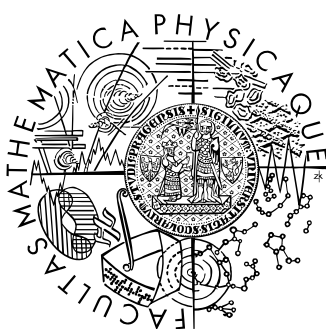


Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Ján Ilavský

Rekonštrukcie povrchov z neparalelných rezov s použitím interpoláčnych metód

Kabinet software a výuky informatiky

Vedoucí diplomové práce: Ing. Petr Felkel, Ph.D.
Katedra počítačů, FEL ČVUT Praha

Studijní program: Informatika, Softwarové systémy, Počítačová grafika

2008

Na tomto mieste by som sa chcel poďakovať hlavne môjmu vedúcemu diplomovej práce Ing. Petrovi Felklovi, Ph. D. za predstavenie problematiky neparalelných rezov, pomoc pri hľadaní riešenia problémov a za veľkú trpezlivosť. Takisto sa chcem poďakovať Vladimírovi Hrinčárovi za inšpiráciu pri dlhých diskusiách o využiteľnosti jednotlivých metód a za pomoc pri pochopení priamych vizualizačných metód. Ďalej moje veľké ďakujem patrí mojej priateľke Barbore Zbiňovskej za pomoc pri oprave gramatických chýb a podporu pri dňoch strávených takmer výlučne za počítačom. V neposlednom rade sa chcem poďakovať mojej rodine za neskutočnú podporu a trpezlivosť.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 5.8.2008

Ján Ilavský

Obsah

OBSAH	3
ZOZNAM OBRÁZKOV	5
ZOZNAM TABULIEK	7
ZOZNAM GRAFOV	8
1 ÚVOD	11
1.1 ZÍSKAVANIE VSTUPNÝCH DÁT	11
1.2 PRIAMA A NEPRIAMA REKONŠTRUKCIA	12
1.3 VÝSTUP REKONŠTRUKCIE	13
2 PRIAMA REKONŠTRUKCIA POVRCHOV Z KONTÚR	14
2.1 PARALELNÉ REZY	14
2.1.1 <i>Rekonštrukcia s použitím parametrickej domény</i>	15
2.2 NEPARALELNÉ REZY	15
2.2.1 <i>Algoritmus založený na Delaunayovej triangulácii</i>	15
3 NEPRIAME METÓDY REKONŠTRUKCIE POVRCHOV – VYTVÁRANIE DISTANCE FUNCTION	18
3.1 POSTUP VYTVÁRANIA DISTANCE FUNCTION	18
3.2 VZDIALENOSTNÁ TRANSFORMÁCIA	19
3.2.1 <i>Vytvorenie binárnej 2D mapy z množiny úsečiek</i>	19
3.2.2 <i>Chamferova transformácia</i>	21
3.3 INTERPOLÁCIA REZOV OBSAHUJÚCICH MAPU VZDIALENOSTÍ	24
3.3.1 <i>Interpolácia dvoch rezov</i>	24
3.4 INTERPOLÁCIA POMOCOU MAXIMÁLNYCH KRUŽNÍC (DISC GUIDED INTERPOLATION)	27
3.4.1 <i>Minimálna množina maximálnych kružníc</i>	28
3.4.2 <i>Výpočet regionálnej korešpondencie</i>	30
3.4.3 <i>Výpočet bodovej korešpondencie</i>	33
3.4.4 <i>Výpočet interpolačného vektora</i>	34
4 ZOBRAZOVANIE IZOPLOCHY	35
4.1 PRIAME ZOBRAZOVANIE IZOPLOCHY	35
4.1.1 <i>Interaktívne sledovanie lúča pre zobrazenie izoplochy</i>	35
4.1.2 <i>Prechádzanie buniek</i>	36
4.1.3 <i>Výpočet priesečníka s izoplochou v rámci bunky</i>	38
4.2 NEPRIAME ZOBRAZOVANIE IZOPLOCHY – PREVOD IZOPLOCHY NA MNOŽINU TROJUHLNÍKOV	40
4.2.1 <i>Marching Cubes – MC (Pochodujúce kocky)</i>	41
4.2.2 <i>Marching tetrahedra – MT (Pochodujúce štvorsteny)</i>	47
4.2.3 <i>Regularised marching tetrahedra – RMT (Regularizované pochodujúce štvorsteny)</i>	48
5 IMPLEMENTÁCIA	58
5.1 ŠTRUKTÚRA PROGRAMU	58
5.2 POMOCNÉ TRIEDY A FUNKCIE (TOOLS)	61
5.3 POSTUP VÝPOČTU REKONŠTRUKCIE	61
5.4 SPRACOVANIE VSTUPNÝCH DÁT	62
5.4.1 <i>Formát ASE súboru</i>	62
5.4.2 <i>Načítanie vstupného modelu (CModel)</i>	63
5.4.3 <i>Vytváranie rezov zo vstupného modelu (CCrossSections)</i>	64
5.5 REKONŠTRUKCIA POVRCHU	66
5.5.1 <i>Reconstructors</i>	66
5.5.2 <i>Interpolators</i>	68
5.5.3 <i>Visualisators</i>	69

5.6	CCELLMESH	71
5.7	UŽÍVATEĽSKÉ ROZHRAŇIE (WINDOWS, CVIEWS)	74
5.7.1	Hlavné okno aplikácie	74
5.7.2	Okno rezov	75
5.7.3	Dialógové okno nastavení rekonštrukcie	75
6	VÝSLEDKY	76
6.1	POROVNANIE MARCHING CUBES, MARCHING TETRAHEDRA A REGULARISED MARCHING TETRAHEDRA	76
6.2	REKONŠTRUKCIA Z PARALELNÝCH REZOV	78
6.3	REKONŠTRUKCIA Z NEPARALELNÝCH REZOV	83
6.3.1	Rekonštrukcia kocky	84
6.3.2	Rekonštrukcia spojených torusov	86
6.3.3	Rekonštrukcia modelu koňa	87
7	ZÁVER	91
7.1	DISKUSIA A PLÁNY DO BUDÚCNA	91
8	APENDIX A – OVLÁDANIE APLIKÁCIE RPZNR	93
8.1	NASTAVENIE PARAMETROV	93
9	APENDIX B – IMPLEMENTAČNÉ DÁTA A ALGORITMY	95
9.1	DÁTA POUŽITÉ V IMPLEMENTÁCII FUNKCIE GETPOINT	95
9.2	HODNOTY KONŠTANTNÝCH POLÍ EDGEREFERENCE A THEDGES	97
9.3	TEST NA ZHLUKOVANIE V ALGORITME RMT	99
10	APENDIX C – OBSAH DVD	101
11	POUŽITÁ LITERATÚRA	102

Zoznam obrázkov

2-1 PROBLÉM KOREŠPONDENCIE KONTÚR	14
2-2 PRÍKLADY PORUŠENIA PODMIENOK A) COMFORMITY CONDITION B) SOLIDITY CONDITION	16
2-3 POSTUP OREZÁVANIA V ALGORITME POPÍŠANOM V SEKCI 2.2.1	17
3-1 OBEČNÉ CHAMFEROVE MASKY.	19
3-2 PIXEL V POLYGÓNE.	20
3-3 HODNOTY PIXLOV PO PRECHODE RASTERIZAČNÝM ALGORITMOM.	21
3-4 CHAMFEROVE MASKY.	21
3-5 PRÍKLAD BITMAPY VYTVORENEJ ALGORITMOM CHAMFER 3X3.	23
3-6 INTERPOLÁCIA HODNÔT NAJBĽIŽŠÍCH SUSEDOV.	25
3-7 ARTEFAKTY VZNIKAJÚCE PRI MALOM UHLE PRI POUŽITÍ ALGORITMU NAJBĽIŽŠIEHO BODU.	26
3-8 POZÍCIE BODOV A A B PRI INTERPOLÁCII S PRIEMERNOU NORMÁLOU.....	26
3-9 REKONŠTRUKCIA PRI MALOM UHLE REZNÝCH ROVÍN POMOCOU PRIEMERNEJ NORMÁLY.....	27
3-10 NESCHOPNOŠT ADAPTÁCIE SA INTERPOLÁCIE POMOCOU PRIEMERNEJ NORMÁLY PRI VEĽMI VZDIALENÝCH REZOCH	27
3-11 MNOŽINA MAXIMÁLNYCH KRUŽNÍČ PO ODSTRÁNENÍ VŠETKÝCH, KTORÉ NESPLŇAJÚ ROVNOSŤ 3-6	29
3-12 ZVOLENIE PRIEMERNEJ ROVINY, V KTOREJ SA POČÍTA REGIONÁLNA KOREŠPONDENCIA. $\alpha < \beta$	30
3-13 ZOBRAZENIE VEKTOROV ca PRI REKONŠTRUKCII MODELU KOŇA.....	32
3-14 VÝPOČET INTERPOLAČNÉHO VEKTORA Z VEKTORA cp	34
4-1 PRIESEČNÍK LÚČA S IZOPLOCHOU PRI PRIAMEJ ZOBRAZOVACEJ METÓDE.....	35
4-2 PRECHÁDZANIE LÚČA MRIEŽKOU BUNIEK PRI METÓDE SLEDOVANIA LÚČA	36
4-3 ZOBRAZENIE DVOJSTUPŇOVEJ HIERARCHIE V DVOJROZMERNOM ANALOGICKOM PRÍPADE URÝCHĽOVANIA ALGORITMU PREHĽADÁVANIA MRIEŽKY	37
4-4 APROXIMÁCIA OBVODU TELES V 2D PRIESTORE POMOCOU ALGORITMU MARCHING SQUARES.	41
4-5 MOŽNOSTI APROXIMÁCIE FUNKCIE VRÁMCI ŠTVORCA (MARCHING SQUARES).....	41
4-6 MOŽNOSTI TRIANGULÁCIE RÔZNYCH SITUÁCIÍ V MARCHING CUBES.....	43
4-7 NESPRÁVNE DETEKOVANIE POVRCHU PRI POUŽITÍ MARCHING CUBES	45
4-8 ŠPECIÁLNY PRÍPAD ZLEJ NÁVÄZNOSTI PRI POUŽITÍ MARCHING CUBES.....	45
4-9 TOPOLOGICKY INKONZISTENTNÉ NAVIAZANIE DVOCH TRIANGULÁCIÍ PRE PRÍPAD 3 A 6.	46
4-10 TOPOLOGICKY KONZISTENTNÉ RIEŠENIE PRÍPADU Z OBRÁZKU 4-9, S POUŽITÍM ALTERNATÍVNEJ TRIANGULÁCIE PRÍPADU 6	46
4-11 OBDVE MOŽNOSTI TRIANGULÁCIE ŠTVORSTENOV.....	47
4-12 VRCHOLY ŠTVORSTENOV PRI POUŽITÍ TZV. CENTER CUBIC LATTICE	47
4-13 ROZLOŽENIE BODOV VZORKOVACEJ MRIEŽKY V PRÍPADE POUŽITIA BODY-CENTERED CUBIC LATTICE	49
4-14 ČÍSLOVANIE VRCHOLOV POUŽITÝCH V IMPLEMENTÁCII ALGORITMU MARCHING TETRAHEDRA.....	50
4-15 ZHLUKOVANIE VRCHOLOV (REGULARISED MARCHING TETRAHEDRA)	51
4-16 KRITÉRIUM PRE OTVORENÝ POVRCH	52
4-17 PRÍPADY RÔZNYCH TOPOLOGIÍ VYSKYTUJÚCICH SA PRI ZJEDNODUŠOVANÍ POVRCHU	52
4-18 DVA DRUHY RÔZNYCH HRÁN VO VZORKOVACEJ MRIEŽKE	54
4-19 VÝPOČET GRADIENTU A NORMÁLY V RÁMCI JEDNEJ PLOCHY	54
4-20 VÝPOČET ZAKRIVENIA IZOPLOCHY	56
4-21 ADAPTOVANIE VZORKOVACEJ MRIEŽKY PRE NEPARALELNÉ REZY	57
5-1 ŠTRUKTÚRA PROGRAMU	60
5-2 POSTUP VÝPOČTU REKONŠTRUKCIE.....	61
5-3 DÁTOVÁ ŠTRUKTÚRA SMESH	64
5-4 NAJHORŠÍ MOŽNÝ PRÍPAD ROZLOŽENIA ÚSEČIEK KONTÚRY REZU.	65
5-5 ZÁVISLOSTI TRIED REKONŠTRUOVANÝCH POVRCH	66
5-6 PREMENNÉ POUŽITÉ PRI ZHLUKOVANÍ VRCHOLOV.....	71
5-7 DISKRÉTNÉ MERANIE OBJEMU POMOCOU VRHANIA LÚČOV. POHĽAD NA REZ SCÉNY CEZ OSU X.	73
5-8 VÝPOČET OBJEMU ROZDIELU DVOCH TELIES.....	74
6-1 REKONŠTRUOVANÉ TELESO ZOBRAZENÉ RÔZNYMI VIZUALIZAČNÝMI METÓDAMI.....	77
6-2 REKONŠTRUKCIA KOCKY POMOCOU PARALELNÝCH REZOV.	78
6-3 REKONŠTRUKCIA KOCKY POMOCOU PARALELNÝCH REZOV NEROVNOBEŽNÝCH Z PODSTAVOU KOCKY.	79
6-4 REKONŠTRUKCIA KOCKY POMOCOU CLOSEST POINT INTERPOLÁCIE PRE RÔZNY POČET REZOV.	79
6-5 REKONŠTRUKCIA KOCKY POMOCOU DISC GUIDED INTERPOLÁCIE PRE RÔZNY POČET REZOV.	80

Zoznam obrázkov

6-6 REKONŠTRUKCIA CYLINDRA PRI POUŽITÍ INTERPOLÁCIE CLOSEST POINT A DISC GUIDED Z RÔZNEHO POČTU REZOV.	82
6-7 REKONŠTRUKCIA KOŇA S POUŽITÍM PARALELNÝCH REZOV A DISC GUIDED INTERPOLÁCIE.....	83
6-8 MODEL GULE ROZREZANÝ POMOCOU VEJÁROVÝCH REZOV	84
6-9 POROVNANIE VÝSLEDKU REKONŠTRUKCIE KOCKY Z NEPARALELNÝCH REZOV POMOCOU DVOCH METÓD.....	84
6-10 PROBLÉM VEĽKEJ CHYBY PRI MALOM POČTE REZOV PRECHÁDZAJÚCICH OBJEMNEJŠÍM TORUSOM.	86
6-11 VEJÁROVÉ REZY MODELU KOŇA.	89
6-12 REKONŠTRUKCIA KOŇA Z VEJÁROVÝCH REZOV S POUŽITÍM DISC GUIDED INTERPOLÁCIE.	89
6-13 REKONŠTRUKCIA KOŇA Z VEJÁROVÝCH REZOV S POUŽITÍM AVERAGE NORMAL INTERPOLÁCIE.	90
8-1 DIALÓGOVÉ OKNO RECONSTRUCTION	93
8-2 TLAČIDLÁ RIADIACE REKONŠTRUKCIU.	94
9-1 VYUŽITIE KONŠTÁNT EDGEREFERENCE A THEEDGES.....	97

Zoznam tabuliek

1-1 PREHĽAD TOMOGRAFOV	12
6-1 POČET TROJUHLNÍKOV VYGENEROVANÝCH PRI POUŽITÍ RÔZNYCH METÓD.	76
6-2 HODNOTY OBJEMOV PRE REKONŠTRUOVANÚ KOCKU NEPARALELNÝMI REZMI.	85
6-3 HODNOTY OBJEMOV PRE REKONŠTRUKCIU ZLOŽENÝCH TORUSOV NEPARALELNÝMI REZMI.	86
6-4 HODNOTY OBJEMOV PRE REKONŠTRUOVANÉHO KOŇA NEPARALELNÝMI REZMI.	88
8-1 PARAMETRE VYTVÁRANIA REZOV.	94

Zoznam grafov

6-1 OBJEMY ROZDIELOV MEDZI ZDROJOVÝM A REKONŠTRUOVANÝM TELESOM V ZÁVISLOSTI OD POČTU REZOV PRI REKONŠTRUKCII KOCKY S POUŽITÍM PARALELNÝCH REZOV.	81
6-2 OBJEMY ROZDIELOV MEDZI ZDROJOVÝM A REKONŠTRUOVANÝM TELESOM V ZÁVISLOSTI OD POČTU REZOV PRI REKONŠTRUKCII SPOJENÝCH TORUSOV S POUŽITÍM PARALELNÝCH REZOV.	81
6-3 OBJEMY ROZDIELOV MEDZI ZDROJOVÝM A REKONŠTRUOVANÝM TELESOM V ZÁVISLOSTI OD POČTU REZOV PRI REKONŠTRUKCII CYLINDRA S POUŽITÍM PARALELNÝCH REZOV.	82
6-4 OBJEMY ROZDIELOV MEDZI ZDROJOVÝM A REKONŠTRUOVANÝM TELESOM V ZÁVISLOSTI OD POČTU REZOV PRI REKONŠTRUKCII KOCKY S POUŽITÍM NEPARALELNÝCH VEJÁROVÝCH REZOV.	85
6-5 OBJEMY ROZDIELOV MEDZI ZDROJOVÝM A REKONŠTRUOVANÝM TELESOM V ZÁVISLOSTI OD POČTU REZOV PRI REKONŠTRUKCII ZLOŽENÝCH TORUSOV S POUŽITÍM NEPARALELNÝCH VEJÁROVÝCH REZOV.	87
6-6 OBJEMY ROZDIELOV MEDZI ZDROJOVÝM A REKONŠTRUOVANÝM TELESOM V ZÁVISLOSTI OD POČTU REZOV PRI REKONŠTRUKCII MODELU KOŇA S POUŽITÍM NEPARALELNÝCH VEJÁROVÝCH REZOV.	88

Název práce: Rekonštrukcie povrchov z neparalelných rezov s použitím interpolačných metód

Autor: Ján Ilavský

Katedra (ústav): Kabinet software a výuky informatiky

Vedoucí diplomové práce: Ing. Petr Felkel, Ph.D. Katedra počítačů, FEL ČVUT Praha

e-mail vedoucího: felkel@fel.cvut.cz

Abstrakt: Rekonštrukcia povrchu z neparalelných rezov je málo skúmaným problémom. Táto práca poskytuje prehľad metód používaných pri jeho riešení. Postupne sú popísané všetky kroky od počiatočného získavania dát až po vizualizáciu rekonštruovaného povrchu. Ťažisko je kladené na metódy vytvárajúce skalárne pole (distance function) pomocou interpolačných metód. Podrobne sú popísané metódy closest point, average normal a disc-guided interpolácie pričom sú určené ich slabé a silné miesta. Z vizualizačných techník sú spomenuté základné princípy fungovania priamych metód a do hĺbky popísané nepriame metódy-pochodujúce kocky, pochodujúce štvorsteny a regularizované pochodujúce štvorsteny. Pomocou programu, kde sú všetky metódy implementované, sú v kapitole s výsledkami porovnané ich výstupy pri rekonštrukcii modelov uložených vo formáte ASE. Takýto model je v programe najprv „rozrezaný“ na množinu rezov. Z týchto je pomocou jednej z rekonštrukčných metód vytvorená distance function, ktorá je následne zobrazená jednou z vizualizačných metód. Porovnávanie je založené na meraní objemu rozdielov medzi pôvodným a rekonštruovaným telesom pomocou metódy vrhania lúča a subjektívnym vizuálnym vnemom.

Klíčová slova: rekonštrukcia, neparalelné rezy, disc-guided interpolácia, regularizované pochodujúce štvorsteny

Title: Surface Reconstruction from Non-parallel Cross-sections Using Interpolation Methods

Author: Ján Ilavský

Department: Department of Software and Computer Science Education

Supervisor: : Ing. Petr Felkel, Ph.D. Katedra počítačů, FEL ČVUT Praha

Supervisor's e-mail address: felkel@fel.cvut.cz

Abstract: Reconstruction from non-parallel cross-sections is not very well studied problem. This work provides overview of methods used to solve it. Step by step there are described all the algorithms from obtaining input data to visualize reconstructed surface. Main focus is put on methods which create distance function interpolating data from cross-sections. The closest point, average normal and disc-guided interpolation are described in details. Out of visualization techniques there are mentioned straight methods and defined in detail marching cubes, marching tetrahedra and regularized marching tetrahedra. All this methods are implemented and used in windows application to compare their results in reconstruction of models saved in ASE format. First of all this model is cut into several cross-sections inside this application. These are used to create distance function using one of the interpolation methods. At the end, distance function is visualized. To compare results, there is used volume computing algorithm based on ray-casting and subjective user opinion based on graphical output.

Keywords: reconstruction, non-parallel cross-sections, disc-guided interpolation, regularised marching tetrahedra

1 Úvod

Rekonštrukciou povrchu sa myslí postup, pri ktorom na základe čiastočných informácií o skúmanom telese dokážeme zobraziť alebo vhodne reprezentovať približný povrch pôvodného telesa. Pri rekonštrukcii z rezov sú takýmito dátami roviny obsahujúce kontúry telesa. S príchodom technológií umožňujúcich tieto dáta získavať, vznikol aj praktický dôvod pre vytváranie rekonštrukčných algoritmov. V posledných desiatich rokoch bolo navrhnutých veľa metód. Rekonštruovanie z paralelných rezov bolo veľmi dobre preskúmané a pri dostatočnej hustote rezov poskytujú súčasné metódy výborné výsledky. Horšie to je s rezmi neparalelnými. Ich využitie nie je také vysoké ako v opačnom prípade a algoritmy môžu byť značne komplikovanejšie. Táto práca sa zaoberá hlavne touto skupinou.

Jednotlivé metódy na rekonštrukciu sa môžu líšiť podľa druhu vstupných dát, podľa postupu ich spracovania, prípadne podľa spôsobu ich zobrazenia. Neexistuje žiadna univerzálna metóda schopná rovnako dobre spracovať obecné rezy. Algoritmy sú častokrát prispôbené konkrétnym situáciám, v ktorých dávajú najlepšie výsledky. V nasledujúcich sekciách sú najprv popísané možnosti, problémy a obmedzenia pri získavaní vstupných dát, výhody a nevýhody jednotlivých metód. Ďalej je načrtnutý úvod do dvoch hlavných skupín rekonštrukcie – priamej a nepriamej. Stručne je popísaná história ich vzniku, rozdiely v ich fungovaní a súčasné trendy. V poslednej časti tejto kapitoly sú rozobrané možnosti reprezentácie a zobrazenia výsledného povrchu.

1.1 Získavanie vstupných dát

Vstupnými dátami sú v našom prípade rezy telesa, ktoré obsahujú informácie o tvare jeho povrchu, prípadne vnútornej štruktúre. V praxi existuje viacero spôsobov ako takéto dáta získať. Môžeme ich rozdeliť do dvoch hlavných skupín: invazívne a neinvazívne.

Invazívne metódy, ako už názov napovedá, získavajú rezy na základe mechanickej deštrukcie skúmaného telesa. Takýto postup je nutné zvoliť vtedy, ak nie je možné skúmať vnútornú štruktúru inak, napríklad kvôli veľkej hustote telesa, prípadne nedostačujúcemu rozlíšeniu dát získaných iným spôsobom. Najjednoduchšou metódou je v tomto prípade postupné mechanické rozrezanie skúmaného objektu na jednotlivé rezy, ktoré sú následne ofotografované. Kontúry je potom možné získať pomocou segmentácie fotografií. Takýmto spôsobom boli napríklad spracované dve ľudské telá v The Visible Human Project [1]. Výstupom invazívnych metód sú spravidla paralelné rezy.

Aj keď môžeme pomocou predchádzajúcej metódy získať veľmi kvalitné dáta, vo väčšine prípadov je skúmaným objektom živý organizmus a fyzický zásah daných rozmerov preto nie je možné vykonať. V takomto prípade môžeme využiť tomografiu. Tomografia pod sebou zahŕňa postupy, ktoré pomocou známych fyzikálnych princípov dokážu získavať obrázky rezov. Prístroje nato určené sa nazývajú tomografy a výsledné snímky tomogramy. Tomografy je možné rozdeliť podľa fyzikálnych javov, ktoré využívajú (viď. tabuľka 1-1).

Fyzikálny jav	Typ tomografu
Röntgenové žiarenie	Počítačová tomografia (CT)
Gamma žiarenie	Jednofotónová emisná tomografia (SPECT)
Elektron-pozitrónová anihilácia	Pozitrónová emisná tomografia (PET)
Nukleárna magnetická rezonancia	Zobrazovanie magnetickou rezonanciou (MRI)
Ultrazvuk	Sonograf

Tabuľka
1-1 Prehľad tomografov

Výsledkom použitia niektorého z tomografov je snímok, prípadne množina snímkov, predstavujúca rezy, na ktorých možno rozoznať oblasti s rôznou hustotou. Takýmto spôsobom je napríklad možné zachytiť orgány v tele človeka. Rekonštrukcia povrchu z neparalelných rezov má najväčšie využitie pre ultrazvuk. Ostatné tomografy dokážu generovať snímky v presne definovaných rovinách a množina hustých paralelných rezov je pre rekonštrukciu veľmi vhodná. Naproti tomu ručným ultrazvukom takmer nie je možné paralelné rezy získať. Nevýhoda v nemožnosti jednoducho generovať rovnobežné roviny je vyvážená viacerými faktormi. Ultrazvuk je mnohonásobne lacnejší a je možné používať ho v každej ambulancii, nie sú s ním spojené ani žiadne vedľajšie účinky na organizmus.

1.2 Priama a nepriama rekonštrukcia

Algoritmy rekonštruujúce povrch je možné rozdeliť na dve základné skupiny. V prvej z nich sa nachádzajú metódy, ktoré vytvárajú trojuholníkovú sieť priamo z jednotlivých rezov. V takomto prípade sú jednotlivé trojuholníky „nalepované“ na okraje kontúr v rovinách rezov. Pretože ide o diskretný problém, je nutné riešiť veľa hraničných situácií a celá úloha je veľmi komplikovaná. Podrobnejšie sa tejto téme venuje kapitola 2. Do druhej kategórie patria metódy vytvárajúce takzvanú distance function. Ide o 3D zobrazenie, ktoré každému bodu v priestore priradí vzdialenosť od povrchu rekonštruovaného telesa. Takéto zobrazenia sú takmer výhradne produktom interpolácií dát (sekcia 3.3) z transformovaných kontúr (sekcia 3.2) v rovinách rezov. Toto 3D zobrazenie je následne možné zobrazíť s použitím niektorého z vizualizačných algoritmov ako izoplochu s hladinou nula.

Veľká výhoda nepriamych metód spočíva v komplexnom riešení celej množiny problémov, ktoré sa pri rekonštrukcii vyskytujú. Všetka pozornosť je venovaná najmä smeru a druhu interpolácie. Najjednoduchšie metódy interpolujú lineárne najbližšie body z najbližších rezových rovín, pričom najkomplikovanejšie sofistikovane určujú smer interpolácie, ktorý sa môže meniť v každom bode priestoru a zároveň je možné interpolovať z viacerých rovín súčasne. Výhodou je aj výsledné 3D zobrazenie, ktoré je možné zobrazíť veľmi presne. V komplexnom riešení množiny problémov v tomto prípade spočíva aj určitá nevýhoda nepriamych metód. Neexistuje totiž žiaden jednoduchý spôsob, ako do rekonštrukcie vložiť logiku rozhodovania sa rekonštruovať povrch v určitých prípadoch iným spôsobom. Je možné meniť zvolený druh interpolácie, ale výsledok je stále plne pod jej kontrolou.

Naproti tomu priame metódy musia riešiť všetky problémy systematicky. Jediným výstupom je v ich prípade trojuholníková sieť. Veľmi ťažko sa pritom dá ovplyvňovať jej regularita.

1.3 Výstup rekonštrukcie

Výstupom rekonštrukčných metód sú takmer výlučne dáta určené na zobrazovanie v trojrozmernom prostredí. Veľký dôraz sa častokrát kladie na možnosť otáčania a približovania náhľadu na takéto dáta. Kvôli tomu bolo v minulosti na tieto účely možné použiť výhradne trojuholníkovú sieť. Takouto sieťou sa myslí množina susediacich trojuholníkov, ktoré sa dotýkajú hranami a ako celok reprezentujú povrch rekonštruovaného telesa. Najväčšou výhodou týchto sietí je možnosť ich rýchleho zobrazovania a pri vhodnej hustote trojuholníkov aj relatívne malá pamäťová náročnosť. Hlavnou nevýhodou je problematické vytváranie sietí, v ktorých sa trojuholníky navzájom len veľmi málo líšia. Pri veľkých rozdieloch vznikajú zobrazovaním využívajúcim Guoradovo tieňovanie nepríjemné grafické artefakty. Výstup v podobe trojuholníkovej siete generujú všetky priame metódy (kapitola 2). Naproti tomu nepriame metódy, ktorých medzikrokom je spomínaná distance function, môžu byť zobrazené nielen pomocou transformácie na trojuholníkovú sieť (sekcia 4.2), ale aj iným spôsobom. V takomto prípade sa jedná o priame zobrazenie izoplochy definovaného v 3D zobrazení. Distance function je v prvom kroku diskretizovaná a prevedená do trojrozmerného poľa, ktoré je možné zobrazovať v reálnom čase pomocou metód ray-tracingu (sekcia 4.1).

