 4224, 10932

7 Záver

Primárnym cieľom práce bolo zostavenie funkčného a rozšíriteľného zdrojového kódu na aproximáciu implicitne definovaných plôch. Cieľom bolo vytvoriť vzorový program s komentármi, ktorý bude voľne dostupný z katedrálneho servera. Práca nadväzuje na dve diplomové práce obhajované v šk. roku 2005/06 zamerané na čiastočné riešenie niektorých podproblémov. Práca Ondreja Jabornika riešila problémy detekcie hrán a rozšírenie známych algoritmov MC a MT. Vybrali sme sa iným smerom. Snažili sme sa čo najjednoduchšie sprístupniť problematiku IF pre bežného užívateľa, t. j. pokiaľ možno bez komplikovaných parametrov, jednoduché GUI a príklady na vizualizáciu. Vytvorili sme vlastné metódy, spravili sme rýchly parser, ktorý zo zadaného vstupného reťazca (objekt string v C#) vypočíta hodnotu IF v bode, následne ďalším algoritmom nájde obálku IF (bouding box) a potom vytvorí trojuholníky z vypočítaných bodov ktoré získame pomocou numerickej metódy. Pridali sme aj podporu booleovských operácií a animácie, kde si môže užívateľ dynamicky meniť hodnotu parametra vstupného vzorca. Program bol spravený v najnovšom vývojovom prostredí MS Visual Studio 2008 v programovacom jazyku C# a na vizualizáciu používa Direct X.

7.1 Budúca práca

Vylepšenie práce vidíme v pridaní nových vzorcov, zadávaní booleovských operácií priamo do vzorcov, zlepšení kvality triangulácie a optimalizovaní algoritmu na hľadanie obálky.

8 Literatúra

Referencie

[1] M. Božek, *Učebný text k predmetu Geometria(2).* Nepublikované, 2004.

[2] dopnit, *Matematická analýza.* Matematika 1, odst. 7.5.

[3] J. Žára, B. Beneš, J. Sochor, P. Felkel, *Moderní počítačová grafika(2.vydání)*:,Computer Press, 2005

[4] `http://en.wikipedia.org/wiki/Computer_graphics`

[5] M. Remešíková www.math.sk/remesik/PG.htm. Prednášky z predmetu Počítačová grafika

[6] E. Ružický, A. Ferko,*Počítačová grafika a spracovanie obrazu.*, Sapientia, Bratislava 1995

[7] Miroslav Nekvinda, Jiři Šrubař, Jaroslav Vild,
Úvod do numerické matematiky, Praha 1976 SNTL-Nakladatelství technické literatúry.

[8] Shaharuddin Salleh,Albert Y. Zomaya, Sakhinah Abu Baka
Computing for numerical methods using VISUAL C++. A JOHAN WILEY & SONS, INC.,PUBLICATION 2007.

[9] `http://sk.wikipedia.org/wiki/C_Sharp`

[10] `http://www.riemers.net/eng/Tutorials/DirectX/Csharp/series1.php`

[11] DirectX – Pavel Pokorný. Začíname programovat'. Grada Publishing a.s.,
2008 www.grada.cz

[12] Jules Bloomenthal, *Polygonization of Implicit Surfaces*, May 1987.

[13] M. Čermák, *Zobrazování povrchu scén definovaných pomocí implicitních funkcí a CSG stromů*. Západočeská univerzita, Plzeň 2001

[14] Čermák Martin and Skala Václav, *Surface curvature estimation for edge spinning algorithm.*. In ICCS 2004, Poland, June 2004.

[15] Ondrej Jabornik. *Triangulácia implicitne definovanej ploch* 2006.

[16] [http://www.csharp-examples.net/string-format-int/]

[17] http://www.efunda.com/math/num_rootfinding/num_rootfinding.cfm Numerical Root Finding

Zoznam príloh

CD-ROM obsahujúci:

- Predkompilovanú verziu programov (vyžadujú .NET Framework 3.5 ,DirectX 9.0c a DirectX 9 kompatibilnú grafickú kartu)
- Zdrojové súbory našej implementácie.
- Súbory STL s výslednými triangulániami
- Elektronickú verziu textu diplomovej práce.