2 Priama rekonštrukcia povrchov z kontúr

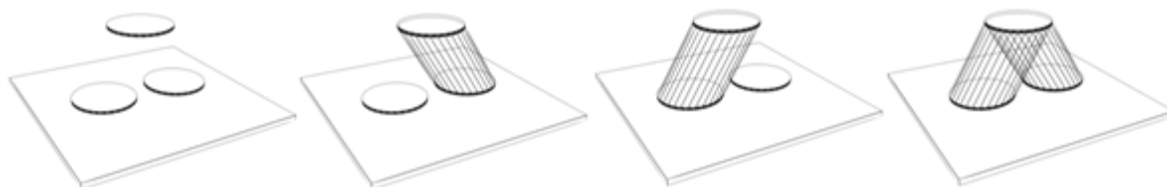
Ako už bolo spomenuté v predchádzajúcej kapitole, priame metódy rekonštrukcie povrchov sa snažia vytvoriť trojuholníkovú sieť bez použitia 3D zobrazenia (distance function). Algoritmy analyzujú jednotlivé rezy a z ich susedných dvojíc sa snažia zistiť pravdepodobný povrch telesa. Takéto rezy môžu byť od seba v praxi značne vzdialené, čo komplikuje zložitosť rozhodovania sa ako povrch rekonštruovať. Nasledujúce sekcie stručne definujú problémy, s ktorými sa potýkajú paralelné rezy a budú aj jemne načrtnuté algoritmy riešiace priamou metódou rekonštrukciu povrchov z rezov neparalelných.

2.1 Paralelné rezy

Veľkou výhodou paralelných rezov, je jednoduchosť ich usporiadania a vzájomná orientácia. Napriek tomu pri veľkej vzdialenosti a rozdielnosti susedných kontúr nie je jednoduché získať z vstupných dát povrch telesa triviálnym spôsobom. Prvé algoritmy zaoberajúce sa rekonštrukciou povrchu [2] rozdelili celý postup na tri hlavné problémy:

Correspondence problém

Tento problém vzniká vždy, keď existuje viacero uzavretých kontúr v rámci jedného rezu (Obrázok 2-1). V takomto prípade je nutné rozhodnúť, ktoré dva, prípadne viaceré kontúry z dvoch susedných rezov tvoria jeden objekt, a teda budú spojené. Všeobecne môže nastať situácia, kedy nie je možné tento problém vyriešiť automaticky, prípadne neexistuje žiadne "správne" riešenie, ktoré je možné získať len z dvoch susedných rezov. V takomto prípade je možné rozšíriť algoritmus tak, aby sa ich pokúsil zohľadniť viacero. Druhou možnosťou je poloautomatické riešenie, v ktorom má možnosť príslušnosť jednotlivých kontúr ohodnotiť človek. Väčšina plne automatických algoritmov riešiacich correspondence problém predpokladá malú vzdialenosť medzi jednotlivými rezmi.



Obrázok
2-1 Problém korešpondencie kontúr

Zobrazenie situácie, v ktorej nie je možné jednoznačne určiť, akým spôsobom má byť vedená triangulácia.

Tiling problém

Ak je vyriešený problém korešpondencie jednotlivých kontúr, možno medzi nimi začať vytvárať trojuholníkovú sieť. Tiling problém rieši práve akým spôsobom majú byť kontúry susedných rezov napojené. Trojuholníky vytvorené v tomto kroku majú dve hrany tvorené jedným vrcholom z kontúry z jedného rezu a jedným vrcholom z rezu susedného. Tieto hrany sú spoločné s hranami susedných trojuholníkov v trojuholníkovej sieti. Tretia hrana vždy leží na jednom z dvoch susedných rezov.

Branching problém

Potreba riešiť branching problém nastáva v okamihu, keď je nutné triangulovať vetviace sa rezy. Teda počet kontúr rekonštruovaného telesa sa v dvoch susedných rezoch líši. V takomto prípade nie je možné na seba jednoducho dve kontúry na seba "nalepiť" a je potrebné vytvoriť vetvenie.

2.1.1 Rekonštrukcia s použitím parametrickej domény

Väčšina priamych rekonštrukčných algoritmov sa snaží vyriešiť hlavne tieto tri kľúčové časti. V súčasnosti však do popredia vystupujú algoritmy, ktoré riešia všetky tri problémy systematicky v jednom kroku. Pre jednu skupinu takýchto algoritmov je charakteristické vytváranie povrchu z parametrickej domény, ktorá je vypočítaná z prieniku kontúr dvoch prekrývajúcich sa rezov. Vo väčšine postupov je táto doména prevedená na trojuholníky, ktoré sú následne mapované do trojrozmerného priestoru. Jedným z takýchto algoritmov je aj algoritmus popísaný v [3].

2.2 Neparalelné rezy

Problém rekonštrukcie povrchov z paralelných rezov je v súčasnosti už veľmi dobre riešený. Existujú metódy, ktoré dokážu pracovať nad ľubovoľnou množinou dát a pritom dávajú uspokojivé výsledky. Použitie ručného ultrazvuku však priamo vyžaduje preskúmať možnosti rekonštruovať povrch z rezov, ktoré môžu byť voči sebe ľubovoľne orientované.

Problém rekonštrukcie z neparalelných rezov s použitím priamych metód bol doposiaľ málo skúmaný. Jediným známym riešením, ktoré pracuje nad ľubovoľnou množinou dát je riešenie Boissonnata a Memariho [4]. Pred týmto riešením sa podobný problémom zaoberali iba Payne a Toga [5], ktorých metóda však pracovala len s jednoduchými útvarmi a nedovoľovala vetvenie.

2.2.1 Algoritmus založený na Delaunayovej triangulácii

Metóda popísaná v (4) používa na rekonštrukciu povrchu Delaunayovu trianguláciu, ktorá bola podobným spôsobom využitá už v algoritme použitom v [3]. Celý postup je rozdelený do troch krokov. V prvom sú vypočítané konvexné bunky tvorené jednotlivými rezmi. V druhom kroku sa v rámci každej takejto bunky rekonštruuje povrch na základe

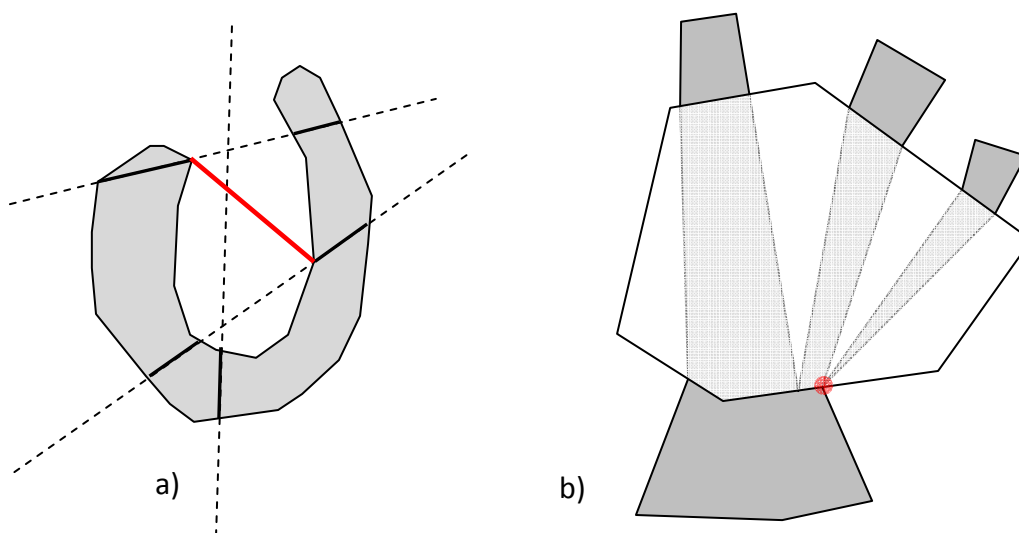
priesečníkov pôvodného objektu so stenami buniek. V poslednom ostáva jednotlivé bunky k sebe pospájať. Keďže prvá úloha je štandardom v počítačovej grafike a posledná je jednoduchá, hlavným ťažiskom je rekonštrukcia povrchu v rámci jednotlivých buniek.

Najdôležitejšou časťou tejto metódy je definovanie miesta vetvenia, teda množiny bodov, v ktorých sa mení topológia ako postupne prechádzame jednotlivé rezy. V tomto prípade je takáto množina bodov charakterizovaná v stenách jednotlivých buniek ako podmnožina Moebiovho diagramu, čo je v podstate rozšírenie Euklidovského Voroniovho diagramu.

Algoritmus vykoná Delaunayovu trianguláciu nad množinou, ktorú tvoria body kontúr rezu v danej bunke a vetviace body. Triangulácia je následne orezaná tak, aby bol výsledkom trojrozmerný ohraničený útvar nepretínajúci rezy, ktoré bunku definujú.

Výsledný objekt je definovaný pomocou úsečiek, ktoré spájajú jednotlivé kontúry. Tieto úsečky musia spĺňať dva hlavné podmienky (obrázok 2-1):

1. **Comformity condition** - žiadna úsečka nesmie pretínať rez mimo hranice kontúr
2. **Solidity condition** - výsledný objekt musí byť ohraničený trojrozmerný útvar (trojuholníková sieť musí obkolesovať neprázdny priestor a nesmú sa v ňom vyskytovať samostatné dvoj alebo jednorozmerné útvary)



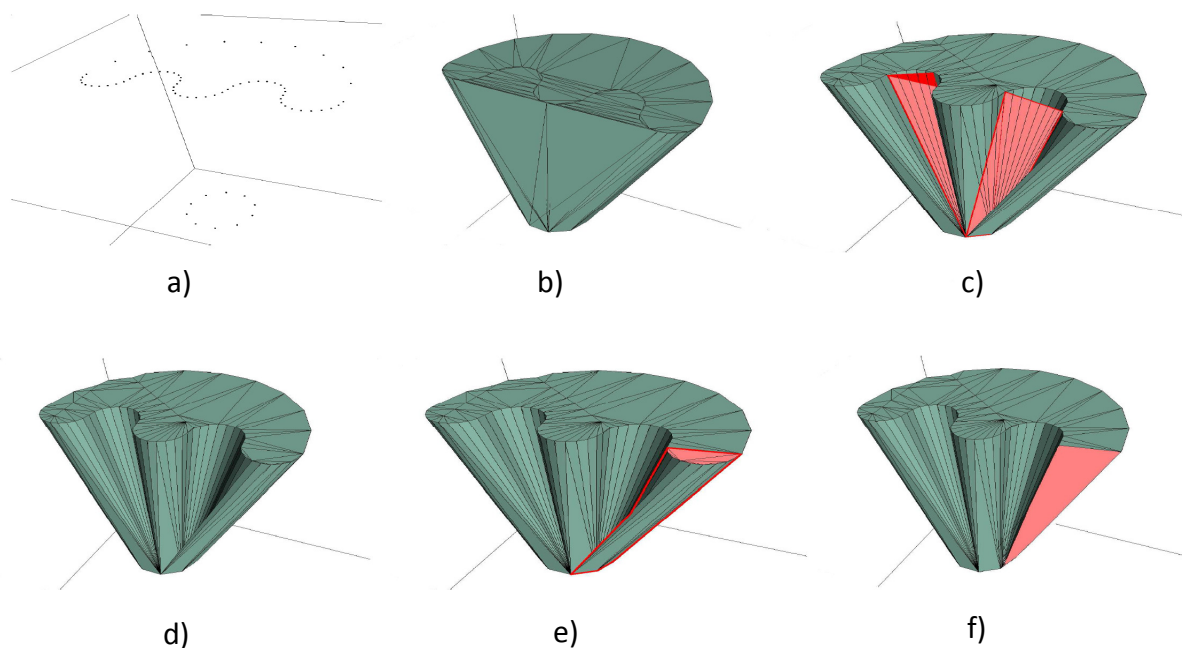
Obrázok

2-2 Príklady porušenia podmienok a) comformity condition b) solidity condition

Na obrázkoch sú znázornené dvojrozmerné analogické prípady porušenia podmienok definovaných v 2.2.1. Na obrázku a) je červenou čiarou vyznačená úsečka spájajúca dve kontúry, pričom pretína rez. Červený bod na obrázku b) ukazuje bodovú singularitu, ktorá nespĺňa solidity condition.

Prvým krokom algoritmu je vypočítanie takzvaného branching a cutting diagramu. Následne sú tieto dve množiny využité na výpočet Delaunayovej triangulácie. Takto vytvorená triangulácia však nespĺňa definované podmienky, a preto je nutné zmenšiť ju. Zmenšovanie je vykonané v troch krokoch, ktoré sú znázornené a popísané na obrázku 2-3.

Výsledkom sú správne triangulované bunky, ktorých spojením vznikne celý rekonštruovaný povrch.



Obrázok
2-3 Postup orezávania v algoritme popísanom v sekcii 2.2.1

a) vstupné body b) počiatočná Delaunayova triangulácia c) odstraňovanie vonkajších štvorstenov, niektoré sú zobrazené červenou farbou d) výsledok predchádzajúceho kroku, ktorý obsahuje singularitu porušujúcu solidity condition e) červeným zobrazené štvorsteny odstraňované v druhom kroku f) výsledná triangulácia

3 Nepriame metódy rekonštrukcie povrchov – vytváranie distance function

Priama rekonštrukcia povrchov z rezov je veľmi komplikovaná. Je potrebné riešiť veľké množstvo problematických situácií (branching, tiling a correspondance problém). Navyše takmer nie je možné ovplyvňovať regularitu vytváraných trojuholníkov, čo vedie k nekvalitnému zobrazeniu (hlavne pri použití Gouraudovho tieňovania). Práve tieto faktory dali podnet k vzniku nepriamych metód, ktoré sa nepokúšajú tvoriť trojuholníkovú sieť, ale 3D zobrazenie – distance function (DF), ktoré v každom bode priestoru udáva akúsi vzdialenosť od povrchu telesa. Prvýkrát s týmto návrhom prišiel Levin pri interpolácii dát z počítačovej tomografie [6]. DF by malo byť spojité zobrazenie, ktoré nadobúda na povrchu telesa hodnotu 0, vo vnútri záporné a vonku kladné hodnoty (v praxi môžu byť hodnoty opačné). Takéto zobrazenie s prahom 0 potom definuje izoplochu, ktorú je možné rýchlo zobrazit s použitím viacerých techník, ktoré sú popísané v kapitole 4.

3.1 Postup vytvárania distance function

DF má vyjadrovať akúsi vzdialenosť¹ každého bodu od povrchu telesa. Preto je potrebné nejakým spôsobom interpolovať vzdialenosti, ktoré možno získať z jednotlivých rezov. So základným postupom prišli Raya a Udupa a nazvali ho shaped-based interpolation [7]. Prvým krokom je v tomto prípade spočítanie vzdialenosti každého bodu z roviny rezov od kontúr, definovaných úsečkami. Tieto boli v tomto prípade vykreslené do dvojrozmernej bitmapy a vzdialenosť bola vypočítaná pomocou algoritmu mestských blokov. Následne mohli byť jednotlivé hodnoty medzi rezmi interpolované lineárnou, prípadne kubickou interpoláciou.

Všeobecne možno rozdeliť postup vytvárania DF na niekoľko krokov. Predpokladajme, že chceme zistiť hodnotu DF v bode P . Túto hodnotu označíme $f(P)$.

Prvým krokom je získanie dvojice, prípadne množiny rezov, ktoré bod P obklopujú. V prípade paralelných alebo mierne neparalelných² rezov stačí zistiť, ktoré dve roviny rezov sú k bodu P najbližšie. Vo všeobecnom prípade môže byť situácia oveľa komplikovanejšia, pričom v extrémnom prípade je možné získať celé bunky tvorené rovinami rezov, podobne ako v prípade algoritmu popísaného v sekcii 2.2.1.

Druhý krok spočíva v transformácii vstupných dát, teda množiny úsečiek uložených v jednotlivých rovinách, na dáta vhodnejšie pre interpoláciu (tzv. vzdialenostná transformácia). Tento výpočet je úplne nezávislý od prechádzajúceho kroku a popisuje ho sekcia 3.2.

¹ V skutočnosti sa nejedná o presne definovanú vzdialenosť. Pre účely rekonštrukcie stačí ak hodnota zobrazenia spojite rastie spolu so vzdialenosťou od povrchu.

² Každý bod roviny musí byť obklopený maximálne dvoma rovinami rezov.

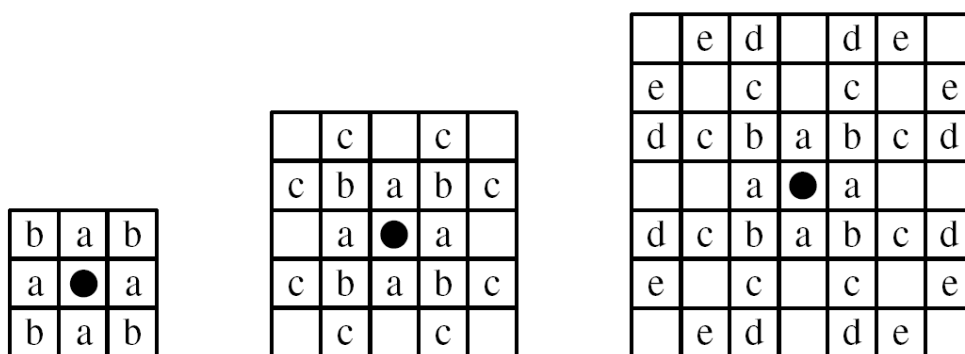
Sofistikovanejšie metódy môžu ďalej transformovať dáta z bitmáp na množinu geometrických primitív [8]. Existujú však aj algoritmy, ktoré vzdialenostnú transformáciu nepoužívajú, ale priamo charakterizujú jednotlivé časti rezov pomocou parametrov ako obsah plochy, ťažisko, rotačný uhol a ďalšie [9].

Posledným krokom je samotná interpolácia (sekcie 3.3 a 3.4) hodnôt zo vzdialenostnej transformácie z rovín získaných v prvom kroku.

V prípade, že v druhom kroku nebola vykonaná vzdialenostná transformácia, na vytváranie DF sa miesto interpolácie využíva morfung binárnych máp kombinovaný s ďalšími technikami.

3.2 Vzdialenostná transformácia

Vhodnou dátovou štruktúrou pre interpoláciu medzi rezmi v priestore je dvojrozmerná bitmapa, ktorá v každom svojom bode určuje vzdialenosť od povrchu telesa v rámci roviny rezu. Existuje viacero algoritmov schopných túto transformáciu vykonať. V praxi nie je potrebné vyžadovať, aby výsledkom bola Euklidovská vzdialenosť a pre prípad rekonštrukcie povrchov stačí využiť algoritmus distribuujúci vzdialenosť pomocou Chamferovej masky veľkosti 3x3. Na zložitejšie prípady sa používal dlho Chamfer s maskou 5x5, prípadne 7x7 (viď. obrázok 3-1), v roku 2004 publikoval Bailey [10] algoritmus schopný vypočítať v každom bode bitmapy Euklidovskú vzdialenosť v lepšom čase ako Chamfer 5x5.



Obrázok

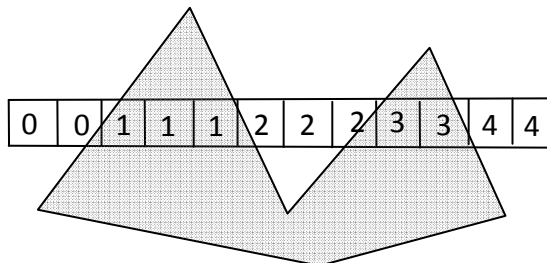
3-1 Obecné Chamferove masky.

Umiestnenie konštánt používaných v Chamferových maskách.

3.2.1 Vytvorenie binárnej 2D mapy z množiny úsečiek

Ešte predtým ako sa začne počítať samotná vzdialenosť jednotlivých pixlov v bitmape, je potrebné previesť množinu úsečiek na binárnu mapu. Táto definuje rez objektu

tak, že pixely s hodnotou 1 ležia vo vnútri a s hodnotou 0 mimo teleso. V prvom kroku je potrebné všetky úsečky rasterizovať. Následne sa prechádzajú jednotlivé riadky bitmapy. Pixely, pre ktoré platí, že medzi nimi a okrajom bitmapy je nepárny počet úsečiek, ktoré skúmaný riadok pretínajú, patria telesu, ostatné sa nachádzajú mimo.



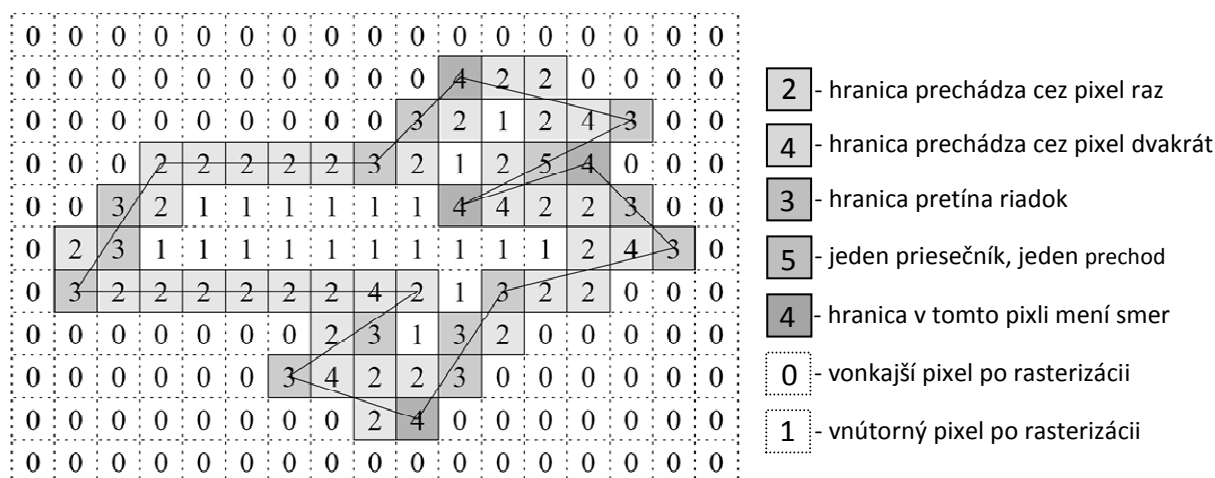
Obrázok
3-2 Pixel v polygóne.

Pixle s nepárnym číslom patria do vnútra telesa.

Úsečky v prvom kroku však nemožno rasterizovať úplne štandardnými spôsobmi. Ak úsečka prechádza nejaký riadok, musí byť označený práve jeden pixel v tomto riadku, inak by pri koncom prechádzaní vznikol chybný výsledok. Podobný prípad nastáva keď dve alebo viacero úsečiek prechádza jedným pixlom.

Tento problém môže byť vyriešený tak, že pri prechádzaní pixlov, ktoré pretína úsečka je hodnota pixlu zvyšovaná o jedna, keď úsečka v danom pixli pretína riadok a takisto je v každom prípade³ zvýšená o dve. Stav po prechode algoritmom ilustruje obrázok 3-3. Následne je táto bitmapa prechádzaná po jednotlivých riadkoch. Riadok sa prechádza zľava doprava. Na začiatku sú pixle označované ako vonkajšie (hodnota 0), až kým sa nenarazí na nepárne číslo. Toto zmení stav a pixle sú označované ako vnútorné (hodnota 1). Každá ďalšia nepárna hodnota mení stav označenia.

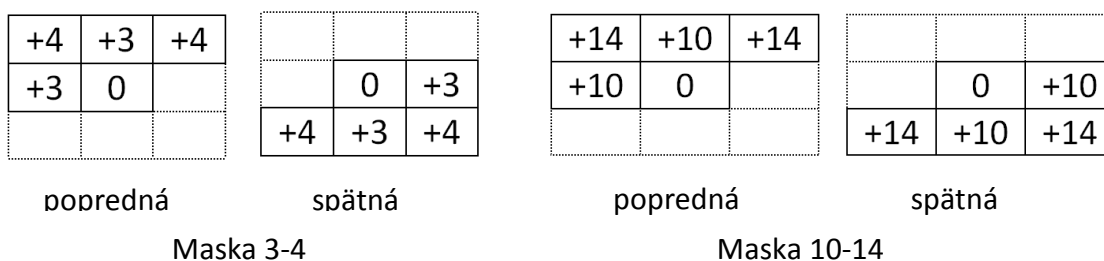
³ Hodnota pixlu, v ktorom úsečka pretína riadok je teda zvýšená celkovo o tri.



Obrázok
3-3 Hodnoty pixlov po prechode rasterizačným algoritmom.

3.2.2 Chamferova transformácia

Rýchlym a jednoduchým riešením ako z binárnej mapy získať mapu vzdialeností je takzvaná Chamferova vzdialenostná transformácia [11]. Táto sa spolieha na predpoklad, že vzdialenosť pixlu je možné zistiť na základe vzdialeností niekoľkých susedných pixlov. Pre účely rekonštrukcie povrchov sa ujala metóda využívajúca masku 3x3 (Chamfer 3x3). Existuje aj riešenie s použitím väčších masiek, napríklad 5x5. Nárast časovej zložitosti je však v porovnaní s pozitívnym dopadom na výsledok značný.



Obrázok
3-4 Chamferove masky.

Chamferova transformácia začína inicializačnou časťou, v ktorej sú pixle s hodnotou 0 nahradené veľkým kladným číslom a pixle s hodnotou 1 veľkým záporným⁴ číslom. Následne

⁴ Kladné a záporné hodnoty je možné vymeniť bez ujmy na všeobecnosti. Je však nutné prispôbiť všetky následné operácie pre opačné hodnoty.

prebieha výpočet v dvoch krokoch. V poprednom a spätnom. V prvom kroku sa prechádza postupne bitmapa z ľavého horného rohu do pravého dolného.

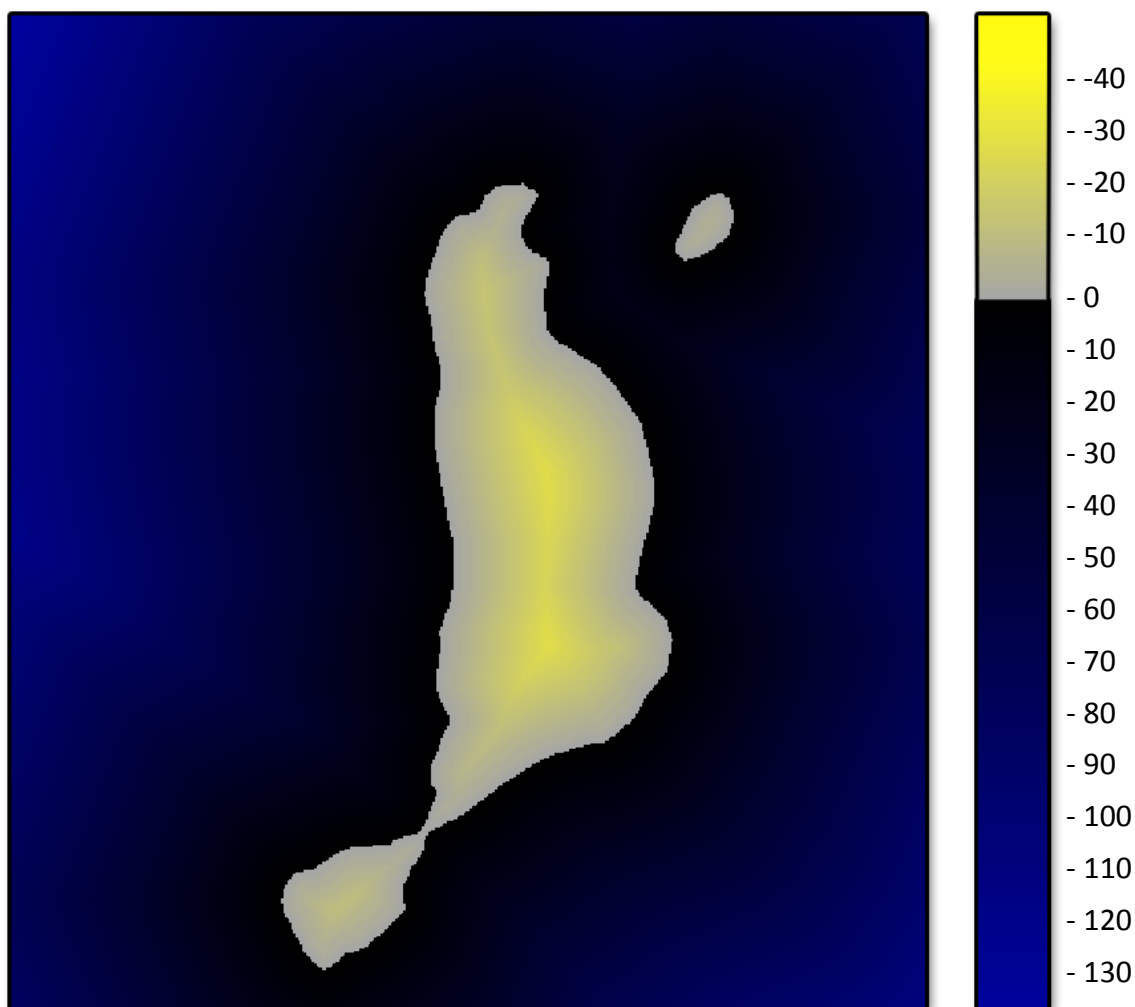
Pre každý vonkajší pixel (v binárnej mape má hodnotu 0) sa vypočíta nová hodnota pixla pomocou Chamferovej masky (viď. obrázok 3-4). Pre poprednú masku Chamfra 3x3, ktorú označíme ako Ch , sa hodnota pixla $P_{i,j}$, ktorý má súradnice i, j vypočíta ako:

$$P_{i,j} = \min \{ P_{i+x,j+y} + Ch(x,y); (x,y) \in \{(0,0), (-1,0), (-1,-1), (0,-1), (1,-1)\} \}$$

3-1

Výsledkom je teda minimum z hodnôt pixlov, ktoré už boli preskúmané, ku ktorým sú pripočítané príslušné hodnoty z Chamfrovej masky. Podobne sa prechádza bitmapa v druhom kroku. Tentokrát však z ľavého dolného rohu do pravého horného. Pre vnútorné pixle sa vo vzorci 3-1 zmení minimum na maximum a algoritmom sa prejdú všetky vnútorné pixle (v binárnej mape majú hodnotu 1).

Algoritmus je možné vylepšiť niektorými úpravami. Napríklad pre pixle, ktoré susedia s kontúrou rezu telesa nastaviť hodnotu +5 pre pixle mimo telesa a -5 v telese. Toto zaručuje, že vzdialenosť je vypočítavaná priamo od samotnej kontúry a nie od najkrajnejších pixlov. Ďalej je možné invertovať operáciu zo vzorca 3-1 pre pixle, ktoré sú mimo teleso tak, že hodnoty masky sú od pixlov odpočítavané a nie pripočítavané a použitá je maximálna hodnota. Toto nám umožní vykonať výpočet súbežne pre celú bitmapu.



Obrázok
3-5 Príklad bitmapy vytvorenej algoritmom chamfer 3x3.

Bitmapa vytvorená algoritmom chamfer 3x3 s použitím masky 10-14. Farby smerom od čiernej k modrej znamenajú nárast vzdialenosti od kontúr rezu, mimo teleso a farby od sivej k žltej znamenajú nárast vzdialenosti od kontúr vo vnútri telesa.

Takmer všetky žiadosti o hodnotu vzdialenosti v reze sú po jednej z predchádzajúcich transformácií uložené v bitmape. Môže sa však stať, že je potrebná hodnota vzdialenosti, ktorá leží mimo bitmapu a je nutné ju vypočítať. Najjednoduchšou možnosťou by bolo prejsť všetky body hranice telesa v reze a vypočítať minimálnu vzdialenosť. Výhodnejšie je však použiť dáta uložené na okrajoch samotnej transformovanej bitmapy. Vzdialenosť po transformácii Chamfrom 3x3 je možné reprezentovať ako Euklidovskú vzdialenosť definovanú úsečkami rovnobežnými alebo naklonenými o 45° k okrajom bitmapy. Pre bitmapu transformovanú Chamfrom s maskou a-b ($a < b$), platí teda, že Chamferovu vzdialenosť bodu $P_{x,y}$ od bodu q môžeme vypočítať takto:

$$P_{x,y}q = \min (a(|j| - |i|) + b|i|, a(|i| - |j|) + b|j|)$$

3-2

Kde i a j sú vzdialenosti bodov $P_{x,y}$ a q v pixeloch pre súradnice x a y . Pre bod $P_{x,y}$, ktorý sa nachádza v hornom koridore nad bitmapou (teda y súradnica je menšia ako 0 a pre x platí $x \geq z$ a $x \leq VX - z$, kde VX je rozlíšenie bitmapy v smere osi x) môžeme hodnotu Chamfrovej vzdialenosti $d_{x,y}$ vypočítať pomocou nasledujúceho vzorca

$$d_{x,y} = \min (P_{x,y}q_{i,0} + d_{i,0})$$

3-3

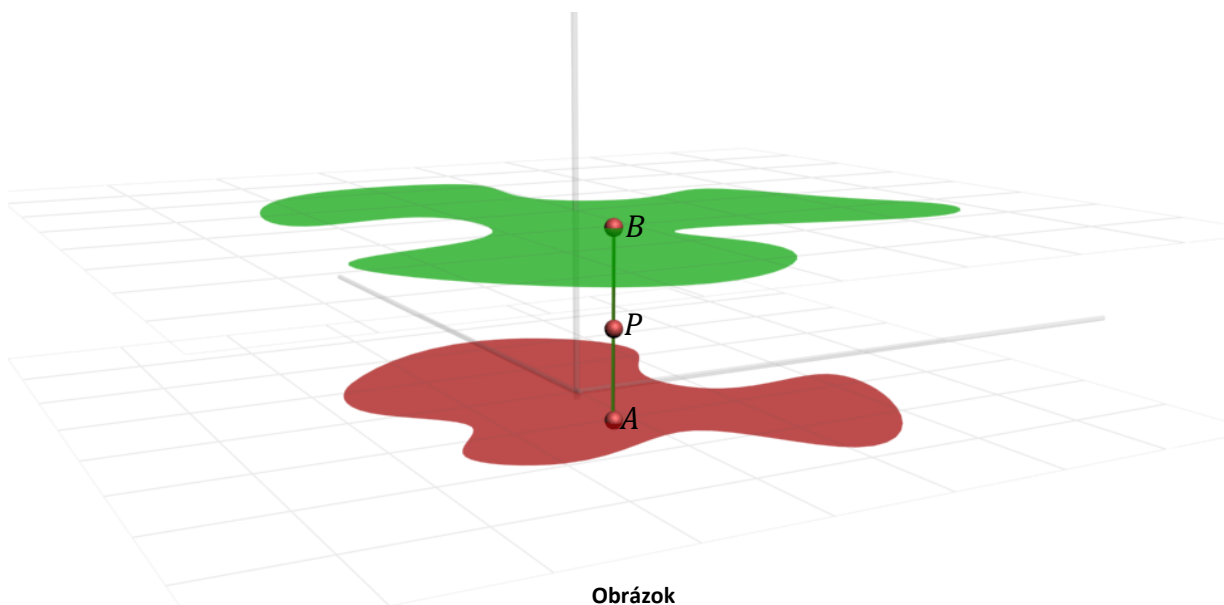
Podobným spôsobom sa dá rovnica 3-3 odvodiť ostatné tri prípady pozície bodu $P_{x,y}$ v okolí bitmapy (vpravo, vľavo a dole). Rovnica 3-3 v podstate definuje algoritmus na výpočet $d_{x,y}$. Postupne prechádzame horný okraj bitmapy od x -ovej súradnice bodu P , smerom doľava aj doprava až k samotným horizontálnym okrajom. Pre každý bod vypočítame Chamfrovu vzdialenosť pomocou rovnice 3-2 a sčítame s hodnotou pixla, na ktorom práve stojíme. Algoritmus môže predčasne skončiť ak sme našli nejakú minimálnu hodnotu $d_{x,y}$, ktorá je menšia ako súčasná vzdialenosť bodov $P_{x,y}$ a $q_{i,0}$.

3.3 Interpolácia rezov obsahujúcich mapu vzdialeností

Úlohou posledného kroku pri vytváraní DF je rozšírenie informácie z 2D bitmáp rezov do 3D priestoru. Toto je možné vďaka interpolácii hodnôt medzi jednotlivými rovinami. V praxi sa najčastejšie využíva interpolácia lineárna. Oveľa dôležitejším faktorom ako druh interpolácie je však v tomto prípade smer, prípadne dráha, po ktorej sa interpoluje.

3.3.1 Interpolácia dvoch rezov

Bežné dáta, ktoré sa majú rekonštruovať sú väčšinou sekvenčné a usporiadané tak, že interpolácia nastáva vždy len medzi dvoma rovinami. Pre bod priestoru P , ktorého hodnotu chceme získať, potrebujeme ďalšie 2 body A a B , ktoré ležia na opačných rezoch (viď. obrázok 3-6).



Obrázok
3-6 Interpolácia hodnôt najbližších susedov.

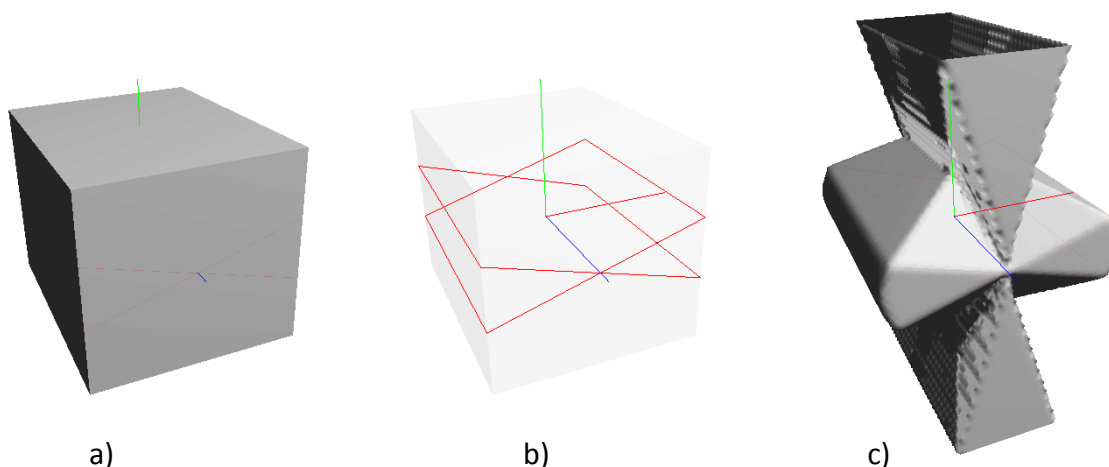
Interpolácia hodnôt v bodoch A a B určujúca hodnotu v bode P , pre paralelné rezy, kde A a B sú najbližší susedia bodu P na susedných rezoch.

Ak označíme $V(A)$ hodnotu v transformovanej bitmape v bode A a podobne $V(B)$ v bode B , hodnotu v bode P , získame lineárnou interpoláciou:

$$V(P) = \frac{V(A) \cdot |B - P| + V(B) \cdot |A - P|}{|A - P| + |B - P|}$$

3-4

Najjednoduchší spôsob ako body A a B získať je výpočet najbližších bodov k bodu P na susedných rezoch. Takáto voľba dáva prijateľné výsledky pre paralelné rezy. Úplne nevhodne sa však chová, ak roviny rezov zvierajú veľmi malý uhol. V takomto prípade vznikajú artefakty šíriace sa po ose týchto dvoch rezov (obrázok 3-7).

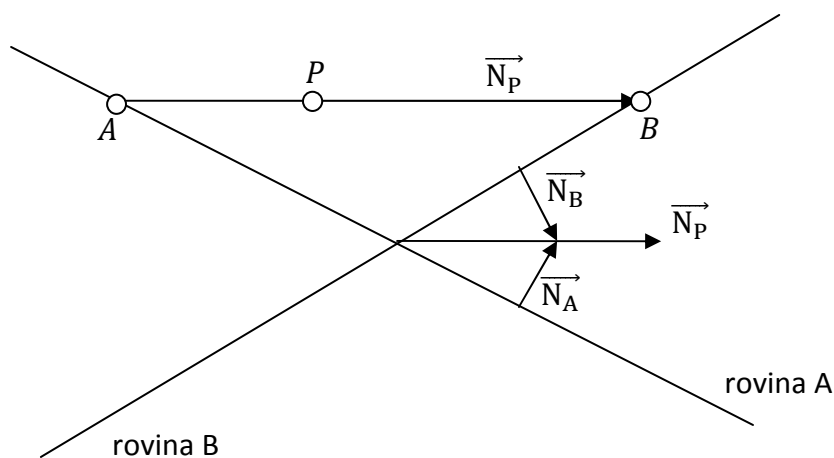


Obrázok

3-7 Artefakty vznikajúce pri malom uhle pri použití algoritmu najbližšieho bodu.

Na obrázku a) je pôvodná kocka. Obrázok b) znázorňuje rezy zvierajúce malý uhol. Na obrázku c) vidno rekonštruovaný povrch.

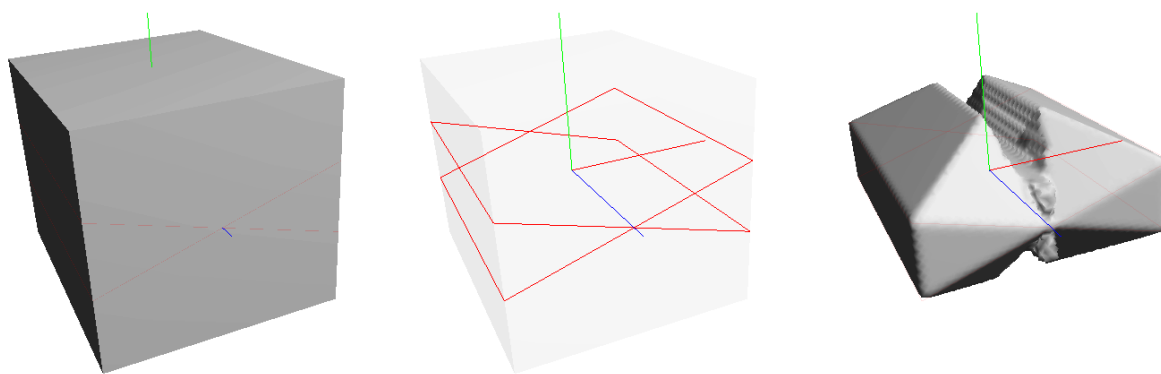
Druhou možnosťou, získania bodov A a B , je výpočet priemernej normály rovín dvoch rezov. Táto normála je následne použitá ako smernica priamky prechádzajúca bodom P . Priesečníky tejto priamky s rovinami rezov určujú naše interpolačné body (obrázok 3-8).



Obrázok

3-8 Pozície bodov A a B pri interpolácii s priemernou normálou.

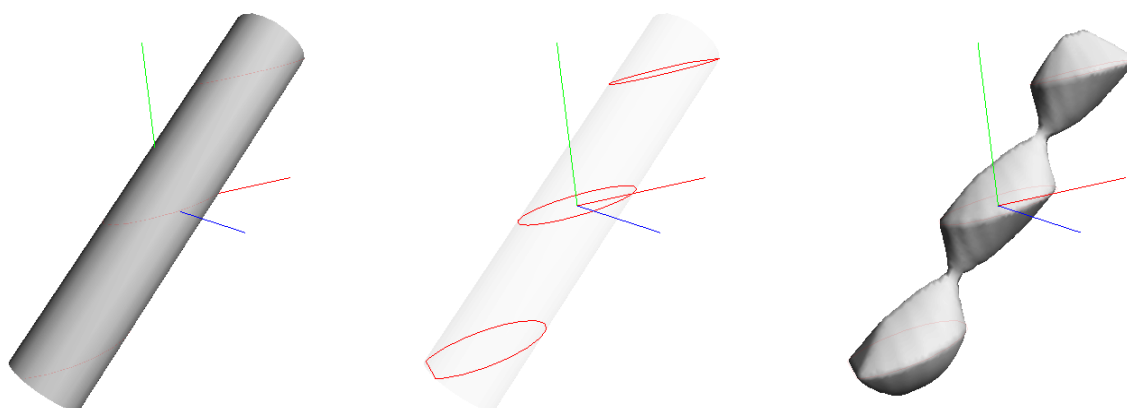
Takéto riešenie je oveľa prirodzenejšie, pretože oproti prípadu najbližších bodov interpolujeme po jednej priamke, vďaka čomu nevznikajú podobné artefakty. Na obrázku 3-9 vidno rozdiel oproti prvému návrhu.



Obrázok

3-9 Rekonštrukcia pri malom uhle rezných rovín pomocou priemernej normály.

Metóda založená na interpolácii cez priemernú normálu poskytuje uspokojivé výsledky vo veľkom množstve prípadov. Neberie však do úvahy vlastnosti rezov. Smer interpolácie je určený iba z orientácie rezných rovín. Tento fakt sa najviac prejavuje, ak sú kontúry telesa v dvoch susedných rovinách príliš odlišné (obrázok 3-10). S novým návrhom, ktorý rieši tento problém prišiel Treece [8].



Obrázok

3-10 Neschopnosť adaptácie sa interpolácie pomocou priemernej normály pri veľmi vzdialených rezoch

3.4 Interpolácia pomocou maximálnych kružníc (disc guided interpolation)

Hlavnou myšlienkou v Treecovom návrhu je definovať rôzne interpolačné smery pre rôzne časti rezov. Tieto smery by mali závisieť od vzájomnej súvislosti jednotlivých častí. Algoritmus sa skladá z troch krokov. V prvom sa prevedú bitmapy rezov na minimálne množiny maximálnych kružníc. Pôvodné dáta sú však zachované. Pre dvojicu rovín, medzi ktorými sa bude interpolovať sa vypočíta príbuznosť týchto kružníc z jednej roviny s

kružnicami z druhej roviny. Výsledkom je množina vektorov, ktoré túto príbuznosť určujú. Pri interpolácii je pomocou týchto čiastkových príbuzností vypočítaný interpolačný vektor pre konkrétny bod P , v ktorom chceme získať hodnotu zobrazenia.

3.4.1 Minimálna množina maximálnych kružníc

Dáta uložené v bitmapách rezov nám síce dávajú dobrú predstavu o lokálnych vlastnostiach, ale nie je z nich možné priamo získať globálnejšie informácie o tvare telesa. Na tieto účely je veľmi vhodná reprezentácia kontúr pomocou takzvaných maximálnych kružníc. Maximálnou kružnicou sa myslí taká kružnica, ktorá sa dotýka kontúr telesa v reze v minimálne dvoch bodoch. Extrahovať takéto kružnice z bitmapy, ktorá bola vytvorená s jednou zo vzdialenostných transformácií je jednoduché. Pre stred každej takejto kružnice platí, že sa nachádza v lokálnom minime, pre ktoré platí:

$$p < q_{i,j} - \begin{cases} a; i,j \in \{(1,0), (-1,0), (0,1), (0,-1)\} \\ b; i,j \in \{(1,1), (-1,1), (-1,-1), (1,-1)\} \end{cases}$$

3-5

Kde i a j sú relatívne koordináty vzhľadom na bod p , a $q_{i,j}$ sú hodnoty uložené v bitmape. a a b sú koeficienty Chamferovej transformácie ($a < b$). Rovnica 3-5 teda preskúma 8 susedných bodov v okolí bodu p . Na získanie všetkých kružníc stačí teda prejsť všetky body bitmapy a otestovať ich na túto podmienku. Pre body mimo teleso (hodnota má opačné znamienko ako vo vnútri telesa), sa v podmienke zmení znamienko - na + a $<$ na $>$. Kružnice môžu existovať aj mimo bitmapu, ale pre účely získania interpolačných vektorov majú len malý význam a preto môžu byť zanedbané.

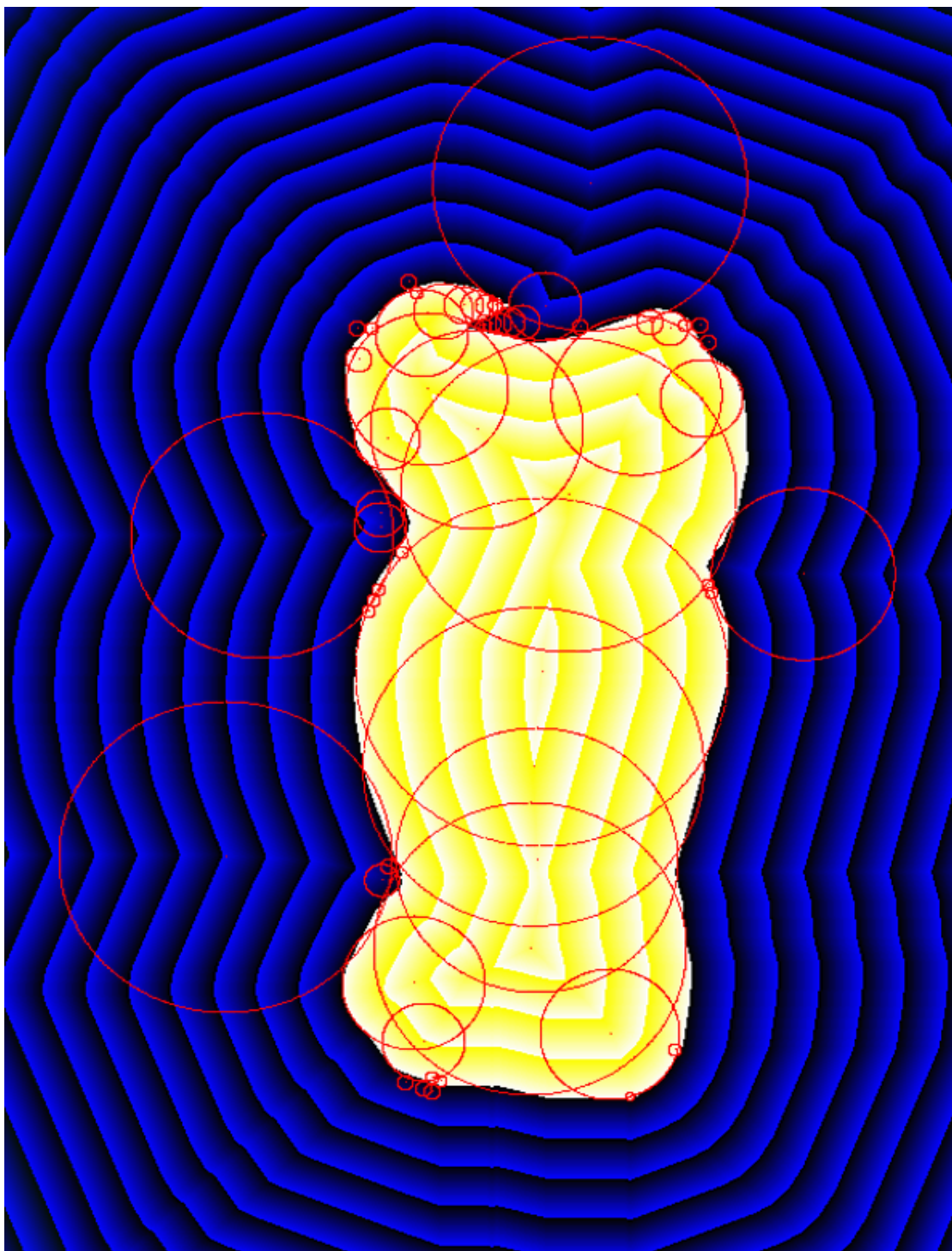
Pomocou takto získaných kružníc je možné definovať presné kontúry rezov. Veľké množstvo z nich je však redundantných, niektoré sú obsiahnuté v iných a neobsahujú žiadne užitočné informácie. Existuje viacero možností ako množinu maximálnych kružníc redukovať na takzvanú minimálnu množinu, v ktorej sú všetky kružnice potrebné na úplnú definíciu kontúr a súčasne žiadna nie je zbytočná. V našom prípade však postačí aj menšia množina ako minimálna, pretože presné hodnoty budú interpolované priamo z bitmap vzdialeností a kružnice sú iba akýmsi vodítkom. Menší počet kružníc zlepšuje aj celkovú časovú náročnosť.

Algoritmus minimalizovania množiny kružníc, ktorý ich generujúci dostatočné množstvo pracuje iteratívne. Vyberie kružnicu s najväčším polomerom a odstráni všetky kružnice, pre ktoré platí, že z nej vyčnievajú o menej ako o polovicu svojho polomeru (vzorec 3-6) a pokračuje druhou najväčšou kružnicou, ktorá sa v množine nachádza. Výsledok je zobrazený na obrázku 3-11.

$$\|C_i - C_j\| \geq |R_i - R_j|$$

3-6

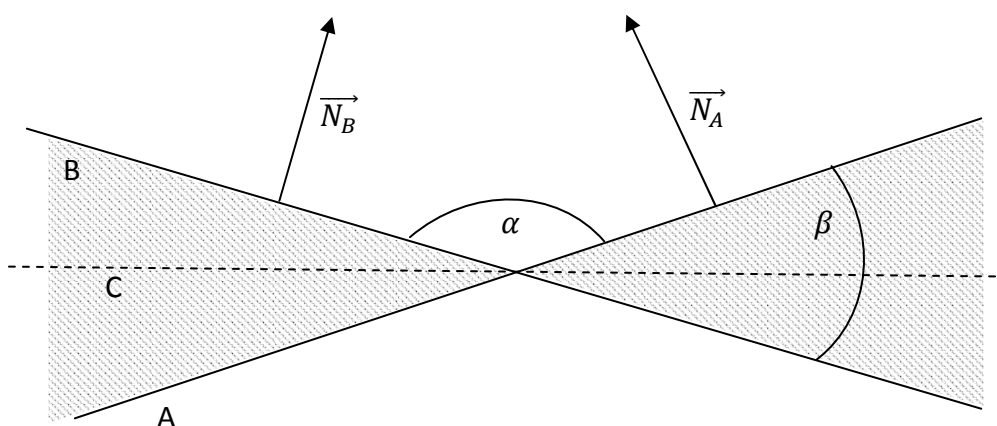
Podmienka na existenciu dvoch kružníc i a j v minimálnej množine. C_i sú súradnice kružnice i a R_i je jej polomer.



3-11 Množina maximálnych kružníc po odstránení všetkých, ktoré nespĺňajú rovnosť 3-6

3.4.2 Výpočet regionálnej korešpondencie

Všetky rezy boli v predchádzajúcom kroku prevedené na množiny kružníc, ktoré popisujú kontúry. Ďalšou úlohou je popísať príbuznosť kružníc z opačných rezov. Preto, aby bolo vôbec možné jednotlivé dvojice porovnávať, najprv sa nájde takzvaná priemerná rovina, do ktorej sú všetky kružnice projektované. Normála takejto roviny je priemerom normál rovín rezov, z ktorých chceme interpolovať. Keďže jednu rovinu môžeme definovať dvoma rôznymi normálami $\vec{N}_+$ a $\vec{N}_-$, pričom $\vec{N}_+ = -\vec{N}_-$, existujú aj dve možnosti ako zvoliť výpočetnú rovinu. Logickejším krokom je interpolovať iba oblasti, ktoré vyplňajú menší uhol a teda rovinu umiestniť do nej (obrázok 3-12).



Obrázok

3-12 Zvolenie priemernej roviny, v ktorej sa počíta regionálna korešpondencia. $\alpha < \beta$

Všetky kružnice definované v rovinách rezov A a B z obrázku 3-12 sú premietnuté po priemernej normále na rovinu C, v ktorej prebiehajú všetky následné výpočty. Pre každú dvojicu kružníc $C_a \in A_{int}$, $C_b \in B_{int}$ a $C_a \in A_{ext}$, $C_b \in B_{ext}$ ⁵, je vypočítaná pravdepodobnosť korešpondencie týchto dvoch kružníc na základe rozdielu hodnôt uložených v bitmape vzdialeností v stredoch kružníc a ich polomeroch. Tento výpočet nám dáva odhad chyby, ktorá ak je malá v porovnaní s polomermi týchto dvoch kružníc, môže byť použitá ako pravdepodobnosť ich korešpondencie. Ak je táto chyba väčšia ako niektorý z polomeroch týchto diskov, neexistuje medzi nimi žiadna korešpondencia. Výslednú korešpondenciu pre kružnice a a b vypočítame takto:

⁵ int je množina kružníc so stredom v telese a ext množina kružníc mimo teleso. Korešpondencia sa počíta teda zvlášť pre interné a externé kružnice.

$$\omega_{a,b} = \begin{cases} \frac{1}{\varepsilon_{a,b} + \mu} + \frac{1}{\varepsilon_{b,a} + \mu} & \text{keď } \varepsilon_{a,b} < r_a \text{ a } \varepsilon_{b,a} < r_b \\ 0 & \text{v opačnom prípade} \end{cases}$$

3-7

$\omega_{a,b}$ je kombináciu chyby spočítanej pre obidve kružnice, r_a a r_b sú ich polomery, μ je konštanta, ktorá zabraňuje vzniku nekonečnej hodnoty $\omega_{a,b}$, a je zvolená ako obsah pixelu v bitmape vzdialeností (pre Chamfer 10-14 je to $10 \cdot 10 = 100$). $\varepsilon_{a,b}$ a $\varepsilon_{b,a}$ sú odhady chyby medzi dvoma kružnicami. Môžeme ich vypočítať pomocou nasledujúcej rovnice:

$$\begin{aligned} \varepsilon_{a,b} &= \text{length}(\vec{l}_{a,b}) - |r_a - d_b| \\ \varepsilon_{b,a} &= \text{length}(\vec{l}_{a,b}) - |r_b - d_a| \end{aligned}$$

3-8

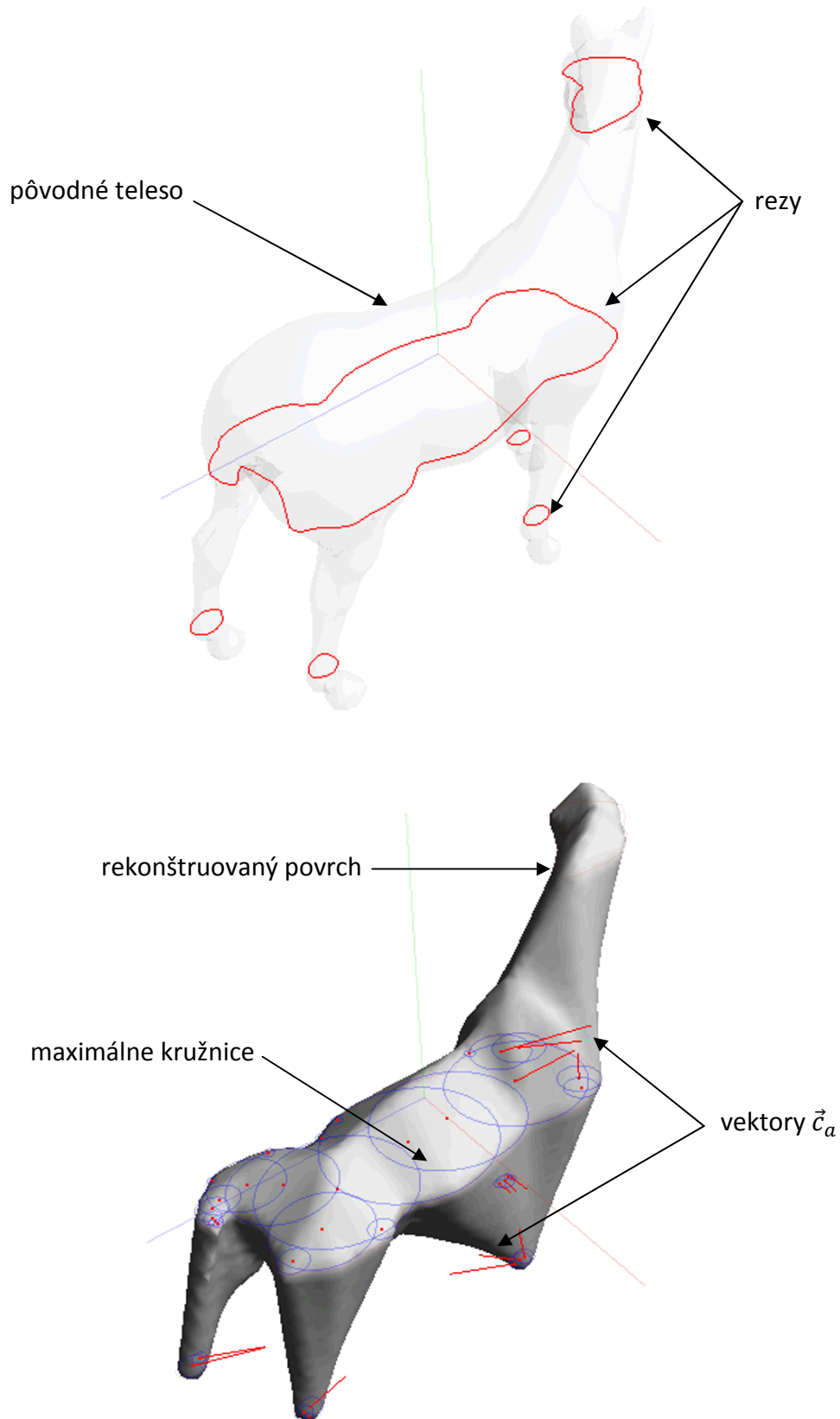
d_a je hodnota v bitmape vzdialeností roviny B v bode, ktorý vznikne projekciou stredu kružnice z roviny A po normále priemernej roviny na rovinu B. Podobným spôsobom je definované aj d_b . Funkcia *length* udáva dĺžku vektora $\vec{l}_{a,b}$, ktorý spája stredy kružníc, premietnutých na priemernú rovinu, ktorých koordináty boli prevedené do bitmapy vzdialeností. Táto dĺžka však závisí od toho akou transformáciou bola bitmapa vytvorená. Ak bola použitá Euklidovská vzdialenosť, je aj *length* vypočítaná ako Euklidovská vzdialenosť, v prípade použitia chamfera je na výpočet použitá rovnica 3-2. Je dobré si uvedomiť, že celý výpočet regionálnej korešpondencie zatiaľ prebieha v 2D priestore bitmáp vzdialeností, ktoré ležia v jednej priemernej rovine. Vďaka tomuto faktoru môže byť korešpondencia vypočítaná pre neparalelné rezy.

Hodnota $\omega_{a,b}$ nám udáva ako veľmi spolu korešpondujú páry kružníc z dvoch susediacich rezov. Ďalším krokom je využiť túto hodnotu a pre každú kružnicu z roviny A určiť vektor smerujúci k oblasti, s ktorou najviac korešponduje. Pre kružnicu C_a je výpočet takýto:

$$\vec{c}_a = \begin{cases} \frac{1}{\sum_{b \in \text{discs}B} \omega_{a,b}} \cdot \sum_{b \in \text{discs}B} \omega_{a,b} \vec{l}_{a,b} & \text{keď } \sum_{b \in \text{discs}B} \omega_{a,b} > 0 \\ 0 & \text{v opačnom prípade} \end{cases}$$

3-9

$\omega_{a,b}$ sú váhy z rovnice 3-7, *discsB* je množina kružníc z roviny B a $\vec{l}_{a,b}$ je vektor smerujúci z premietnutého stredu kružnice C_a do priemernej roviny do rovnako premietnutého stredu kružnice C_b . Vektory vypočítané takýmto spôsobom určujú ako korešponduje kružnica z roviny A s kružnicami roviny B. Keďže na výpočet sú použité váhy založené na odhade chyby, len zriedkakedy smeruje vektor priamo do stredu inej kružnice. Takýmto spôsobom je zaručená možnosť výskytu úplnej, žiadnej, ale aj čiastočnej korešpondencie. Rovnakým spôsobom je vypočítaná korešpondencia kružníc z roviny B s kružnicami s rovinou A. Korešpondencia vypočítaná iba z jednej strany nie je postačujúca kvôli nesymetrickosti.



Obrázok
3-13 Zobrazenie vektorov $\vec{c}_a$ pri rekonštrukcii modelu koňa

Obrázok zobrazuje rekonštrukciu modelu koňa pomocou troch paralelných rezov. Červené čiary reprezentujú vektory $\vec{c}_{a,b}$. Vrchný rez neobsahuje ani jeden takýto vektor, pretože pre dvojicu rovín ich stačí vypočítať len pre jednu z nich. Nie sú zobrazené vonkajšie kružnice.

3.4.3 Výpočet bodovej korešpondencie

Na to, aby bolo možné reálne využiť korešpondenčné vektory, je nutné vedieť ich smer a veľkosť nielen pre stredy kružníc, ale pre akýkoľvek bod P z roviny rezu. K výpočtu takéhoto vektora v ľubovoľnom bode môžeme použiť vektory $\vec{c}_a$. Podobne ako v predchádzajúcom prípade, výsledkom bude vážený priemer. Tentokrát sa však najprv zistia korešpondencie medzi kružnicami z roviny A a bodom P . Tieto hodnoty sú následne použité ako váhy určujúce dôležitosť jednotlivých vektorov $\vec{c}_a$ pre tvorbu výsledného vektora $\vec{c}_p$. Ten vypočítame takto:

$$\vec{c}_p = \begin{cases} \frac{1}{S_{int}} \sum_{a \in int} \omega_a \vec{c}_a & \text{keď } S_{ext} = 0 \\ \frac{1}{2(l_{int} + d_p)(S_{int} + S_{ext})} \left[(l_{int} + 2d_p) \sum_{a \in int} \omega_a \vec{c}_a + l_{int} \sum_{a \in ext} \omega_a \vec{c}_a \right] & \text{inak, keď } d_p > 0 \\ \frac{1}{2(l_{ext} - d_p)(S_{int} + S_{ext})} \left[(l_{ext} - 2d_p) \sum_{a \in int} \omega_a \vec{c}_a + l_{ext} \sum_{a \in ext} \omega_a \vec{c}_a \right] & \text{inak} \end{cases} \quad 3-11$$

int je množina kružníc ležiacich v telese, ext zase množina kružníc ležiacich mimo teleso, $S_{ext} = \sum_{a \in ext} \omega_a$ a $S_{int} = \sum_{a \in int} \omega_a$, d_p je hodnota v bitmape vzdialeností v bode P . Váhové koeficienty sú definované podobne ako v rovnici 3-7, rozdiel je len v tom, že namiesto stredy kružnice druhej roviny máme bod P :

$$\omega_a = \frac{1}{\varepsilon_a + \mu}$$

3-12

$$\varepsilon_a = length(\vec{l}_{a,p}) - |r_a - d_p|$$

3-13

Podobne ako v prípade korešpondencií kružníc, je korešpondencia bodu a kružnice definovaná pomocou vzájomnej vzdialenosti a vzdialeností⁶ od kontúr rezu.

Výpočet korešpondenčného vektora $\vec{c}_p$ (rovnica 3-11) sa rozdeľuje na tri prípady. Prvý nastáva vtedy, ak neexistujú žiadne externé kružnice, prípadne bod P nemá so žiadnou externou kružnicou hodnotu korešpondencie väčšiu ako 0. V opačnom prípade je výpočet zložený z kombinovaného váženého priemeru s rôznymi hodnotami s malým rozdielom pre body mimo teleso $d_p > 0$ a v telese $d_p < 0$. Ďalej je výsledok ovplyvnený parametrami l_{int} a l_{ext} , ktoré vyjadrujú vzdialenosť bodu P od externých a interných kružníc:

⁶ Polomer kružnice, udáva vzdialenosť k najbližšiemu bodu na kontúre rezu.

$$l_{int} = \frac{1}{s_{int}} \sum_{a \in int} \omega_a l_a, \quad l_{ext} = \frac{1}{s_{ext}} \sum_{a \in ext} \omega_a l_a$$

3-14

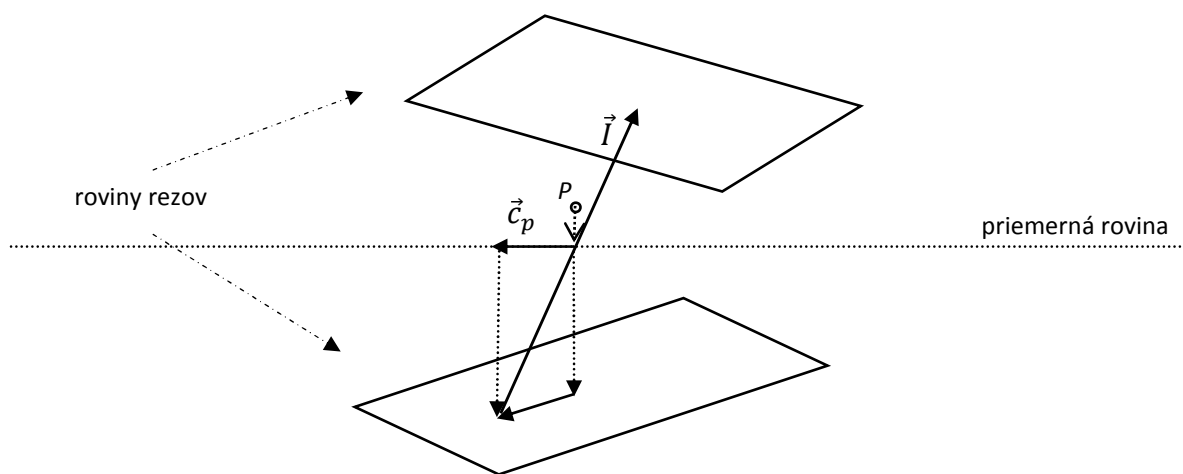
Kde l_a je vzdialenosť bodu P od stredu kružnice C_a . Tieto parametre slúžia na to, aby bod, ktorý sa nachádza v telese bol ovplyvnený viac internými kružnicami ako externými a naopak.

Pre ľubovoľný bod A v priestore medzi dvoma rezmi vypočítame vektory $\vec{c}_p$ v bodoch, ktoré vzniknú ako priesečníky obkolesujúcich dvoch rezov a vektora prechádzajúceho bodom A v smere normály na priemernú rovinu. Výsledný vektor $\vec{c}_p$ je potom priemerom prvého a opačného k druhému. Takýto vektor je umiestnený v priemernej rovine.

3.4.4 Výpočet interpolačného vektora

Posledným krokom je na základe bodovej korešpondencie určiť smer interpolačného vektora I . Ak získame takýto smer, potom pre ľubovoľný bod P môžeme určiť dva body na susedných rezoch ako priesečníky vektora I prechádzajúceho bodom P . Hodnotu distance function bude potom možné získať v bode P lineárnou interpoláciou hodnôt v týchto dvoch bodoch.

Pre ľubovoľný bod P medzi dvoma rezmi vypočítame vektor $\vec{c}_p$. Takýto vektor určuje smer interpolácie od „spodného“ rezu k „hornému“. Spodná rovina je tá, z ktorej sme pri výpočte výsledného vektora $\vec{c}_p$ zobrali priamy vektor a horná tá, z ktorej sme zobrali opačný. Vektor $\vec{c}_p$ je potom možné premietnuť priamo na spodnú rovinu alebo opačne na hornú. Výhodnejšie je premietnuť na takú rovinu rezu, ktorá spôsobí, že premietnutý vektor bude smerovať preč od priemernej roviny. V takomto prípade nevznikajú žiadne artefakty ani pri prekrížení rovín rezov. Smer interpolácie je potom určený koncovým bodom premietnutého vektora a bodom P (obrázok 3-14).

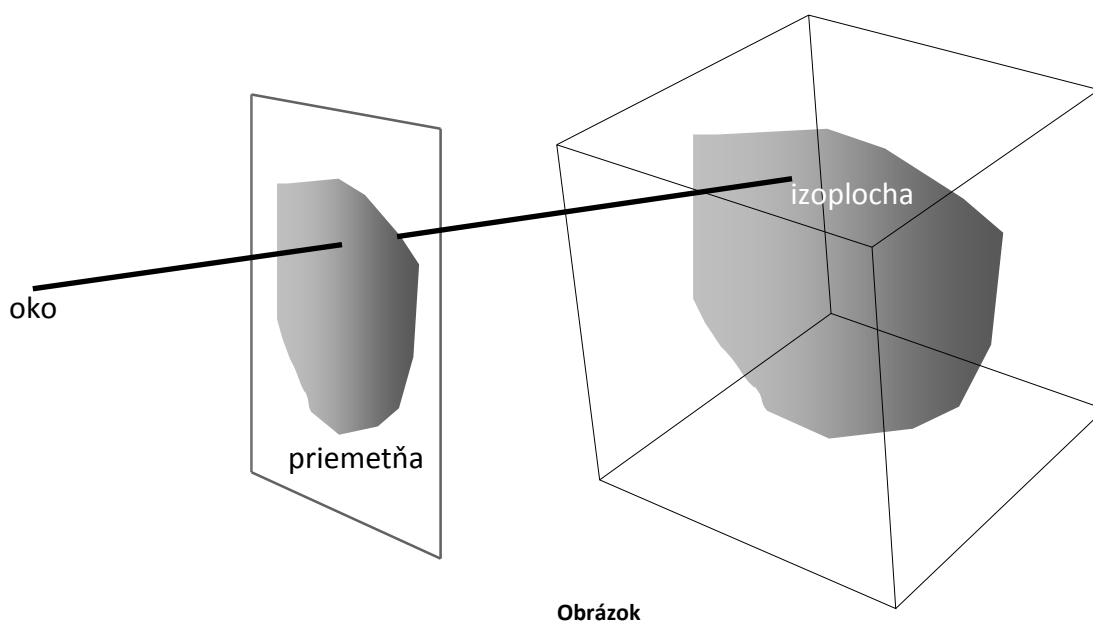


Obrázok
3-14 Výpočet interpolačného vektora z vektora $\vec{c}_p$

4 Zobrazovanie izoplochy

4.1 Priame zobrazovanie izoplochy

Priame zobrazovacie metódy nepoužívajú žiadne polygonálne primitíva, namiesto toho sa snažia zobraziť povrch izoplochy pomocou priamej projekcie na priemetňu. Z principiálneho hľadiska ide o veľmi jednoduchý spôsob. Celý priestor je rozdelený na bunky, v ktorých vrcholoch sú definované hodnoty DF. Farba pixla na obrazovke je získavaná pomocou postupného prechodu lúča od kamery, cez priemetňu a ďalej cez sieť buniek. V každej z týchto buniek je nutné rozhodnúť, či v nej lúč izoplochu pretína alebo nepretína. Komplikácie však nastávajú práve v jednotlivých častiach algoritmu. Na tieňovanie je nutné použiť niektorý zo svetelných modelov, nie je možné využiť metódy tieňovanie trojuholníkov. Nezanedbateľným problémom je aj veľká časová náročnosť výpočtu. Na rozdiel od nepriamych zobrazovacích metód musí byť pri každej zmene (poloha svetla, poloha pozorovateľa...) celý postup vykonaný odznova.



4-1 Priesečník lúča s izoplochou pri priamej zobrazovacej metóde

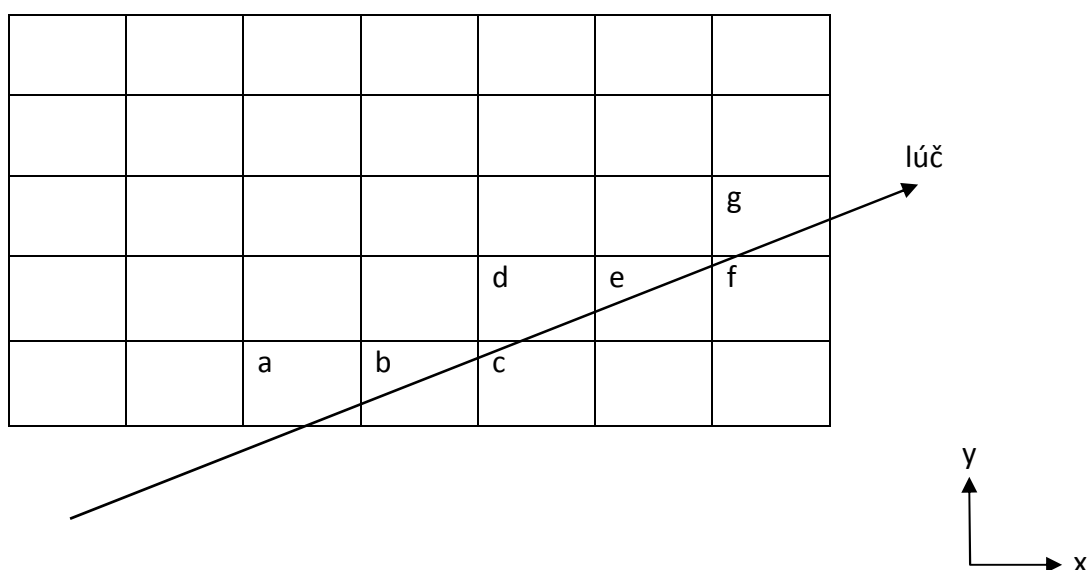
4.1.1 Interaktívne sledovanie lúča pre zobrazenie izoplochy

Základný algoritmus priameho zobrazovania izoplochy je popísaný v [12]. Tento algoritmus sa skladá z troch častí. V prvej sa prechádzajú bunky, ktoré neobsahujú priesečník s izoplochou. V druhej sa vypočítava priesečník s izoplochou v bunke, ktorá priesečník obsahuje a v tretej je vypočítavané tieňovanie povrchu. Celý tento proces sa vykonáva pre

každý zobrazovaný pixel zvlášť⁷. Vďaka tomu je sledovanie každého lúča nezávislé na ostatných a celú metódu je možné veľmi ľahko paralelizovať. Veľkou výhodou je možnosť jednoduchého pridania pokročilejších tieňovacích techník akými sú napríklad tieň.

4.1.2 Prechádzanie buniek

Prvá časť algoritmu je v najjednoduchšom prípade založená na metóde popísanej v [13]. Základom je algoritmus DDA. V nasledujúcom texte je popísaná metóda pre dvojrozmerný prípad. Prechod do tretieho rozmeru je priamočiary.



Obrázok

4-2 Prechádzanie lúča mriežkou buniek pri metóde sledovania lúča

Správne prechádzanie mriežky je znázornené na obrázku 4-2. Algoritmus musí postupne navštíviť bunky a, b, c, d, e, f, g v danom poradí. Rovnicu lúča v takomto prípade vyjadríme pomocou parametrického vyjadrenia $\vec{u} + t\vec{v}$ pre $t > 0$. Lúč je v algoritme rozdelený na intervaly, pričom každý z nich prechádza práve jednou bunkou.

Metóda sa skladá z dvoch základných krokov. V prvom sa určí prvá bunka, do ktorej lúč vstúpi. Inicializované sú celočíselné hodnoty X a Y, ktoré definujú pozíciu tejto bunky v mriežke. Ďalej sú zistené hodnoty stepX a stepY hodnotami 1 alebo -1 podľa toho, či lúč bude v danej súradnici klesať alebo rásť (toto je možné zistiť pomocou znamienok zložiek vektora $\vec{v}$).

V ďalšom kroku je vypočítaná hodnota parametru t , v ktorom pretína lúč hranicu nasledujúcej vertikálnej bunky. Podobne je získaná aj hodnota pre horizontálny smer. Minimum z týchto dvoch parametrov udáva, ako ďaleko je možné ísť a nepreskočiť pritom

⁷ V praxi môže byť pre niektoré pixle sledovaných viacero lúčov a pre niektoré žiaden. Toto sa však uplatňuje hlavne pri zobrazovaní objemových dát a nie pri zobrazovaní izoplôch.

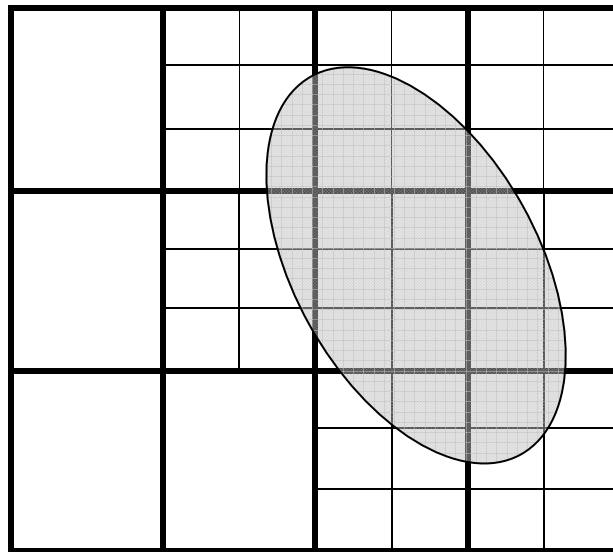
do žiadnej ďalšej bunky. Pomocou tejto hodnoty sú vypočítané prírastky v horizontálnom a vertikálnom smere označené $tDeltaX$ a $tDeltaY$. Na základe týchto parametrov je možné určiť nasledujúcu bunku a začiatočnú pozíciu lúča v nej. Toto je základný náčrt druhého kroku popisovaného algoritmu:

```

loop
{
    if (tMaxX < tMaxY)
    {
        tMaxX += tDeltaX;
        X += stepX;
    }
    else
    {
        tMaxY += tDeltaY;
        Y += stepY;
    }
    NextVoxel(X, Y);
}

```

Algoritmus postupne prechádza jednotlivé bunky. Cyklus končí ak je nájdená bunka, v ktorej sa nachádza priesečník s izoplochou alebo sa lúč dostane mimo mriežku. Na prechod do tretieho rozmeru stačí pridať parametre končiace Z a hľadať minimum z hodnôt $tMaxX$, $tMaxY$ a $tMaxZ$.



Obrázok

4-3 Zobrazenie dvojstupňovej hierarchie v dvojrozmernom analogickom prípade urýchľovania algoritmu prehľadávania mriežky

Takto navrhnutý algoritmus je jednoduchý ale v praxi značne pomalý. Jeden zo spôsobov ako ho zrýchliť bol popísaný v [12]. Zrýchlenie nastáva pri prechádzaní prázdnych buniek⁸. Bunky sú najprv rozdelené na rovnaké skupiny. Na ich mieste sú potom vytvorené

⁸ Prázdnu bunkou sa v prípade zobrazovania izoplochy definovanej pomocou DF myslí taká bunka, ktorá má vo všetkých krajných bodoch hodnotu vyššiu ako hraničnú hodnotu izoplochy.

väčšie bunky, ktoré tieto pôvodné zahŕňajú. Hodnoty DF v krajných bodoch sú potom definované maximálnymi a minimálnymi hodnotami z pôvodných buniek. Algoritmus prechádzajúci bunky takto najprv skontroluje hodnoty v týchto veľkých bunkách a iba ak zistí, že sa v niektorej z nich musí nachádzať priesečník s izoplochou, prehľadáva hlbšie.

Pri mriežke obsahujúcej $n \times n \times n$ buniek a použití trojstupňovej hierarchie bude priemerná zložitosť vyhľadávania $O(\sqrt[3]{n})$ pričom v najhoršom prípade ostáva stále $O(n)$, v praxi však k najhoršiemu prípadu takmer nedochádza.

4.1.3 Výpočet priesečníka s izoplochou v rámci bunky

V predchádzajúcom kroku sme získali bunku, v ktorej môže nastať priesečník so sledovaným lúčom. Na vypočítanie čo najpresnejšej polohy priesečníka s izoplochou máme k dispozícii iba hodnoty DF vo vrcholoch bunky. V praxi sa hodnoty DF vo vnútri takejto bunky vypočítavajú pomocou trilineárnej interpolácie. Lúč prechádzajúci bunkou je potom charakterizovaný kubickým polynómom. Výsledok priesečníka lúča s izoplochou je potom definovaný ako prvý nezáporný koreň tejto rovnice. Existujú aj presnejšie a hladšie interpolácie, ktoré sú však oveľa náročnejšie na časovú zložitosť a ich využitie v zobrazovaní v reálnom čase je zatiaľ nevhodné. V nasledujúcich odstavcoch je najprv popísaná presná metóda výpočtu z trilineárnej interpolácie a ďalej jej aproximácie, ktoré poskytujú síce nepresnejšie výsledky, ale výpočet je značne urýchlenný.

Presná metóda hľadania priesečníka s izoplochou

Označme hodnoty zobrazenia DF vo vrcholoch bunky ako $p_{i,j,k}$ ($i, j, k \in \{0,1\}$), potom hodnotu DF vo vnútri bunky v bode $(u, v, w) \in [0,1]^3$ je vypočítaná z trilineárnej interpolácie ako:

$$\rho(u, v, w) = \sum_{i,j,k \in \{0,1\}} u_i v_j w_k \rho_{i,j,k}$$

4-1

Pričom $u_0 = u$, $u_1 = 1 - u$, $v_0 = v$ a tak ďalej. Keď sú súradnice bunky definované takto: $V = [x_0..x_1] \times [y_0..y_1] \times [z_0..z_1]$, potom hodnota $\rho(p)$ ľubovoľného bodu $p \in V$ môže byť vypočítaná pomocou transformovaného bodu:

$$p_0 = (u_0^p, v_0^p, w_0^p) = \left(\frac{x_1 - p_x}{x_1 - x_0}, \frac{y_1 - p_y}{y_1 - y_0}, \frac{z_1 - p_z}{z_1 - z_0} \right)$$

4-2

Ak označíme lúč, ktorým prechádzame bunku parametrickou rovnicou $R(t) = a + tb$, pričom lúč je ohraničený v bunke na intervale t_{in} až t_{out} a definujeme hodnotu DF v ľubovoľnom bode, ležiacom na lúči v danej bunke ako $\rho(t) = \rho(R(t))$, potom túto hodnotu môžeme vypočítať z polynómu:

$$\rho(t) = \sum_{i,j,k \in \{0,1\}} (u_i^a + tu_i^b)(v_j^a + tv_j^b)(w_k^a + tw_k^b)$$

4-3

Po roznásobení zátvoriek vznikne rovnica:

$$\rho(t) = At^3 + Bt^2 + Ct + D$$

4-4

pričom koeficienty sú definované takto:

$$\begin{aligned} A &= \sum_{i,j,k} u_i^b v_j^b w_k^b \rho_{i,j,k} \\ B &= \sum_{i,j,k} (u_i^a v_j^b w_k^b + u_i^b v_j^a w_k^b + u_i^b v_j^b w_k^a) \rho_{i,j,k} \\ C &= \sum_{i,j,k} (u_i^b v_j^a w_k^a + u_i^a v_j^b w_k^a + u_i^a v_j^a w_k^b) \rho_{i,j,k} \\ D &= \sum_{i,j,k} u_i^a v_j^a w_k^a \rho_{i,j,k} \end{aligned}$$

4-5

Priesečník s izoplochou definovanou hranicou s hodnotou ρ_{iso} je potom najmenšie $t \in [t_{in}, t_{out}]$, pre ktorý je nulový polynóm:

$$f(t) := \rho(t) - \rho_{iso}$$

4-6

Takto vytvorený polynóm je možné vypočítať pomocou Schwarzeho metódy [14], ktorá najprv skúma špeciálne prípady (malé koeficienty A a B), ktoré vie vyriešiť priamočiaro. Výpočet už však aj v tomto prípade obsahuje druhú odmocninu. Plný kubický polynóm sa ďalej rieši pomocou Cordanoveho vzorca, ktorý bohužiaľ obsahuje niekoľko kosínusov a na výpočtový čas ešte náročnejších inverzných kosínusov. Navyše potrebujeme získať všetky riešenia, aby sme mohli určiť prvý priesečník s izoplochou.

Aproximačné metódy

Kontrastom k presným metódam sú metódy, ktoré sa snažia vypočítať výsledok priesečníka lúča s izoplochou čo najrýchlejšie, pričom sú tolerované určité nepresnosti.

Najjednoduchšou aproximačnou metódou je algoritmus, ktorý nachádza priesečník v každej bunke, ktorej hodnoty obkolesujú hodnotu určujúcu povrch izoplochy. Poloha priesečníka je potom určená ako stred medzi vstupom a výstupom lúča z bunky. Aj keď je táto metóda veľmi rýchla, vznikajú pri nej artefakty veľkosti bunky a preto sa v praxi takmer nepoužíva.

Oveľa lepšie výsledky poskytuje metóda, ktorá vypočíta hodnoty DF v bode vstupu a výstupu lúča do bunky a pomocou lineárnej interpolácie týchto dvoch hodnôt nájde priesečník s izoplochou. Takýmto spôsobom dosiahneme oveľa lepšie výsledky ako v predchádzajúcej metóde. Napriek tomu táto metóda nedokáže nájsť priesečník s izoplochou, pre ktorú má rovnica 4-6 dva korene a hodnoty DF vo vstupnom a výstupnom bode sú obidve väčšie alebo menšie ako ρ_{iso} .

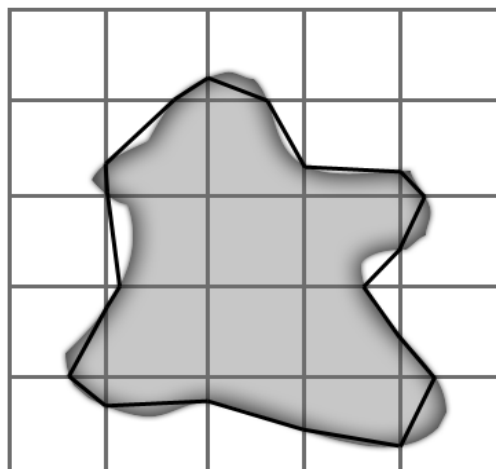
Ďalšou možnosťou je použitie Neubauersovej metódy, ktorá funguje na základe predchádzajúcej. Najprv je vypočítaný priesečník pomocou lineárnej interpolácie ako v predchádzajúcom prípade. V tomto bode je následne spočítaná trilineárnou interpoláciou hodnota DF. Ďalej je pôvodný lúč rozdelený na dve časti a na základe hodnôt je určené miesto potenciálneho priesečníka. Na túto polovicu je potom algoritmus aplikovaný rekurzívne. Riešenie je síce presnejšie ako v predchádzajúcom prípade, ale nerieši problém, keď nie je priesečník vôbec zistený. Pribúda dokonca problém, keď algoritmus v prípade výskytu troch priesečníkov vráti posledný.

Všetky tieto problémy rieši algoritmus publikovaný v [15]. V prvom kroku vypočíta extrémy polynómu 4-6. Tieto dva extrémy rozdelia lúč v rámci bunky najviac na tri časti, ktoré sú postupne prechádzane od začiatku do konca, pričom v ich koncoch sú vypočítavané hodnoty polynómu 4-6. Keď je nájdený interval, platí pre neho že obsahuje práve jeden koreň, obsahuje nulu, neobsahuje extrém, koreň je v intervale t_{in}, t_{out} a je to prvý koreň na ceste lúčom cez bunku. Potom je v tomto intervale určená približná poloha priesečníka pomocou Neubauersovej metódy. Takýto algoritmus je oveľa rýchlejší ako presné riešenie vypočítané pomocou Schwarzovej metódy, napriek tomu poskytuje veľmi dobré výsledky.

4.2 Nepriame zobrazovanie izoplochy – prevod izoplochy na množinu trojuholníkov

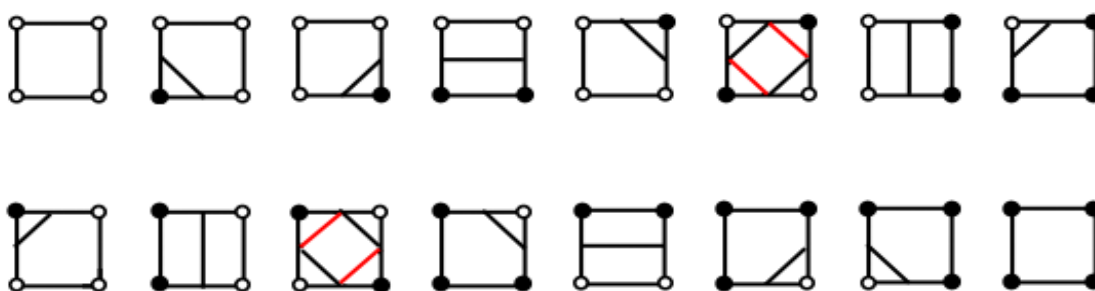
Jednou z najrozšírenejších reprezentácií 3D telies primárne určených na zobrazenie je v súčasnosti množina trojuholníkov. Súvisí to najmä s jednoduchými a rýchlymi algoritmami, pomocou ktorých je možné zobrazovať veľké množstvo trojuholníkov v reálnom čase. V neposlednom rade, takmer každý osobný počítač obsahuje grafickú kartu, schopnú tieto úlohy ešte viac urýchliť. Snaha previesť izoplochu na množinu trojuholníkov je preto veľmi logickým krokom. V súčasnosti existuje viacero metód, ktoré tento prevod dokážu efektívne vykonať. Oproti priamym zobrazovacím technikám majú tieto algoritmy aj niekoľko nevýhod. Nie je jednoducho možné zobrazovať rôzne fyzikálne vlastnosti povrchov, algoritmy väčšinou pracujú nad komplikovanými dátovými štruktúrami a pri potrebe zobraziť izoplochu v inej hĺbke (hodnota DF $f \neq 0$) je nutné vygenerovať celú trojuholníkovú sieť odznova.

4.2.1 Marching Cubes – MC (Pochodujúce kocky)



Obrázok
4-4 Aproximácia obvodu telesa v 2D priestore pomocou algoritmu marching squares.

Algoritmus Marching Cubes (MC) vychádza priamo z 2D varianty nazvanej **marching squares**. Marching squares slúži na jednoduchú aproximáciu zobrazenia $f: \mathbb{R}^2 \rightarrow \mathbb{R}$, ktorá nadobúda podobne ako izoplocha hodnotu 0 na obode, záporné hodnoty vo vnútri a kladné mimo telesa. Celý priestor si v tomto prípade rozdelíme na množinu susediacich štvorcov (viď. obrázok 4-4). Následne je možné pomocou lineárnej interpolácie polôh vrcholov a hodnôt zobrazenia f v jednotlivých štvorcov vrcholoch spočítať približné priesečníky telesa s hranami. Na detekciu hrán, ktoré obsahujú priesečník telesa s hranami stačí jednoduchá tabuľka, obsahujúca všetky možnosti. V tomto prípade ich je 16 (viď. obrázok 4-5). Biely kruh znamená zápornú hodnotu zobrazenia f , čierny naopak kladnú. Výsledný obraz je teda množina takto vytvorených úsečiek.



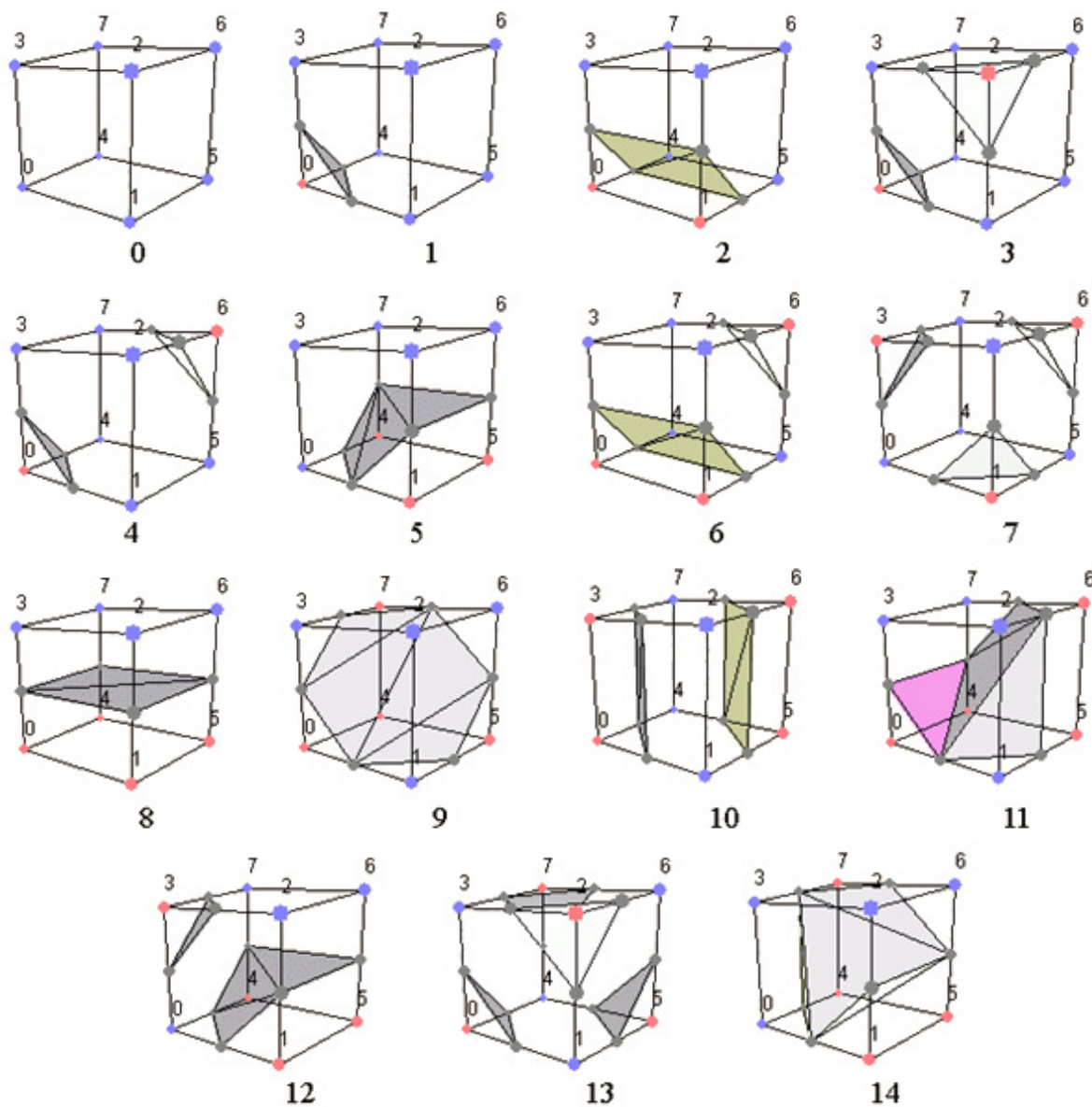
Obrázok
4-5 Možnosti aproximácie funkcie vrámci štvorca (marching squares)

Algoritmus **Marching Cubes** (MC) prvýkrát publikovali v roku 1987 Lorensen a Cline. Slúži na transformáciu skalárneho poľa $f: \mathbb{R}^3 \rightarrow \mathbb{R}$, konkrétne povrchu, pre ktorý platí $f(a) = 0$ na množinu trojuholníkov (všeobecne môžeme zobrazovať povrch $f(a) = b$, ale z praktického hľadiska v tom nie je žiaden rozdiel). Jeho hlavnou myšlienkou je rozdelenie

3D priestoru na bunky zložené z kociek. Tieto vyplňajú celý priestor, ktorý chceme zobraziť. Izoplocha je následne rekonštruovaná v rámci každej bunky zvlášť. Podobne ako v prípade marching squares, sú jednotlivé možnosti ako triangulovať kocku uložené v tabuľke. Stačí teda spočítať index do tabuľky, interpolovať hodnoty vrcholov a výsledkom je 0 až 4 trojuholníkov v rámci jednej kocky, ktoré spoločne s trojuholníkmi z ostatných kociek aproximujú povrch izoplochy.

Jednotlivé kroky algoritmu:

1. Vytvorenie pravouhlej mriežky
 2. Získanie hodnôt vo vrcholoch
 3. Výpočet indexu do tabuľky jednotlivých prípadov
 4. Interpolácia polohy vrcholov pomocou hodnôt zobrazenia f
 5. Výpočet normál.
1. *Vytvorenie pravouhlej mriežky.* Prvým krokom je definovanie pozícií vrcholov kociek. Tieto určujú aký kvalitný bude výsledok. Pri zvolení veľmi hrubej mriežky môže dôjsť k veľkým chybám aproximácie až k chybným výsledkom v topológii. Veľmi jemná mriežka naopak redukuje riziko vzniku chyby, ale generuje aj oveľa väčšie množstvo trojuholníkov. Všeobecne nie je nutné aby, boli vrcholy od seba rovnako vzdialené a dokonca nemusí byť mriežka ani pravouhlá. Pri zachovaní určitých vlastností je možné algoritmus použiť bez väčších zmien aj v tomto prípade.
 2. *Získanie hodnôt vo vrcholoch.* V tomto prípade ide iba o získanie hodnoty skalárneho poľa f vo vrchole v . Samotný algoritmus MC v tomto prípade nehrá žiadnu rolu, je však dobré si uvedomiť, že všeobecne môže byť získanie tejto hodnoty časovo náročné, a preto by sa mala každá implementácia vyvarovať opakovaných žiadostí o hodnoty v tom istom bode.
 3. *Výpočet indexu do tabuľky jednotlivých prípadov.* Indexom v tejto tabuľke je binárne číslo. Jednotka uložená na pozícii i hovorí v tomto prípade o kladnej hodnote zobrazenia f vo vrchole i , nula zase o zápornej hodnote. Číslo i určuje i -tý vrchol v kocke, teda môže nadobúdať hodnoty od 0 po 8. Výsledkom je 8-bitové číslo. Ak je jeho hodnota 0 alebo 255 znamená to, že celá kocka sa nachádza vo vnútri alebo von, preto sa môžu nasledujúce kroky v tomto prípade preskočiť. Tabuľka má teda 256 riadkov. Na každom z nich je uložený zoznam hrán kocky, ktoré izoplocha pretína. Tento zoznam je možné reprezentovať opäť pomocou 8-bitového čísla. Všetky prípady priesečníkov izoplochy s hranami kocky je možné odvodiť z 15 základných prípadov (obrázok 4-6) s použitím rotácie a inverzie.



Obrázok
4-6 Možnosti triangulácie rôznych situácií v marching cubes

4. *Interpolácia polohy vrcholov pomocou hodnôt zobrazenia f .* V kroku 3. sme určili, cez ktoré hrany prechádza izoplocha. Pre úplnú rekonštrukciu je však potrebné vypočítať presné body priesečníkov. Tie je možné získať napríklad lineárnou interpoláciou:

$$P = A + (B - A) \frac{-f(A)}{f(B) - f(A)}$$

P je výsledný bod. A a B sú vrcholy kocky, medzi ktorými vedie hrana a $f(A)$ a $f(B)$ sú hodnoty daného zobrazenia v týchto bodoch. Ďalšou možnosťou je umiestnenie bodu do stredu hrany, prípadne do najbližšieho suseda. Tieto aproximácie v praxi však nemajú veľké uplatnenie.

5. *Výpočet normál.* Pre správne zobrazenie jednotlivých trojuholníkov je nutné poznať normály vo všetkých vrchoch zobrazovaných trojuholníkov. Existuje viacero možností ako normály vypočítať.

Najjednoduchšou, ktorá pritom dáva veľmi pekné výsledky je výpočet gradientu priamo vo vrchole trojuholníka:

$$N_x = \frac{f(x + \Delta, y, z) - f(x - \Delta, y, z)}{\Delta}$$

$$N_y = \frac{f(x, y + \Delta, z) - f(x, y - \Delta, z)}{\Delta}$$

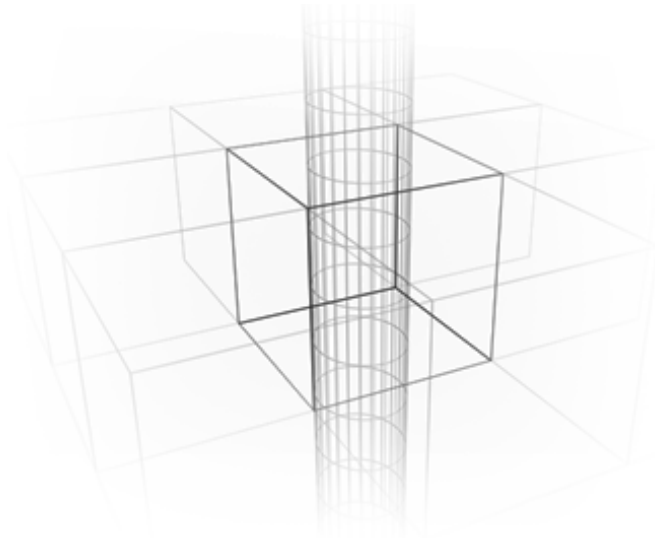
$$N_z = \frac{f(x, y, z + \Delta) - f(x, y, z - \Delta)}{\Delta}$$

4-8

N_x, N_y, N_z sú jednotlivé zložky normály. Δ je dostatočne malé číslo, pomocou ktorého je možné zachytiť správanie sa zobrazenia f na okolí vrcholu trojuholníka. Jedinou nevýhodou je nutnosť poznať hodnoty zobrazenia f v ďalších šiestich bodoch, čo môže viesť pri implementácii k veľkému spomaleniu algoritmu.

Ďalšou možnosťou je využitie hodnôt zobrazenia f vo vrchoch kociek. Pomocou nich je možné spočítať približné normály pre tieto vrcholy a interpoláciou podobne ako v bode 4. vypočítať normály pre vrcholy trojuholníkov.

Existuje aj riešenie, ktoré využíva iba pozície vrcholov trojuholníkov. V tomto prípade je vypočítaná normála každého trojuholníka zvlášť. Následne je normála pre vrchol trojuholníka vypočítaná ako priemer normál všetkých trojuholníkov, ktoré majú daný vrchol spoločný. Treba si však uvedomiť, že takýmto spôsobom nie je možné určiť či jednotlivé normály smerujú do vnútra alebo von z telesa. Toto je možné zistiť pomocou ďalšieho bodu, ležiaceho mimo trojuholník, pričom poznáme hodnotu zobrazenia f v tomto bode.

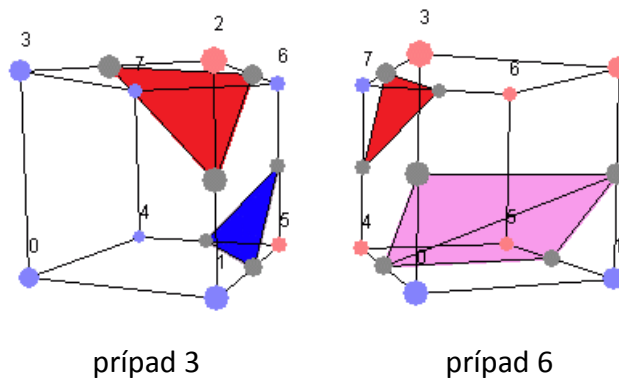


Obrázok

4-7 Nesprávne detekovanie povrchu pri použití marching cubes

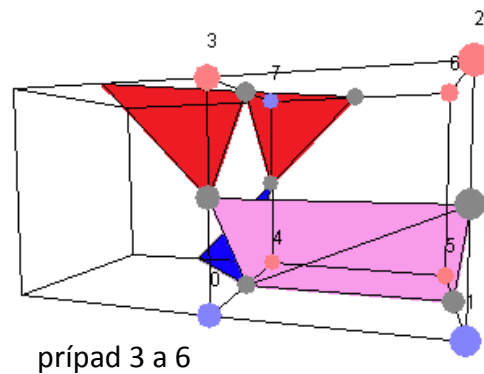
Teleso prechádza stredom kocky. Všetky vrcholy sú považované za body mimo telesa. Povrch teda nie je vôbec rekonštruovaný.

Algoritmus postupne prechádza jednu kocku za druhou a generuje trojuholníkovú sieť. Kvalita výstupu pritom závisí najmä na hustote mriežky. Bohužiaľ so zväčšujúcou sa hustotou narastá aj počet trojuholníkov. Okrem toho existujú aj nejednoznačné prípady, kde môžu vznikať chyby v topológii rekonštruovaného telesa. Najbežnejším problémom pri zvolení príliš riedkej mriežky je možnosť výskytu celej uzatvorenej časti rekonštruovaného telesa vo vnútri jednej kocky. V takomto prípade majú všetky vrcholy kocky kladné hodnoty funkcie f a tým pádom sa nevygeneruje ani jeden trojuholník (viď. obrázok 4-7).

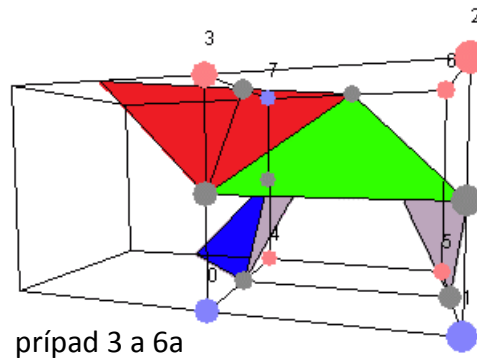


Obrázok

4-8 Špeciálny prípad zlej náväznosti pri použití marching cubes



Obrázok
4-9 Topologicky inkonzistentné naviazanie dvoch triangulácií pre prípad 3 a 6.



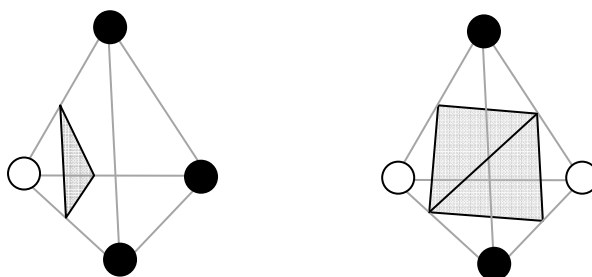
Obrázok
4-10 Topologicky konzistentné riešenie prípadu z obrázku 4-9, s použitím alternatívnej triangulácie prípadu 6

Na obrázku 4-8 vidno dva prípady triangulácie, ktoré majú rovnaké ohodnotenie vrcholov na jednej strane kocky a teda môže nastať situácia, kde budú stáť vedľa seba. Obrázok 4-9 však znázorňuje, že daná triangulácia vygeneruje trojuholníky, ktoré na seba nebudú nadväzovať a v rekonštruovanom povrchu vzniknú diery. V takomto prípade existuje riešenie s použitím alternatívnej triangulácie 6a, pri ktorej ostane pôvodná topológia zachovaná. Správna triangulácia je zobrazená na obrázku 4-10.

Kvôli tomuto problému vzniklo viacero modifikácií pôvodného algoritmu. Väčšina z nich je založená na rozšírenej tabuľke možností a dôkladnejšom preskúmaní každej kocky. Takto upravené návrhy zaručujú topologickú konzistentnosť vygenerovanej siete trojuholníkov s pôvodnými dátami.

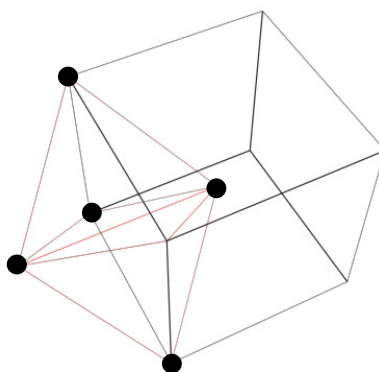
4.2.2 Marching tetrahedra – MT (Pochodujúce štvorsteny)

Ďalšou možnosťou ako sa vyhnúť problémom topologickej inkonzistencii je rozdelenie kocky na 5 štvorstenov. Priesečníky izoplochy sa potom vypočítavajú s hranami jednotlivých štvorstenov. Kocku je možné rozdeliť na daných 5 štvorstenov dvoma spôsobmi. Kvôli náväznosti jednotlivých trojuholníkov je nutné tieto dva rozdelenia striedať. Pri triangulácii štvorstenu existujú len 2 možnosti (viď. obrázok 4-11), ostatné sa dosiahnu rotáciou alebo inverziou, tým sa veľmi zredukuje tabuľka používaná na určovanie situácií.



Obrázok
4-11 Obidve možnosti triangulácie štvorstenov

S elegantným riešením prišli Chan a Purisma [16], ktorí navrhli rozdeliť priestor na štvorsteny, ktorých vrcholy budú ležať na vrcholoch a v strede kocky. Každý takýto štvorsten má dva vrcholy, ktoré sú totožné s vrcholmi kocky ležiacimi na jednej hrane a dva, ktoré ležia v stredoch dvoch susedných kociek. Toto rozloženie ilustruje obrázok 4-12. Takýmto spôsobom sa vyriešila nutnosť striedať dva rôzne rozloženia. Všetky štvorsteny sú rovnaké aj ich rozloženie v priestore je pravidelnejšie ako v prípade rozdelenej kocky. Povrch nimi generovaný je hladší a lepšie aproximuje skutočný povrch izoplochy. Veľkou nevýhodou však stále ostáva veľké množstvo generovaných trojuholníkov, ktoré presahuje dokonca aj množstvo trojuholníkov generovaných pomocou MC.



Obrázok
4-12 Vrcholy štvorstenov pri použití tzv. center cubic lattice

4.2.3 Regularised marching tetrahedra – RMT (Regularizované pochodujúce štvorsteny)

Algoritmus MC produkuje v priemere 3 trojuholníky na jednu kocku. V prípade, ak sú použité sofistikovanejšie metódy zaručujúce topologickú konzistenciu medzi pôvodnými a rekonštruovanými dátami, počet trojuholníkov sa ešte zväčší. Ďalším problémom je možnosť výskytu trojuholníkov veľmi rozdielných veľkostí, ktoré zhoršujú vizuálny vnem zo zobrazovanej plochy. Marching tetrahedra, s použitím vhodného rozloženia štvorstenov jednoducho vyriešili problém s topológiou, ale generujú ešte väčšie množstvo trojuholníkov ako MC. Kvôli týmto problematickým vlastnostiam vzniklo veľké množstvo rôznych metód, ktoré sa snažia zjednodušiť už hotovú trojuholníkovú sieť. Využívajú sa pritom hlavne postupy, pri ktorých sú vrcholy trojuholníkov zhlukované do bodov.

Veľké množstvo trojuholníkov by v dnešnej dobe nemusel byť taký problém, oveľa závažnejšie sú veľké rozdiely vo veľkostiach trojuholníkov, spôsobujúce vizuálne chyby.

Algoritmy zjednodušujúce trojuholníkovú sieť znižujú počet aj rozdiely vo veľkostiach trojuholníkov. Takmer všetky však pracujú už len nad samostatnou trojuholníkovou sieťou a nevyužívajú informácie obsiahnuté v samotnej izoploche.

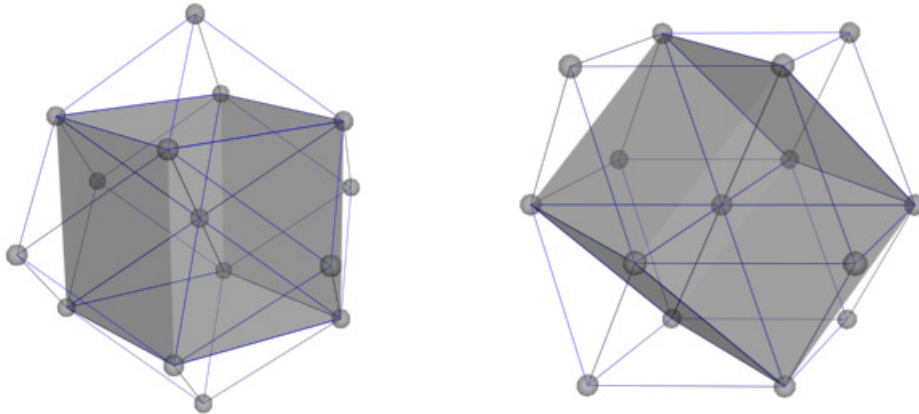
Výhody a vlastnosti RMT

Hlavnou myšlienkou RMT je skombinovať dva jednoduché algoritmy, pochodujúce štvorsteny na vytvorenie trojuholníkovej siete a zhlukovací algoritmus na jej zjednodušenie. Zhlukovací algoritmus je možné vykonať ešte pred samotnou trianguláciou v rámci buniek mriežky a teda je zachovaná topológia pôvodných dát (toto sa dosahuje len veľmi ťažko pri použití metód pracujúcich už na samotnej trojuholníkovej sieti). Výsledkom sú takmer rovnaké trojuholníky, ktoré je možné generovať v rádovo rovnakom čase ako pri MT. V nasledujúcom zozname je výpis výhod, ktoré prinášajú RMT oproti MT alebo MC:

- Zjednodušovanie siete s použitím zhlukovania prebieha pred trianguláciou a využíva pritom pôvodné dáta.
- Narozdiel od väčšiny zjednodušujúcich algoritmov je zachovaná topológia.
- Dodatočný čas na zjednodušenie trojuholníkovej siete je kompenzovaný redukciou počtu vzniknutých bodov a trojuholníkov.
- Vlastnosti trojuholníkov sa dramaticky zlepšia, čo vedie k takmer perfektnému tieňovaniu povrchu.
- Počet trojuholníkov sa typicky zníži až o 70% v porovnaní s pôvodnými MT.
- Algoritmus je schopný vypočítať gradienty a zakrivenia v jednotlivých vrchoch, s použitím dát uložených vo vrchoch mriežky.

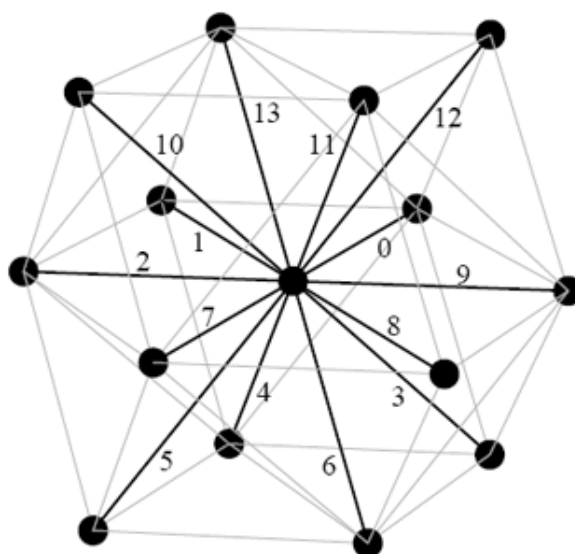
Vzorkovacia mriežka

Prvým krokom je vytvorenie vzorkovacej mriežky, v ktorej vrcholoch budeme zisťovať hodnoty zobrazenia f , ktoré definuje izoplochu. Na naše účely veľmi dobre poslúži mriežka zobrazená na obrázku 4-12. Táto nám dáva oveľa lepšie vzorkovacie rozlíšenie ako pôvodná, ktorá bola založená na rozdelení kocky na 5 štvorstenov. V tomto prípade vznikajú takmer pravidelné štvorsteny, ktoré majú dve protiľahlé steny dĺžky 2 a ostatné dĺžky $\sqrt{3}$ (viď. obrázok 4-13)



Obrázok
4-13 Rozloženie bodov vzorkovacej mriežky v prípade použitia body-centered cubic lattice

Obrázok 4-14 zobrazuje rozloženie a očíslovanie vzorkovacích bodov presne tak, ako sa používa v implementácii algoritmu.



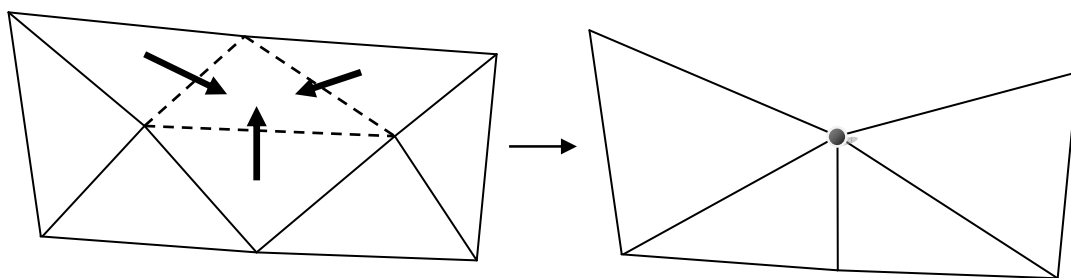
Obrázok

4-14 Číslovanie vrcholov použitých v implementácii algoritmu marching tetrahedra

Číslovanie vrcholov používané pri označovaní priesečníkov. 0..6 sú nové vrcholy pri prechádzaní cez mriežku 7..13 sú tie, ktoré boli preskúmané už predtým. Opačné hrany sa vždy líšia o 7

Označovanie priesečníkov

Podobne ako pri predchádzajúcich algoritmoch sa snažíme zistiť, ktoré hrany pretínajú povrch izoplochy. Tentokrát však neskúmame jednotlivé štvorsteny, ale postupne prechádzame všetky body vzorkovacej mriežky. Každý takýto vrchol má 14 susedov. Pri prechode nám stačí zistiť, či sa nepretína s izoplochou iba 7 nových hrán, ostatných 7 bolo už označených pri prechode predchádzajúcich bodov. Ak zistíme, že hodnoty zobrazenia f majú pre koncové body skúmanej hrany opačné znamienko, označíme na príslušnú pozíciu do bitového poľa najbližšieho vrcholu existenciu priesečníka. V našom prípade, kde je izoplocha definovaná ako množina bodov $I = \{x \in \mathbb{R}^3; f(x) = 0\}$, bude existencia priesečníka uložená do bitového poľa bodu, ktorého absolútna hodnota zobrazenia f je menšia. Keďže každý vrchol mriežky obsahuje takéto bitové pole, značne sú uľahčené ďalšie výpočty.



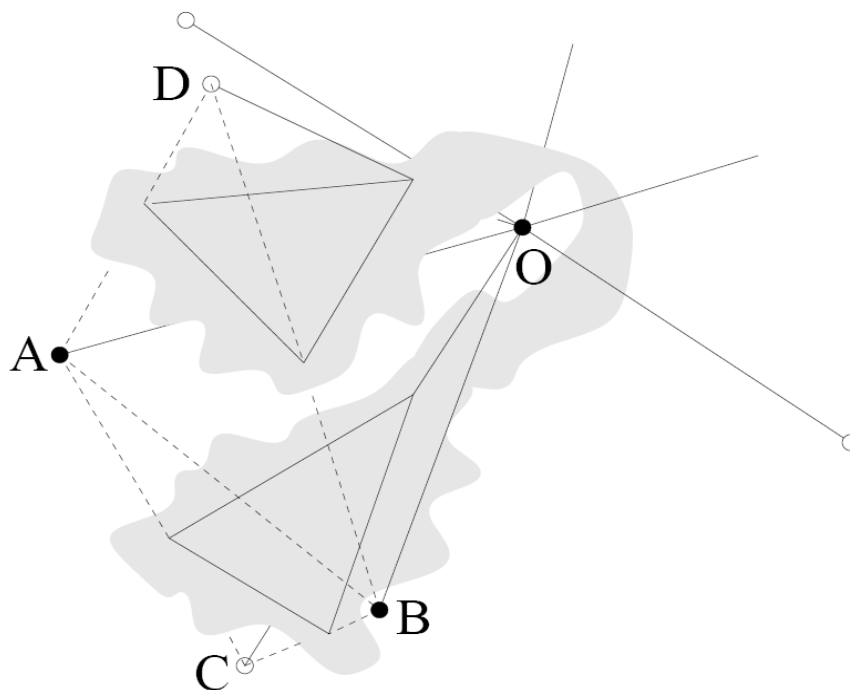
Obrázok
4-15 Zhlukovanie vrcholov (regularised marching tetrahedra)

Zjednodušovanie povrchu. Tučne označený trojuholník je zhlukovaním zmenšený na jeden bod. Z pôvodných 8 trojuholníkov ostanú len štyri, pričom je zachovaná pôvodná topológia.

Zjednodušovanie siete – zhlukovanie vrcholov

Hneď ako sú označené všetky vrcholy, môže sa začať rozhodovať o tom, ktoré z vrcholov budú unifikované v jeden a tým pádom budú eliminované viaceré trojuholníky (obrázok 4-15). V ideálnom prípade by táto situácia mohla nastať pre každý skúmaný bod mriežky. V niektorých situáciách by však mohlo dôjsť k narušeniu pôvodnej topológie, preto je potrebné preskúmať nasledujúce prípady:

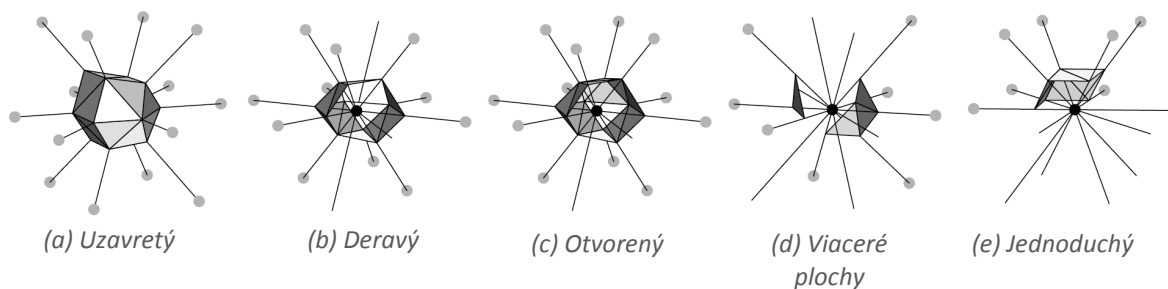
- **Uzavretý povrch** – Všetky susediace body majú opačné znamienko zobrazenia f ako centrálny bod. Povrch izoplochy teda tento bod obklopuje a nemôže byť zhluknutý do jedného vrcholu, pretože by tak celý zanikol. V takomto prípade ostáva celý povrch v okolí tohto bodu nezmenený.
- **Deravý povrch** – V takomto prípade je okolo centrálného bodu jediná plocha, ale s dierami. Ak by sme ho nahradili jediným bodom, tieto diery by zanikli. Opäť teda zhlukovanie nenastáva.
- **Otvorený povrch** – V tomto prípade máme jednu plochu bez dier, ale jej okraje sú rozložené tak, že zhluknutie by mohlo viesť k ich uzatvoreniu, čím by vznikli dve hrany alebo dva trojuholníky otočené proti sebe. Povrch musí byť teda opäť nezmenený



Obrázok
4-16 Kritérium pre otvorený povrch

O je centrálny vzorkovací bod, A až D sú nejaký jeho susedia. Existencia OA a OB bez blízkeho priesečníku s izoplochou a hrán OC a OD s priesečníkom s izoplochou vedie k zrúteniu sa povrchu, ak je spojený v okolí O a ak priesečníky s izoplochou na hranách AD a AC sú blízko bodu A, alebo priesečníky na hranách BC a BD blízko bodu B.

- **Viaceré plochy** – Priesečníky s hranami tvoria viacero separátnych plôch. Každá z nich môže byť zhluknutá do jedného bodu.
- **Jednoduchý povrch** – Najjednoduchšia situácia, kde sa nachádza len jedna plocha, ktorú je možné nahradiť jedným bodom.



Obrázok
4-17 Prípady rôznych topológií vyskytujúcich sa pri zjednodušovaní povrchu

(a) – (c) nemôžu byť zhlukované. (d) a (e) môžu byť zhluknuté do jedného bodu pre každý samostatný povrch.

Kedže informácia o blízkom priesečníku izoplochy s hranou je uložená na príslušnej bitovej pozícii, je možné detekovať všetky patologické prípady, v ktorých nie je možné previesť zhľukovanie iba za pomoci bitových operácií.

Najkomplexnejší je prípad otvoreného povrchu. Obrázok 4-16 popisuje situáciu, kedy nie je možné previesť zhľukovanie. Takto je potrebné otestovať všetkých 36 hrán v okolí centrálného bodu (4-14). Implementácia tejto časti algoritmu je popísaná v sekcii 9.3.

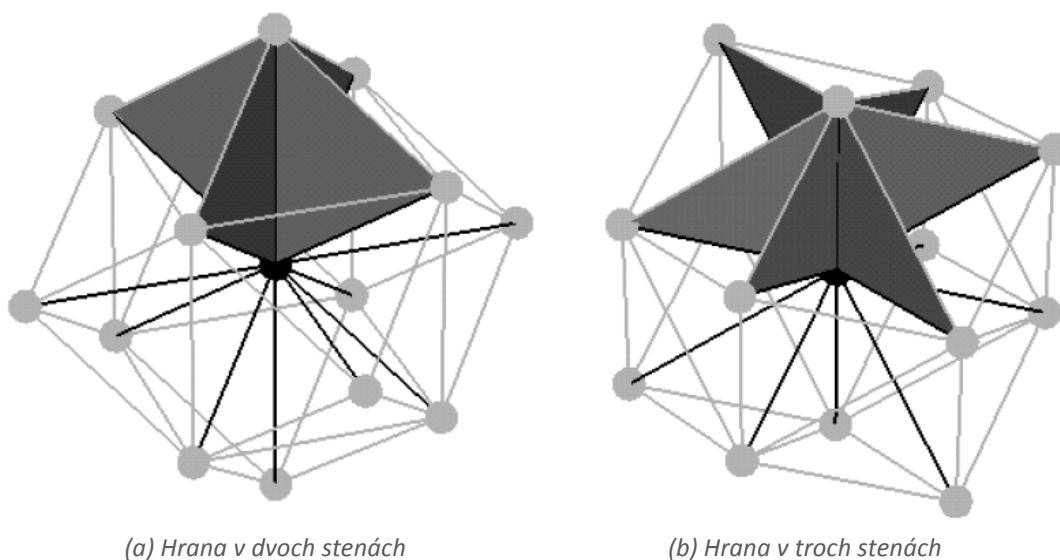
Triangulácia

V okamihu, keď máme vyriešené všetky prípady, keď nie je možné povrch zjednodušiť, môže algoritmus začať vytvárať samostatné trojuholníky. Postupne sa prechádzajú všetky body mriežky. Každý takýto bod obsahuje referenciu na bod priesečníka s izoplochou, ktorý vznikol na susediacej hrane. Jednotlivé body môžu byť uložené v lineárnom zozname. Každý vrchol mriežky patrí do 24 štvorstenov a každý štvorsten obsahuje 4 vrcholy. Tým pádom pri prechode stačí triangulovať iba 6 nových štvoruholníkov. Tieto sa triangulujú pomocou referencií uložených v jednotlivých vrchoch. Obrázok 4-11 zobrazuje jediné dva možné prípady. Ak sú však niektoré referencie totožné, znamená to, že hrana prípadne celý trojuholník bol zhľuknutý do jedného bodu. V takomto prípade sa nevygeneruje nič.

Najjednoduchším spôsobom ako určiť pozíciu vrcholu, ktorý vznikne zhľuknutím viacerých vrcholov je priemer ich pozícií. Už v takomto prípade dostávame oveľa lepšie vizuálne výsledky ako pri MC, alebo MT. To že nastane zhľukovanie je veľmi pravdepodobné a teda v priemere vzniká iba jeden vrchol na jeden vrchol mriežky, ktorý mal označenie priesečníka s izoplochou. Výsledná trojuholníková sieť je oveľa pravidelnejšia a počet trojuholníkov sa dramaticky znížil.

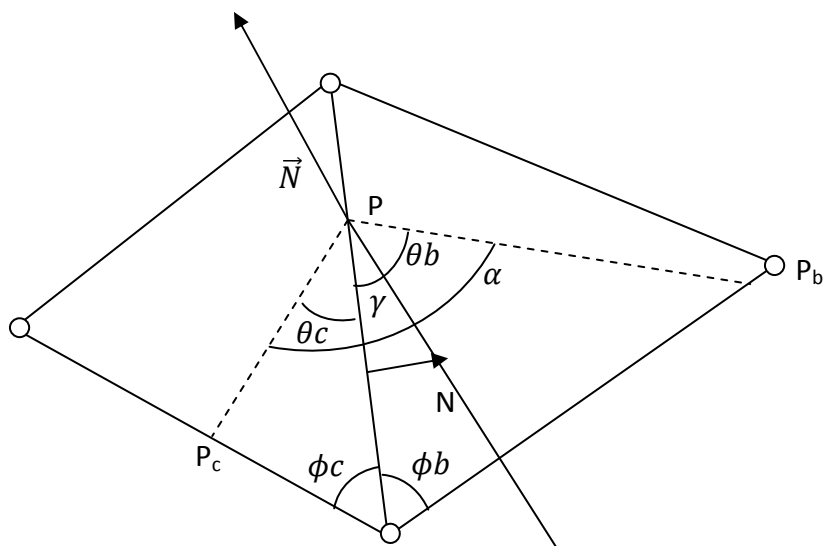
Výpočet normál a zakrivenia

Ako v prípade marching cubes, aj v tomto prípade je potrebné vypočítať normály, pre správne tieňovanie výslednej trojuholníkovej siete. Je možné použiť rovnaké metódy ako v prípade MC, naskytá sa nám však aj jedno komplexnejšie riešenie.



Obrázok
4-18 Dva druhy rôznych hrán vo vzorkovacej mriežke

Štruktúra vzorkovacej siete nám dáva možnosť vypočítať normály s použitím hodnôt zobrazenia f definujúceho izoplochu v bodoch vzorkovacej siete. Normála sa v tomto prípade počíta pre každú hranu, ktorá pretína izoplochu. V našom prípade existujú dva druhy týchto hrán. Prvá skupina sa nachádza na priesečníku dvoch stien a druhá na priesečníku troch stien (viď obrázok 4-18). Normálu získame pomocou výpočtu gradientu v smeroch určených stenami, ktoré danú hranu pretínajú.



Obrázok
4-19 Výpočet gradientu a normály v rámci jednej plochy

Vrcholy o, a, b, c sú vrcholmi vzorkovacej mriežky. Čiarkovaná čiara zobrazuje aproximáciu izoplochy. P_c a P_b sú priesečníky izoplochy s hranami oc a ob . $\vec{N}$ je priemer normál na úsečky $P_c P$ a $P_b P$. Zakrivenie je charakterizované uhlom α . N_x je zložka gradientu v smere definovanom plochou oac . γ je uhol, ktorý zvierá oa a normála $\vec{N}$.

Obrázok 4-19 znázorňuje situáciu pri vypočítavaní zložky gradientu v smere jednej z plôch pretínajúcu hranu oa . Ak sú hodnoty zobrazenia f pre body o, a, b, c sú $d_o \dots d_c$, potom uhly θ_c a θ_b , ktoré zvierajú hrana oa s časťami aproximovanej izoplochy môžeme spočítať pomocou nasledujúceho vzorca:

$$\theta_b = \tan^{-1} \left[\frac{\sin(\phi_b)}{\frac{(d_o - d_b)|oa|}{d_o - d_a|ob|} - \cos(\phi_a)} \right]$$

4-9

$$\theta_c = \tan^{-1} \left[\frac{\sin(\phi_c)}{\frac{(d_o - d_c)|oa|}{(d_o - d_a)|oc|} - \cos(\phi_c)} \right]$$

4-10

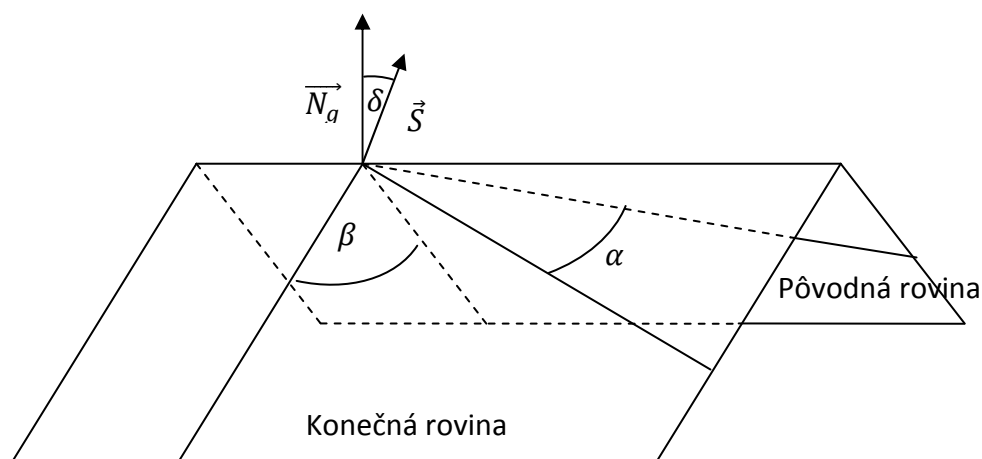
Uhol γ vypočítame ako $\gamma = \left| \frac{\theta_b - \theta_c}{2} \right|$. V tomto prípade nám výraz $\tan(\gamma)$ dáva projekciu normály $\vec{N}$ na rovinu kolmú na úsečku oa v pomere k jednotkovej dĺžke oa .

$$\frac{|N_x|}{|oa|} = \tan(\gamma)$$

4-11

Pre $|oa| = 1$ teda platí $|N_x| = \tan(\gamma)$. Smer vektoru $\vec{N}_x$ je možné zistiť jednoduchým vektorovým súčinom. Pre $\theta_c > \theta_b$ bude $\vec{N}_x = (\vec{ob} \times \vec{oc}) \times \vec{oa}$ v opačnom prípade $\vec{N}_x = -(\vec{ob} \times \vec{oc}) \times \vec{oa}$. Výslednú normálu vypočítame podľa toho, koľko stien prechádza cez úsečku oa (vid'. obrázok 4-18) V prípade, že ide o 2 steny, stačí spočítať projekcie $\vec{N}_x$ a jednotkový vektor v smere $\vec{oa}$. V prípade troch susedných stien sa jednotlivé zložky vektorov $\vec{N}_x$ vynásobia $\frac{2}{3}$ a opäť sa k nim pripočíta jednotkový vektor v smere $\vec{oa}$. Násobenie $\frac{2}{3}$ vychádza z toho, že jednotlivé steny v tomto prípade zvierajú uhol 60° . V prípade, ak vzorkovacia mriežka nespĺňa túto vlastnosť, takýmto postupom vznikajú odchýlky od normály.

Zakrivenie izoplochy v rámci danej roviny vyjadruje potom uhol $\alpha = \theta_b + \theta_c$. Na vypočítanie reálneho zakrivenia izoplochy v danom bode však potrebujeme ďalší výpočet, pretože rovina, v ktorej sme získali α , nezvierá pravdepodobne s izoplochou 90° uhol.



Obrázok
4-20 Výpočet zakrivenia izoplochy

. Pôvodne spočítaný uhol α je potrebné transformovať do výslednej roviny, ktorá je definovaná normálou $\vec{N}$.

Keď je α zakrivenie v pôvodnej rovine, $\vec{N}_g$ je normála na izoplochu a δ je uhol, ktorý zvierá normála $\vec{N}_g$ s rovinou, ktorá je kolmá na pôvodnú rovinu a na úsečku oa , potom pre výsledné zakrivenie β platí:

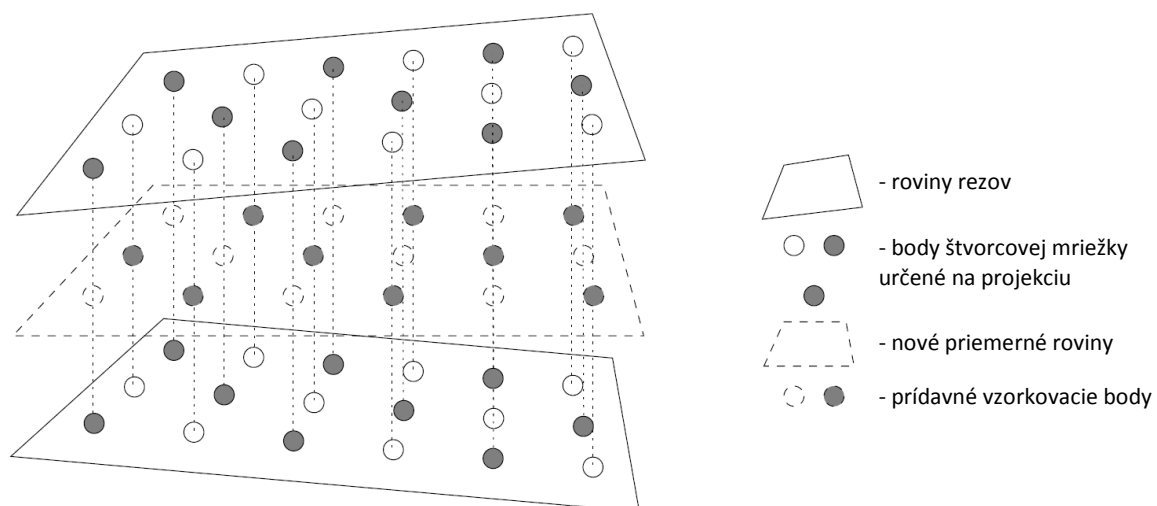
$$\frac{1}{\tan^2\left(\frac{\beta}{2}\right)} = (1 - \cos^2 \delta) \left(\frac{1}{\sin^2\left(\frac{\alpha}{2}\right)} - 1 \right)$$

4-12

Pre nás je dôležité maximálne zakrivenie, teda minimálna hodnota $\left| \tan^2\left(\frac{\beta}{2}\right) \right|$ pre každú plochu, v ktorej sme zakrivenie počítali.

Využitie hodnôt zakrivenia

Výpočet pozície pri zhlukovaní vrcholov pomocou priemernej pozície funguje veľmi dobre pre hladké povrchy. Nevhodne však zjemňuje ostré hrany. Do výpočtu pozície novej pozície vrcholu je preto vhodné začleniť hodnotu zakrivenia izoplochy ostatných vrcholov. Vhodným riešením je vážený priemer, kde váhy vypočítame ako $\omega_i = \frac{1}{\left| \tan^2\left(\frac{\beta}{2}\right) \right|_{\min}}$ a výslednú pozíciu $p_0 = \frac{1}{\sum_{i \in S} \omega_i} \sum_{i \in S} \omega_i p_i$, kde S je množina vrcholov, z ktorých počítame priemer a p_i sú pozície jednotlivých bodov.



Obrázok
4-21 Adaptovanie vzorkovacej mriežky pre neparalelné rezy

Modifikácie

Pri použití 3D ručného ultrazvuku sú výstupom neparalelné, avšak sekvenčné rezy. V takomto prípade je možné upraviť vzorkovaciu mriežku tak, aby lepšie s danými rezmi splývala. Jednotlivé vrcholy z obrázku 4-13 sú premietané po priemernej normále dvoch susedných rezov. Týmto spôsobom je možné získavať hodnoty priamo z jednotlivých rezov a nie je potrebné ich interpolovať. V praxi sa však rezy môžu nachádzať relatívne ďaleko od seba a je preto potrebné vytvárať ďalšie roviny (obrázok 4-21).

Pri takejto modifikácii je však dobré uvedomiť si, že výpočty normál spomínané v sekcii 0 nebudú presné kvôli deformácii vzorkovacej mriežky.

5 Implementácia

Cieľom navrhnutého programu bolo vytvoriť systém schopný porovnávať rôzne rekonštrukčné metódy. Implementované boli viaceré algoritmy popísané v kapitolách 3 a 4 spolu s ďalšími doplnkovými algoritmami určenými na získavanie vstupných dát a meranie dát výstupných. Aplikácia bola naprogramovaná v C++ s použitím WIN API, OpenGL a bola testovaná na operačnom systéme Windows Vista Business Edition 32-bit. Program bol pomenovaný RPZNR.

5.1 Štruktúra programu

Program bol navrhnutý tak, aby umožňoval skúmať rekonštruovanie povrchov z rezov rôznych modelov. Skladá sa z troch hlavných celkov. Prvým je množina tried a funkcií, ktoré dokážu načítať model vytvorený v externom programe, vytvoriť jeho rezy a ďalej ich spracovať. Takto získané rezy sú poskytnuté druhej množine tried, ktoré z nich rekonštruujú povrch. Posledná množina zabezpečuje interakciu medzi užívateľom a celým programom. Celá hierarchia je zobrazená na diagrame 5-1.

1. Spracovanie vstupných dát

CModel – slúži na načítanie vstupného modelu zo súboru, uloženie týchto geometrických dát v pamäti a ich zobrazenie.

CCrossSections – z dát uložených v CModel vytvára množinu ľubovoľných rezov, tieto rezy obsahujú mapy vzdialeností.

2. Rekonštrukcia povrchu

Reconstructors – vytvárajú implicitné rezy pomocou CCrossSections a zabezpečujú výpočet distance function pomocou Interpolators.

Interpolators – interpolujú hodnoty z dvoch rovín rezov.

Visualisators – rekonštruujú izoplochu pomocou distance function z Reconstructors. Výstupom je množina trojuholníkov.

3. Užívateľské rozhranie

Windows – zahŕňa triedy obsluhujúce dialógové okná, poskytujúce užívateľské rozhranie programu.

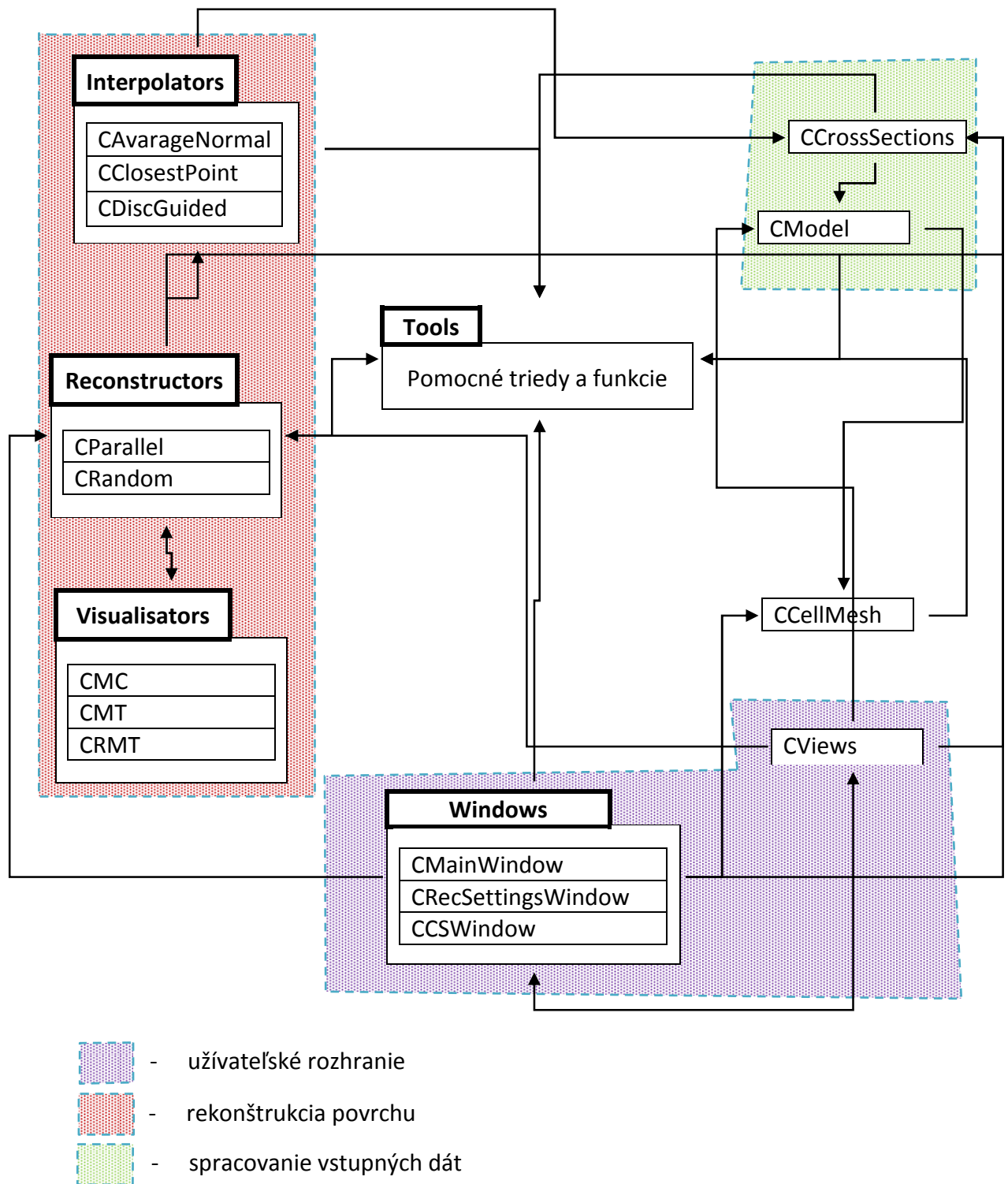
CViews – zobrazuje vstupný, výstupný model a rezy v hlavnom okne programu.

4. Ostatné

CCellMesh – slúži na výpočet objemov modelov.

Tools – zahŕňa množinu pomocných tried a funkcií.

Všetky triedy, ktoré majú v programe práve jednu inštanciu sú potomkami triedy CSingleton, ktorá pomocou statických premenných existenciu jednej inštalácie zaručuje. Pred používaním je potrebné triedu inicializovať zavolaním metódy create a na konci ju uvoľniť metódou destroy. Použitie je potom takéto: `nazov_triedy::GS().nazov_metody`



Obrázok
5-1 Štruktúra programu

5.2 Pomocné triedy a funkcie (Tools)

Kedže navrhovaná aplikácia pracuje z veľkej časti s grafickými dátami, prvými implementovanými triedami boli triedy **CVector4** a **CMatrix4x4**. Tieto slúžia na základnú prácu s vektormi a maticami.

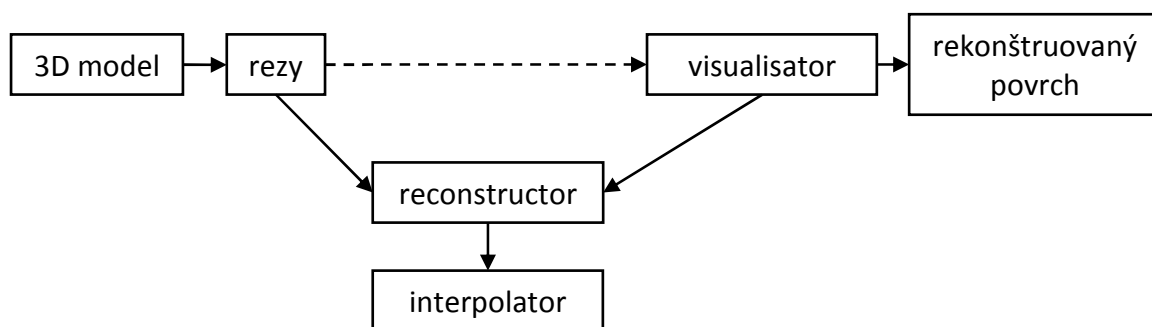
Ďalšou pomocnou triedou je trieda **CTimer**, ktorá slúži na meranie času, ktorý je potrebný na vykonanie jednotlivých častí programu. Trieda obsahuje mapu `std::map<std::string, SProfiler*>`, ktorá uchováva časové profily, priradené k menám. Meranie času sa potom vykonáva pomocou funkcií `TimerStartProfile(char *szName)` `TimerEndProfile(char *szName)`, ktoré sa umiestnia na začiatok a koniec meranej časti.

Zápis do logovacieho súboru, prípadne na obrazovku pomocou štandardného windows message boxu zabezpečuje trieda **CLog**. Výpis sa potom vykonáva metódou `CLog::GS().Write`.

Poslednou pomocnou triedou je **CArcBall**, ktorá slúži na jednoduché ovládanie pohľadu na modely vo výstupnom okne. Pomocou kliknutia myši je uchopený bod na povrchu gule a následným pohybom je táto guľa otáčaná a s ňou aj všetky modely v scéne.

5.3 Postup výpočtu rekonštrukcie

Celý rekonštrukčný proces je zložený z viacerých krokov. Najprv sa načíta vstupný 3D model, ktorý je následne rozrezaný na jednotlivé rezy. Vizualizačnej triede zobrazujúcej izopovrch sa nastaví odkaz na rekonštrukčnú funkciu, ktorá pre ľubovoľný bod v priestore určuje, z ktorých dvoch rezov sa bude hodnota distance function interpolovať a pomocou odkazu na interpolačnú funkciu vráti vizualizačnej triede jej presnú hodnotu. Takýmto spôsobom vizualizačná trieda opakovane žiada rekonštrukčnú funkciu o hodnoty distance function a postupne rekonštruuje povrch pomocou trojuholníkovej siete. Postup je znázornený na diagrame 5-2



Obrázok
5-2 Postup výpočtu rekonštrukcie

5.4 Spracovanie vstupných dát

Keďže práca nebola nijako previazaná s medicínskym zázemím, všetky zdrojové dáta bolo potrebné získavať iným spôsobom ako v reálnej praxi, pričom bolo nutné zachovať možnosť porovnania zdroja s výsledkom. Preto bolo ako najvhodnejšie riešenie vybrané načítanie trojrozmerného modelu vytvoreného v externom programe a jeho následné „rozrezanie“. Za vstupný formát bol zvolený textový formát ASE, kvôli jednoduchosti jeho štruktúry. Modely sú v takomto súbore uložené striktne ako množiny trojuholníkov.

5.4.1 Formát ASE súboru

ASE je textový súbor, zložený z tagov vo formáte:

```
*MENO_TAGU parametre oddelené medzerami {  
    množina ďalších tagov  
}
```

pričom zložené zátvorky sú nepovinné u tagov neobsahujúcich žiadne podtagy. Pre naše účely nebolo potrebné skúmať všetky časti súboru. Štruktúra skúmanej časti súboru potom vyzerá takto:

```

...
*GROUP "názov skupiny" {
    *GEOMOBJECT {
        ...
        *MESH {
            ...
            *MESH_VERTEX_LIST {
                ...
                *MESH_VERTEX parametre
                *MESH_VERTEX parametre
                ...
                ...
            }
            *MESH_FACE_LIST {
                *MESH_FACE parametre
                *MESH_FACE parametre
                ...
                ...
            }
            *MESH_NORMALS {
                *MESH_FACENORMAL parametre
                *MESH_VERTEXNORMAL parametre
                *MESH_VERTEXNORMAL parametre
                *MESH_VERTEXNORMAL parametre
                ...
            }
        }
    }
}

```

Tri bodky označujú buď nepodstatnú časť, alebo ďalšie dáta. `MESH_VERTEX_LIST` definuje množinu všetkých vrcholov, pričom parametre `MESH_VERTEX` sú: poradové číslo vrcholu nasledované súradnicami x , y , a z . Pretože ASE súbory boli vytvorené v programe, ktorý používa inú súradnicovú sústavu, súradnice sú v programe najprv transformované podľa nasledujúcich rovníc: $x = x'$, $y = z'$, $z = -y'$. `MESH_FACE_LIST` určuje množinu trojuholníkov. Jeden trojuholník sa skladá z troch vrcholov, na ktoré odkazujú parametre tagu `MESH_FACE`, ktoré sú vo formáte: číslo trojuholníka: A: číslo prvého vrcholu B: číslo druhého vrcholu C: číslo tretieho vrcholu a nasledujú ďalšie nepoužívané parametre. Posledné spracované dáta sú normály trojuholníkov a jednotlivých vrcholov. Tieto nemajú žiadne využitie pri vytváraní rezov a slúžia iba na lepšie (tieňované) zobrazenie vstupného modelu.

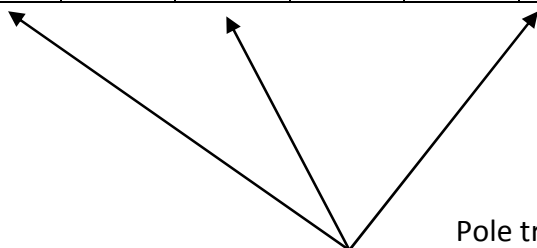
5.4.2 Načítanie vstupného modelu (CModel)

Načítanie dát zo súboru ASE má na starosti trieda `CModel`. Základom sú parsovacie metódy začínajúce prefixom `ParseASE`, ktoré vyšetrujú jednotlivé tagy. V prípade zloženého tagu sa v rámci jednej metódy volá rekurzívne ďalšia, neznáme tagy sú ošetrené metódou `ParseASEUnknown`. Načítanie modelu sa potom vykoná zavolaním metódy `LoadASE`.

Jednotlivé trojuholníky sú uložené v dátovej štruktúre `SMesh`, ktorá obsahuje polia vrcholov, trojuholníkov a normál. Pole trojuholníkov obsahuje v každom prvku číselné indexy vrcholov, z ktorých sa skladá. Pole normál definuje pre každý vrchol pre jednotlivé trojuholníky normály tak, ako sú uložené v poly trojuholníkov (obrázok 5-3).

Pole vrcholov:

V0	V1	V2	V3	V4	V5	V6	V7	V8	V9
----	----	----	----	----	----	----	----	----	----



Pole trojuholníkov:

T0	T1	T2	T3
----	----	----	----

N00 N01 N02	N10 N11 N12	N20 N21 N22	N30 N31 N32
-------------	-------------	-------------	-------------

Pole normál:

Obrázok
5-3 Dátová štruktúra `SMesh`

Pri načítaní modelu sa vykonajú ešte ďalšie funkcie. Model je normalizovaný na veľkosť definovanú konštantou `NORMALIZED_MODEL_SIZE` a je umiestnený rovnomerne okolo bodu 0, 0, 0. Ďalej je vytvorený display list pre OpenGL, pre rýchlejšie zobrazovanie. To sa vykoná zavolaním metódy `Draw`.

5.4.3 Vytváranie rezov zo vstupného modelu (`CCrossSections`)

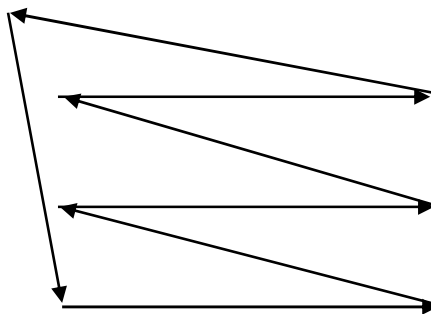
Ďalším krokom spracovania vstupných dát je prechod od 3D modelu tvoreného sieťou trojuholníkov k množine rezových bitmáp definovaných v kapitole 3.2. Na tieto účely slúži trieda `CCrossSections`.

Jednotlivé rezy sú uložené vo vektore `std::vector<SPlane*> PlanesList`. Štruktúra `SPlane` obsahuje viacero premenných charakterizujúcich rez. Najdôležitejšími sú pritom bitmapy definujúce kontúry (`BitBitmap`, `Bitmap`) a matice a vektory definujúce polohu rezu v 3D priestore (`TransfM`, `InvTransfM`, `TranslateV`, `NormalV`, `TransfTranslateV`).

Ľubovoľne orientovaný rez sa vytvára pomocou metódy `AddCrossSectionsPlane`. Výsledná bitmapa definujúca kontúry vzniká postupne s použitím viacerých medzikrokov. Prvým je spočítanie rotačnej matice, ktorá otáča všetky body okolo stredu súradnicovej sústavy tak, aby ležali v jednej rovine (konštantná súradnica `y`). Táto transformácia je aplikovaná na všetky vrcholy vypočítané pri rezaní modelu kvôli značnému zjednodušeniu

výpočtu v 2D priestore. V ďalšom kroku sú vyhľadane všetky priesečníky rezovej roviny z trojuholníkmi 3D modelu. Výsledkom je množina dvojíc bodov tvoriacich úsečky uložených vo vektore `std::vector<SLinePoint> LinePointsArray`, pričom `SLinePoint` je štruktúra obsahujúca jeden bod a pointer na štruktúru `Sline`, ktorá uchováva dva body a informáciu či už bola táto úsečka použitá. Každá úsečka je uložená vo vektore `LinePointsArray` dvakrát. Prvýkrát obsahuje počiatočný bod úsečky a druhýkrát koncový. Takto vytvorený vektor úsečiek je zoradený podľa x-ových súradníc bodov, čím je zabezpečené následné jednoduché vyhľadávanie susedných úsečiek, ktoré môžu byť pospájané do výslednej kontúry rezu. Výsledkom je usporiadaná množina úsečiek, pre ktoré platí, že koncový bod jednej je začiatkovým bodom nasledujúcej. Pomocou vzdialenostnej transformácie popísanej v kapitole 3.2, je takáto množina prevedená na bitmapu vzdialeností. Transformáciu zabezpečujú tri metódy pracujúce nad dvoma dvojrozmernými poľami jediného rezu. `PrepareBorder` rasterizuje kontúry a uloží ich do dvojrozmerného poľa `BitBitmap`, ktoré potom definuje okraje číslom rôznym od 0. Takáto bitmapa je následne zmenená pomocou metódy `FillBitBitmap` tak, aby všetky body vo vnútri kontúry mali hodnotu 1 a mimo 0. Z `BitBitmap` sú nakoniec pomocou metódy `ComputeNearPointCH3x3` Chamferovou transformáciou vypočítané hodnoty vzdialeností uložené do dvojrozmerného poľa `Bitmap`.

Pretože väčšina vstupných modelov neobsahuje žiadne extrémne množstvo trojuholníkov, pri vyhľadávaní priesečníkov medzi rezovou rovinou a modelom sa prehľadávajú všetky trojuholníky. Časová zložitosť je teda v tomto prípade $O(n \cdot m)$, kde n je počet trojuholníkov a m je počet rezových rovín. Následné zoradenie úsečiek v rámci jedného rezu pomocou algoritmu quick sort má v priemernom prípade zložitosť $O(n \cdot \log n)$ kde n je počet úsečiek. Algoritmus napájajúci úsečky môže mať v najhoršom možnom prípade (obrázok 5-4) pre rez obsahujúci jednu kontúru zložitosť až $O(n^2)$. V takejto situácii ležia dve polovice vrcholov nad sebou, teda majú rovnakú súradnicu x . V prípade viacerých kontúr sa môže situácia ešte skomplikovať, keď sa budú x -sové súradnice prekrývať. V praxi sú však rezy generované z modelov, ktoré generujú oveľa jednoduchšie tvary a zložitosť je v takomto prípade lineárna.



Obrázok

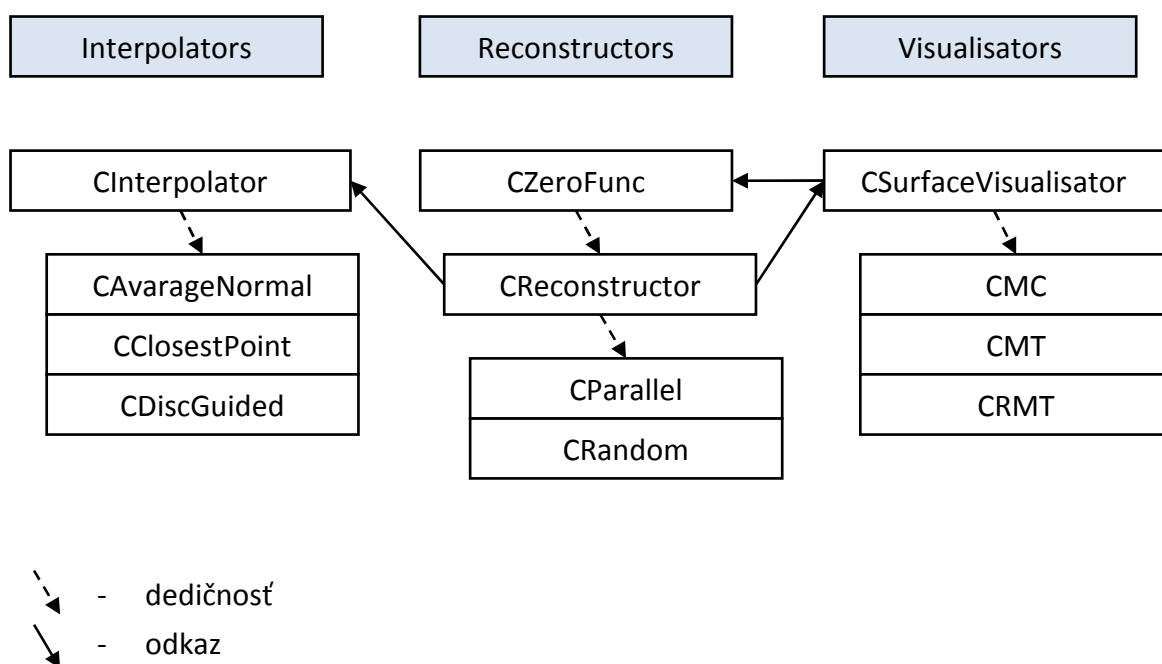
5-4 Najhorší možný prípad rozloženia úsečiek kontúry rezu.

Dôležitou súčasťou `CCrossSections` sú viaceré metódy, ktoré vypočítavajú prechody medzi priestorom bitmáp a priestorom modelu (`PointToPlanef`, `PointToSpacef`, `ConvertPointToPlanef`, `ConvertPointToSpacef`). Pomocou nich je možné

implementovať aj metódu, ktorá vráti hodnotu distance function ležiacu na bitmape pre ľubovoľný bod 3D priestoru, ktorý leží v rovine rezu (`GetPointDist`).

5.5 Rekonštrukcia povrchu

Po spracovaní vstupných dát je možné pristúpiť k rekonštrukcii povrchu. Skladá sa z troch hlavných častí: reconstructors, interpolators a visualisators, ktoré sú vzájomne poprepájané. Základný princíp bol už načrtnutý v kapitole 5.3. Reconstructor zaisťuje pozíciu hlavnej zastrešovacej triedy, ktorá riadi celý proces. Obsahuje odkazy na inštancie tried z interpolators a visualisators a má preddefinované umiestnenia základných rezov. Interpolator vypočítava hodnoty distance function v ľubovoľnom bode priestoru z dvoch bitmáp rezov. Z ktorých rezov sa má interpolovať rozhoduje reconstructor. Visualisator obsahuje odkaz na reconstructor a žiada ho o výpočty hodnôt distance function, z ktorých vytvára geometrickú interpretáciu povrchu. Vzájomná previazanosť tried je načrtnutá na diagrame 5-5.



Obrázok
5-5 Závislosti tried rekonštruujúcich povrch

5.5.1 Reconstructors

Do množiny reconstructors patria tri triedy `CReconstructor`, `CParallel` a `CRandom`. `CReconstructor` je otcovská trieda nasledujúcich dvoch. Celý rekonštrukčný proces pozostáva z postupného volania ich metód v určitom poradí:

Najprv sa vytvorí inštancia jednej zo synovských tried. Tieto dve sa líšia v možnostiach akými môžu byť usporiadané ich rezy. `CParallel` predpokladá paralelné sekvenčne zoradené rezy. `CRandom` nekladie na orientáciu a usporiadanie žiadne požiadavky:

```
CReconstructor * pReconstructor = new CParallel;
```

Môže nasledovať vytvorenie preddefinovaných rezov (ak je tento krok vynechaný, rezy musia byť vytvorené pomocou volania `AddCrossSectionsPlane` triedy `CCrossSections`). Jediným parametrom je v tomto prípade poradové číslo rozloženia rezov:

```
pReconstructor->CreateDefaultCrossSections(1);
```

Pomocou konštánt je určený interpolator a visualisator. V prípade interpolátora sú k dispozícii `CLOSEST_POINT`, `AVERAGE_NORMALS` a `DISC_GUIDED`. Visualisator je zase možné zvoliť z `MARCHING_CUBE`, `MARCHING_TETRAHEDRA` a `REG_MARCHING_TETRAHEDRA`:

```
pReconstructor->SetVisualisator(REG_MARCHING_TETRAHEDRA);  
pReconstructor->SetInterpolator(CLOSEST_POINT);
```

Ďalej sa volá metóda `compute`, ktorá sa postará o skontrolovanie správneho rozloženia rezov:

```
pReconstructor->Compute();
```

Nakoniec je prevedená samotná rekonštrukcia povrchu pomocou predchádzajúcich nastavení:

```
pReconstructor->Polygonize();
```

Výsledný model je zobrazený zavolaním metódy `draw`:

```
pReconstructor->Draw();
```

Pri volaní metódy `Polygonize` žiada jeden z vizualizátorov opakovane o hodnoty `distance function`. Tieto sú mu predávané po zavolaní metódy `GetValue`, ktorá je deklarovaná v triede `CZeroFunc` (ktorá je otcovskou triedou `CReconstructor`) a definovaná v obidvoch rekonštrukčných triedach.

V metóde `GetValue` sa `reconstructor` snaží nájsť dve roviny, z ktorých by bolo vhodné interpolovať hodnoty vzdialeností. Toto je najdôležitejšia vlastnosť rekonštrukčných tried. Samotný výpočet hodnoty z takto nájdenej roviny má na starosti jeden z interpolátorov. V tejto implementácii bolo použité kritérium dvoch najbližších rekonštrukčných rovín k danému bodu, pričom ho musia obkolesovať z rôznych strán.

V prípade `CRandom` sa najprv vyhľadá najbližšia rovina prehľadaním všetkých rezov. Následne je vyhľadávaná druhá najbližšia, pre ktorú platí, že ju nepretína polpriamka vedúca z daného bodu do najbližšieho bodu prvej roviny. Podobným spôsobom pracuje aj `CParallel`. Ten však môže využívať vlastnosť, že rezové roviny sú paralelné a sekvenčne zoradené. Na vyhľadanie dvojice rezov mu teda stačí jeden prechod. Algoritmus je vylepšený

tým, že si pamätá poslednú pozíciu a od nej pokračuje v smere, ktorým prebiehalo prehľadávanie naposledy. Vďaka tejto vlastnosti je časová zložitosť vyhľadania prvej dvojice $O(n)$, kde n je počet rezových rovín. Každá ďalšia dvojica rovín je pri vhodnom volaní `GetValue` (napríklad postupne zdola nahor v smere kolmom na rovinu rezov) získaná v konštantnom čase.

5.5.2 Interpolators

Množinu tried `interpolators` tvorí jedna otcovská a tri synovské triedy: `CInterpolator`, `CClosestPoint`, `CAvarageNormal`, `CDiscGuided`. Prvou volanou metódou týchto tried je `Compute`, ktorá zabezpečí predpočítanie niektorých dát. Potom je možné po zavolaní metódy `Interpolate` vypočítať smer interpolácie a určiť hodnotu `distance function`. V prípade `CClosestPoint`, `CAvarageNormal` ide o jednoduché geometrické operácie, ktoré definujú rovnaký smer interpolácie vo všetkých bodoch tak ako sú popísané v kapitole 3.3.1. Posledná trieda (`CDiscGuided`) implementuje algoritmus popísaný v kapitole 3.4.

Implementácia prvých dvoch tried je úplne priamočiara, posledná je popísaná podrobnejšie.

Súčasťou triedy `CDiscGuided` je aj trieda `CCorrespondenceCalc`, ktorá sa stará o väčšinu výpočtov a obsahuje viacero dátových štruktúr, ktoré sú popísané nižšie.

Inicializačný krok po zavolaní metódy `Compute` jednu z týchto štruktúr naplní. Konkrétne ide o `std::vector<SDiscsPlane*> DiscsPlanesArray`. Štruktúra `SDiscsPlane` obsahuje množinu maximálnych kružníc definovanú v kapitole 3.4.1. O ich výpočet sa postará jej vlastná metóda. Keďže je potrebné prejsť všetky body bitmapy, táto operácia má časovú zložitosť $O(n^2)$, kde n je rozlíšenie bitmapy. Ďalšou štruktúrou je dvojrozmerné pole `PAvrPlane ** AvrPlanes`. Jej hodnoty nie sú na začiatku vypočítavané, pretože nie je možné určiť, pre ktoré dva rezy bude priemerná rovina potrebná. Pre dve rezové roviny s číslami i a j `AvrPlanes[i][j]` určuje priemernú rovinu nachádzajúcu sa medzi nimi. V štruktúre `PAvrPlane` sú potom uložené projekcie všetkých diskov z obidvoch obkolesujúcich rovín a ďalšie premenné definujúce priemernú rovinu.

Pri zavolaní metódy `Interpolate` je najprv zistené, či sú obidve roviny, z ktorých sa bude interpolovať definované. Tento krok je tu preto, lebo metódy z `reconstructors` sa pokúšajú nájsť dve roviny rezov, ale to sa im nemusí vždy podariť. V prípade nekompletnej dvojice nie je možné interpolovať a hodnota `distance function` je vypočítaná iným spôsobom⁹. V našom prípade je to `powf(fDist0 + 1, 2) - 1 + fValue0`, kde `fDist0` je vzdialenosť od rezovej roviny a `fValue0` je hodnota `distance function` v bode rezu najbližšom k skúmanému bodu.

V opačnom prípade sú interpolačné roviny uložené do premenných `iPlane0`, `iPlane1`, `pPlane0` a `pPlane1`. Prvé dve obsahujú poradové číslo roviny, druhé dve odkaz.

⁹ V takomto prípade je možné použiť konštantu, alebo rýchlo klesajúcu funkciu.

V prípade ak je `iPlane0 > iPlane1`, hodnoty aj odkazy sú vymenené, kvôli ošetreniu duplicitných symetrických výpočtov.

Následne je s týmito parametrami zavolaná metóda `CorrespondenceCalc.Calc`. Táto zistí, či už bola inicializovaná a naplnená štruktúra `AvrPlanes[iPlane0][iPlane1]`. Ak áno, nie je potrebné vykonávať žiadne ďalšie kroky. V opačnom prípade je nutné vypočítať premenné definujúce priemernú rovinu, uložiť do nej projekcie diskov zo susedných rezov a vypočítať hodnoty vektorov $\vec{c}_a$. Keďže tieto vektory sú vypočítavané pre každú kružnicu z jednej roviny rezu pomocou korešpondencie s kružnicami druhého rezu, časová zložitosť ich výpočtu je $O(n \cdot m)$, kde n je počet kružníc jednej roviny a m druhaj. Kvôli zjednodušeniu väčšiny výpočtov je najprv vypočítaná matica transformujúca priemernú rovinu do roviny x , z a naspäť. Pomocou tejto matice je jednoduché vypočítať projekcie diskov. Stačí použiť transformačnú maticu a ignorovať súradnicu y , prípadne ju nastaviť na hodnotu transformovanej priemernej roviny. Implementácia výpočtu hodnôt vektorov $\vec{c}_a$ je potom priamočiara.

Pre určenie smeru interpolácie je potrebné vypočítať ešte vektory $\vec{c}_p$. Tento výpočet je veľmi podobný a priamočiary ako v prípade vektorov $\vec{c}_a$. Zabezpečuje ho metóda `CorrespondenceCalc::ComputeCp`.

5.5.3 Visualisators

Pomocou tried z množiny `reconstructors` a `interpolators` máme k dispozícii metódu, ktorá pre ľubovoľný bod priestoru určí hodnotu `distance function`. V programe boli implementované všetky tri algoritmy popísané v kapitole 4.2. Všetky tri zastrešuje otcovská trieda `CsurfaceVisualisator`, pričom jednotlivé sú implementované v triedach `CMC`, `CMT` a `CRMT`.

Keďže implementácie `marching cubes` a `marching tetrahedra` sú všeobecne známe ďalej bude dopodrobna opísaná iba implementácia `regularised marching tetrahedra`, ktorá vychádza z článku [17], pričom sa však vo viacerých bodoch líši.

Implementácia pracuje s troma rovinami bodov, ktoré definujú vrcholy mriežky zobrazenej na obrázku 4-14. Každý takýto bod obsahuje svoju pozíciu v priestore, hodnotu `distance function`, ďalej takzvaný `EdgeIndex` (14-bitové číslo), ktorý obsahuje na i -tom bite jedničku práve vtedy, ak leží priesečník na hrane z centrálného bodu¹⁰ do vrcholu i a tento priesečník je bližšie k centrálnemu bodu a nakoniec odkazy na priesečníky s izoplochou `SVertexAndNormal * EdgeVertex[7]`, ktoré vzniknú na hranách vedúcich do vrcholov 7 až 15. Nie je potrebné pre každý bod mriežky, aby obsahoval odkazy na všetkých 14 možných priesečníkových vrcholov, pretože viacero z nich je pre susedné body mriežky spoločných.

Keďže na celú realizáciu algoritmu stačia tri roviny, v programe sú definované ako `SPlanePoint **DataPlane[3]`, pričom `SPlanePoint`, je dátová štruktúra obsahujúca

¹⁰ Bod ležiaci na vzorkovacej mriežke v strede štruktúry zobrazenej na obrázku 4-14. Program sa naň odkazuje ako `DataPlane[y % 3][x][z]`.

všetky premenné popísané v predchádzajúcom odseku. Táto štruktúra sa využíva cyklickým spôsobom, teda ak sa pracuje práve s rovinou y (ľubovoľné celé číslo), odkazuje sa na ňu ako `DataPlane[y % 3][x][z]`.

Pri prechádzaní priestorom sa postupne inicializujú dáta mriežky a na základe hodnôt `distance function` sa vypočítavajú pozície priesečníkových bodov s izoplochou. Tieto sú uložené do poľa a odkazy na nich sú umiestnené v dátovej štruktúre príslušného bodu mriežky. Viacero odkazov môže vďaka zhlukovaniu smerovať na ten istý bod. Nakoniec sa prechádzajú jednotlivé štvorsteny, ktoré sú triangulované na základe už vypočítaných priesečníkov.

Vytváranie povrchu zastrešuje metóda `Polygonize`, ktorá volá postupne pre každé y v danom poradí nasledujúce metódy:

```
PolygonizeZeroPass(y)
PolygonizeFirstPass(y + 1)
PolygonizeSecondPass(y + 1)
PolygonizeThirdPass(y + 1)
```

V prvom kroku sú inicializované pozície bodov mriežky, hodnoty `distance function` v týchto bodoch a vynulované odkazy na priesečníkové body. Pretože pre ostatné metódy je potrebné mať inicializované aspoň dve susedné roviny, prvá metóda sa volá dvakrát. Vzorkovacia mriežka nie je pravidelná. Líši sa vždy pre párne a nepárne hodnoty súradníc z a y . Preto sú pozície bodov inicializované takýmto spôsobom:

```
PX = x * fDx * 2 + (z % 2) * fDx - (y % 2) * fDx;
PY = y * fDy;
PZ = z * fDz;
```

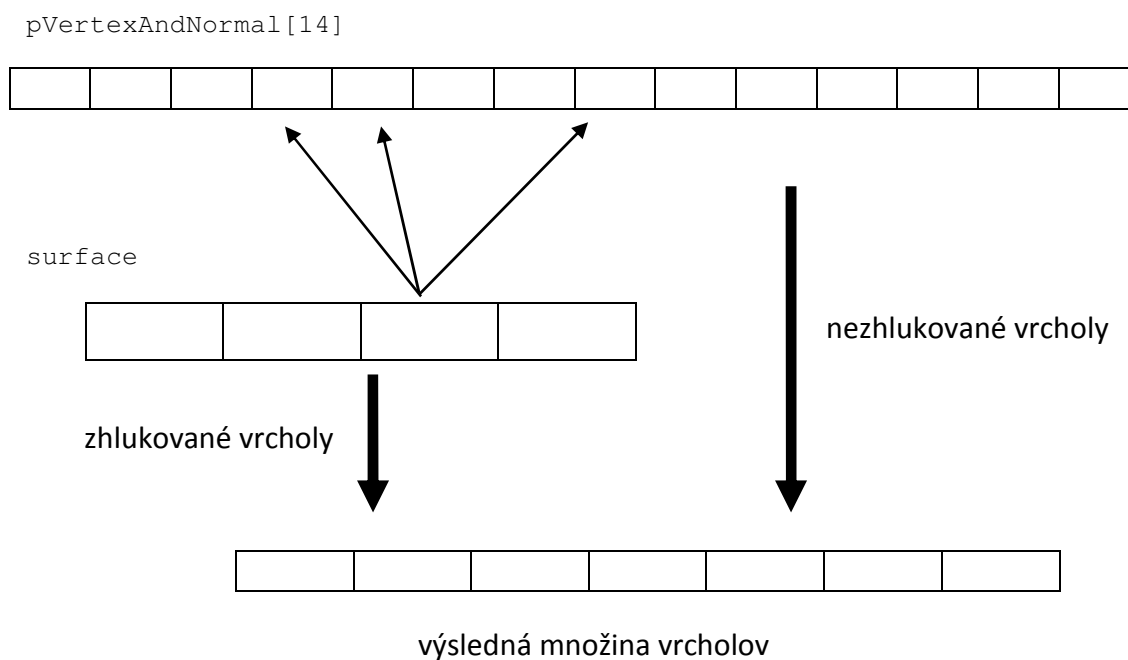
Pričom `fDx`, `fDy` a `fDz` sú premenné definujúce vzdialenosti jednotlivých bodov.

Druhý krok spočíva vo vypočítaní indexov pre označenie blízkych priesečníkov na hranách spájajúcich mriežkové body. Postupne sú porovnávané hodnoty `distance function` centrálného bodu a jeho susedov (tak ako je zobrazený na obrázku 4-14). Ak je zistený priesečník s hranou, uloží sa táto skutočnosť ako bitová hodnota do premennej `iEdgeIndex` bližšieho bodu mriežky. Aby bolo možné odkazovať sa na všetkých štrnást susedov centrálného bodu, bola implementovaná funkcia `GetPoint`. Táto vráti pre súradnice mriežky x , y , z a index susedného bodu i (nadobúdajúceho hodnoty 0 až 13) odkaz na štruktúru príslušného bodu. Kvôli neuniformite vzorkovacej mriežky je funkcia `GetPoint` implementovaná s použitím konštánt, ktoré sú vypísané v kapitole 9.1.

V treťom kroku, sú na základe hodnôt uložených v `iEdgeIndex`, určené možnosti zlučovania vrcholov. Presnú implementáciu rozhodovania možno nájsť v článku [17]. Ďalej sú vytvorené priesečníkové body reprezentované štruktúrou `SvertexAndNormal`. Odkazy na nich sú uložené v štrnásti prvkovom poli `pVertexAndNormal[14]`, ktoré určuje príslušnosť priesečníka k jednotlivým hranám. Na základe hodnôt vypočítaných pre zhlukovanie uložených v poli `surface`, je možné spojiť viaceré body do jedného. V takomto prípade sú totožné odkazy zmazané a nahradené jedným, ukazujúcim na novo vzniknutý vrchol. Celú situáciu popisuje obrázok 5-6. K takto vzniknutým vrcholom je vypočítaná príslušná normála. Nakoniec je vytvorená množina vrcholov uložená do samostatného poľa

(`VertexAndNormalList`), kvôli možnosti jednoducho uvoľniť použitú pamäť pri zničení objektu a odkazy na tieto vrcholy sú priradené do polí `EdgeVertex`, prislúchajúcim bodom mriežky.

Posledným krokom je postupné prechádzanie štvorstenov tvorenými dvoma susednými rovinami vzorkovacej mriežky a na základe priesečníkov s ich hranami určiť správnu trianguláciu. Túto funkciu vykonáva metóda `PolygonizeTetrahedron`, ktorá dostane na vstupe odkazy na šesť vrcholov ležiacich na hranách štvoruholníka. Maximálne štyri z týchto odkazov však nie sú prázdne. Pre tri priesečníky stačí vytvoriť jeden trojuholník, pre štyri sú potrebné dva. Novo vzniknutý trojuholník je uložený do poľa `TrianglesList`. Z dôvodu neuniformnej vzorkovacej mriežky nie je jednoduché získať odkazy na všetky priesečníkové vrcholy patriace jednému štvorstenu. Z tohto dôvodu sú definované dve polia konštánt `EdgeReference` a `TEEdges`, ktoré obsahujú číselné referencie do `EdgeVertex`. Konkrétne hodnoty sú vypísané v kapitole



Obrázok

5-6 Premenné použité pri zhlukovaní vrcholov

5.6 CCellMesh

Trieda `CCellMesh`, tvorí svoju vlastnú skupinu. Jej hlavným účelom je výpočet objemu vstupného telesa, rekonštruovaného telesa a telesa vzniknutého logickou operáciou XOR pôvodných dvoch.. Vypočítané hodnoty uloží do premenných `fVolume0`, `fVolume1` a `fVolume0xor1`.

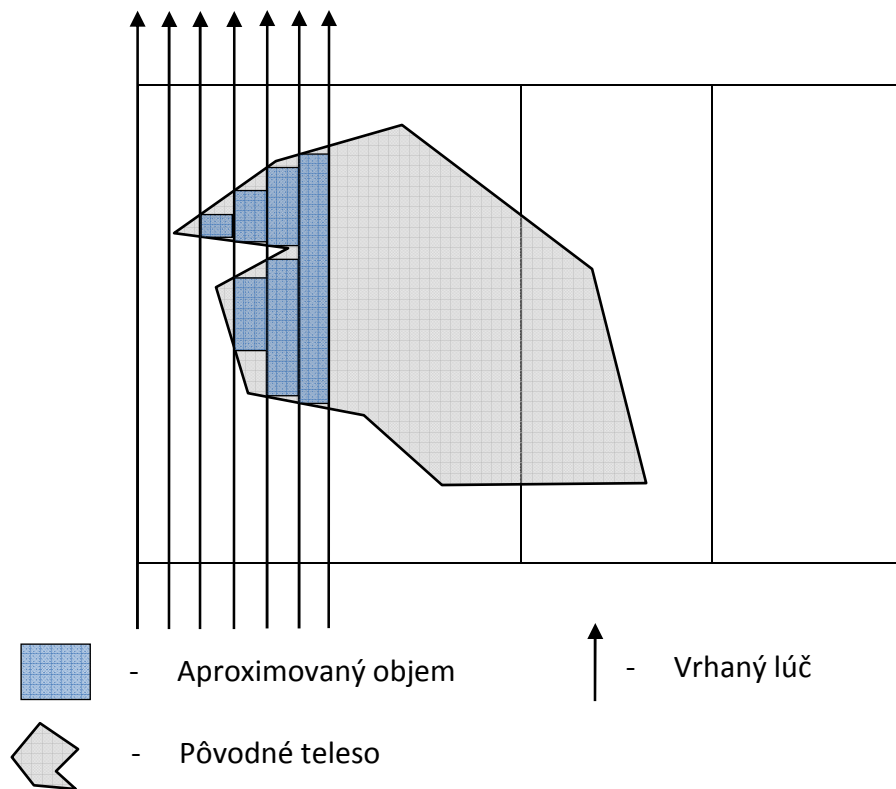
Na získanie týchto hodnôt používa jednoduchý algoritmus založený na vrhaní rovnobežných objemových lúčov cez telesá.

Prvým krokom je „rozsekanie“ modelu do množiny buniek. Bunky sú tvorené identickými, osovo orientovanými kvádrami, ktoré vyplňajú celý skúmaný priestor, ležia vedľa seba iba pre osy x a y . Bunky v smere z zahŕňajú všetky súradnice z . Pri každom vytváraní trojuholníka¹¹ je zavolaná metóda `AddTriangle`, ktorá určí množinu všetkých buniek, ktoré trojuholník pretína. Každá takáto bunka je reprezentovaná štruktúrou `SCell`, ktorá obsahuje pole všetkých takýchto trojuholníkov. Jednotlivé bunky sú uložené v dvojrozmernom poli `PCell ** CellsGrid0` a `PCell ** CellsGrid1`. Prvé pole je určené pre zdrojový model a druhé pre rekonštruovaný. Veľkosť skúmaného priestoru a buniek sa nastavuje pomocou funkcie `Init`.

Výpočet samotného objemu je potom vykonaný zavolaním metódy `ComputeVolumes`, ktorej jediný parameter určuje počet lúčov vrhaných v jednej rovine, rovnobežne s osou z , prechádzajúcich jednou bunkou. Skutočný počet lúčov vrhaných jednou bunkou je teda druhá mocnina daného parametru.

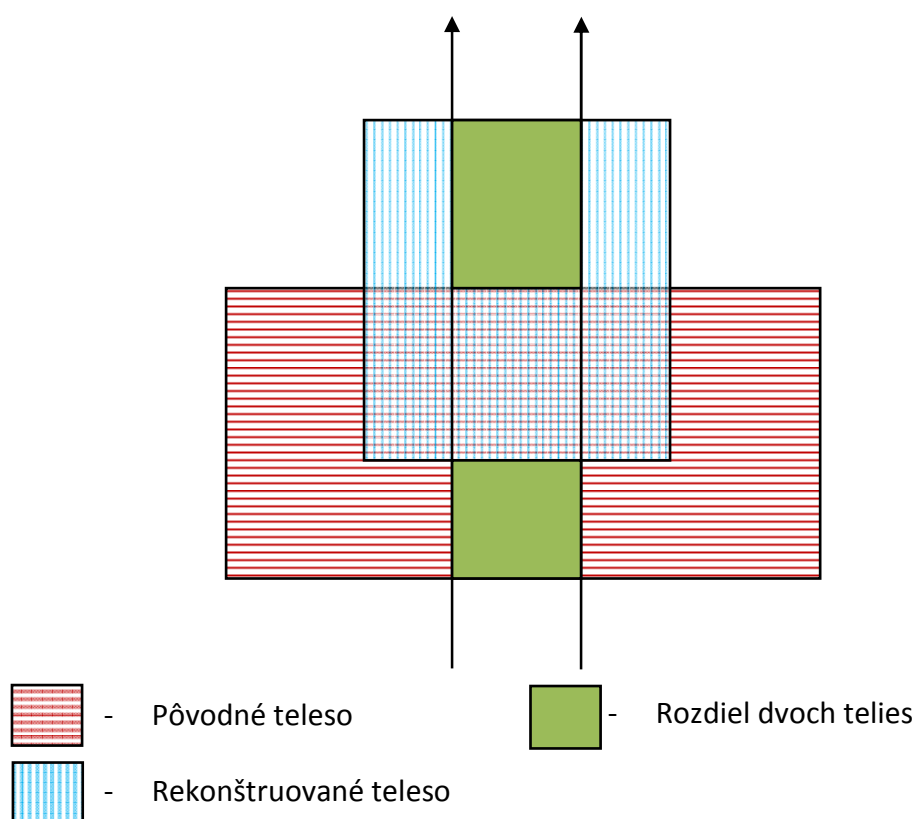
Algoritmus začína pre jeden stĺpec buniek (pozdĺž osi z) vytvorením množiny lúčov, ktoré sú v rámci tohto stĺpca rozmiestnené do dvojrozmernej matice. Postupne sú vypočítavané priesečníky lúčov s trojuholníkmi pre jednotlivé bunky. Tieto priesečníky sú uložené v štruktúre definujúcej lúč – `SRay`. Po prechode jedným stĺpcom sú hodnoty priesečníkov v jednotlivých lúčoch zoradené pomocou algoritmu quicksort. Postupným priechodom cez tieto hodnoty je potom možné určiť približný objem telesa v rámci skúmaného stĺpca (obrázok 5-7) Lúče sú súbežne vytvárané pre zdrojové (`RaysArray0`) aj rekonštruované (`RaysArray1`) teleso. Vďaka tomu je možné súbežným prechádzaním oboch lúčov určiť objem rozdielu medzi dvoma modelmi $((A - B) + (B - A))$, obrázok 5-8).

¹¹ Trojuholníky sú vytvárané pri načítaní vstupného modelu a pri rekonštruovaní povrchu.



Obrázok

5-7 Diskrétné meranie objemu pomocou vrhania lúčov. Pohľad na rez scény cez osu x.



Obrázok
5-8 Výpočet objemu rozdielu dvoch telies.

5.7 Uživatelské rozhranie (Windows, Cviews)

Základom vytvoreného užívateľského rozhrania je 32 bitové Win API. Počas celého behu programu sú zobrazené dve základné a jedno dialógové okno. Každé okno má vlastnú triedu, ktorá sa stará o všetky jeho činnosti od vytvorenia až po zničenie.

5.7.1 Hlavné okno aplikácie

Hlavným oknom celej aplikácie je okno prepojené s triedou `CMainWindow`. Jeho hlavnou úlohou je poskytnúť priestor na zobrazenie vstupných a výstupných modelov a umožniť užívateľovi určiť smer pohľadu. Celá interakcia prebieha v statickej metóde `WindowProc`, v ktorej sú odchyťované najmä správy týkajúce sa myši (`WM_MOUSEMOVE`, `WM_LBUTTONDOWN`...) a správa `WM_PAINT`. Ovládanie myšou je preposielané do triedy `CarcBall`, ktorá umožňuje pohybom myši otáčať scénu tak, ako by bol kurzor prichytený k povrchu gule[18]. Pri spracovaní správy `WM_PAINT` sa volá metóda `Draw` triedy `CViews`. Táto pomocou API OpenGL zobrazí pôvodný model, rekonštruovaný model alebo obidva súčasne, podľa toho ako je

nastavená premenná `ViewType`. Zobrazenie jednotlivých modelov majú na starosti postupne triedy `Cmodel`, `CCrossSections` a `Creconstructor`.

5.7.2 Okno rezov

Druhým oknom aplikácie je prehliadač bitmáp rezov. Jeho hlavnou úlohou je zobrazenie náhľadov rezových rovín vytvorených v triede `CCrossSections`. Tieto bitmapy nie sú nijak upravované a ide teda o priamu vizualizáciu hodnôt používaných pri výpočtoch. Toto zobrazenie má však iba ilustračný charakter, a preto je úplne dostačujúce. O funkčnosť okna sa stará trieda `CCSWindow`. Podobne ako hlavné okno aj táto obsahuje metódu `WindowProc`, ktorá spracováva správy. Na okne sú umiestnené dve tlačidlá. Pri kliknutí na nich sa spracuje správa `WM_COMMAND` a zavolá sa metóda `UpdateData`, ktorá zabezpečí nastavenie textu zobrazeného na okne a skopírovanie hodnôt vzdialeností z rezu uloženom v `CCrossSections` do bitmapy `hBitmap`. Táto bitmapa je potom priradená k device kontextu `hCompatibleDC`, pomocou ktorého je možné prekresliť okno vždy, keď príde správa `WM_PAINT`.

5.7.3 Dialógové okno nastavení rekonštrukcie

Posledným oknom aplikácie je dialógové okno určené na nastavenie parametrov rekonštrukcie. Podobne ako v predchádzajúcich prípadoch, ovládanie všetkých jeho funkcií zabezpečuje vlastná trieda `CRecSettingsWindow`. Toto okno obsahuje viacero tlačidiel, edit boxov, combo boxov a statický text. Na začiatku je väčšina tlačidiel nastavený parameter disabled. Ich postupným stláčaním sa stávajú aktívne. Pri spracovaní správ, definujúcich užívateľskú aktivitu sú nastavované parametre tried vykonávajúcich rekonštrukciu, prípadne sú volané metódy spúšťajúce jej jednotlivé kroky.

6 Výsledky

Vo vytvorenom programe bolo simulovaných viacero rekonštrukcií pri rôznych nastaveniach všetkých implementovaných algoritmov. Porovnávané boli hlavne výsledne modely rekonštrukcie, pričom nebol kladený dôraz na časovú náročnosť. Hlavným kritériom bol v tomto prípade subjektívny pohľad, ktorý umožňoval vďaka dobrému zobrazeniu pôvodného a rekonštruovaného telesa určiť hlavné slabiny a výhody jednotlivých metód. Ako sekundárny aspekt boli použité hodnoty objemov telies a ich rozdielov.

6.1 Porovnanie marching cubes, marching tetrahedra a regularised marching tetrahedra

Pretože ťažiskom rekonštrukcie je grafické znázornenie rekonštruovaných modelov najprv boli porovnané jednotlivé vizualizačné metódy. Marching cubes je z implementačného hľadiska najjednoduchšou metódou. V základnom tvare však nezabezpečuje topologickú korektnosť povrchu. Môžu v ňom teda v určitých prípadoch vzniknúť diery. Marching tetrahedra týmto neduhom netrpí. Pri porovnateľne veľkej vzorkovacej mriežke však generuje oveľa väčšie množstvo trojuholníkov. Tento problém je vyriešený pri použití regularised marching tetrahedra. Táto generuje dokonca menšie množstvo trojuholníkov ako marching cubes a výsledné zobrazenia sú presnejšie. Presné hodnoty sa získavajú však len veľmi ťažko kvôli rôznym vzorkovacím mriežkam, ktoré sú použité v týchto algoritmoch. V tabuľke 6-1 sú zobrazené viaceré namerané hodnoty.

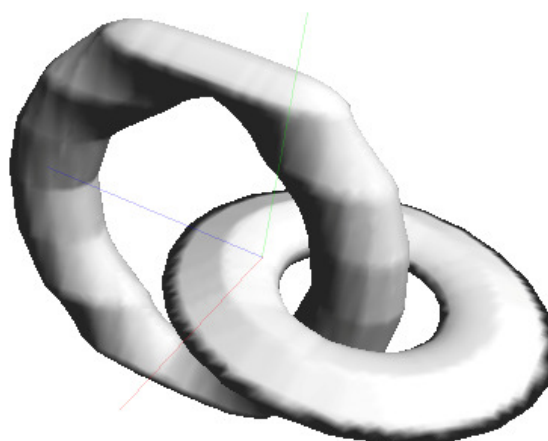
Model	Metóda	Počet trojuholníkov
box.ase	MC	22452
box.ase	MT	84216
box.ase	RMT	8692
2torus.ase	MC	23788
2torus.ase	MT	74196
2torus.ase	RMT	7880
cone.ase	MC	14764
cone.ase	MT	50908
cone.ase	RMT	5192
kon.ase	MC	9396
kon.ase	MT	29356
kon.ase	RMT	2990

Tabuľka

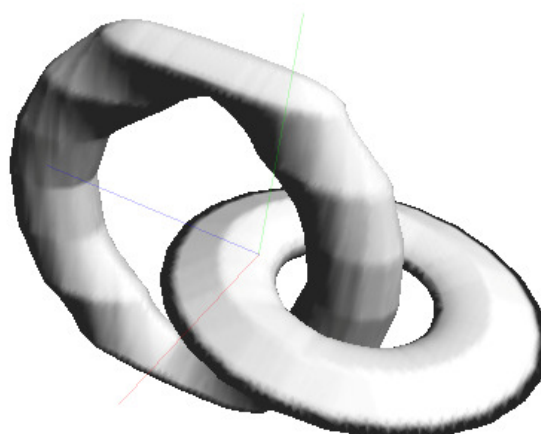
6-1 Počet trojuholníkov vygenerovaných pri použití rôznych metód.

Počet trojuholníkov generovaných regularised marching tetrahedra je rádovo nižší ako v prípade marching tetrahedra.

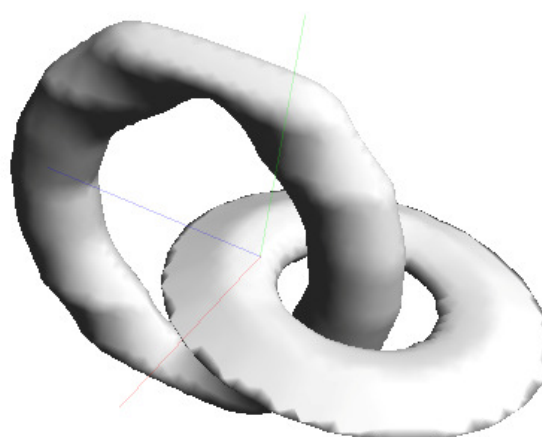
marching cubes (MC)



marching tetrahedra (MT)



regularised marching
tetrahedra (RMT)



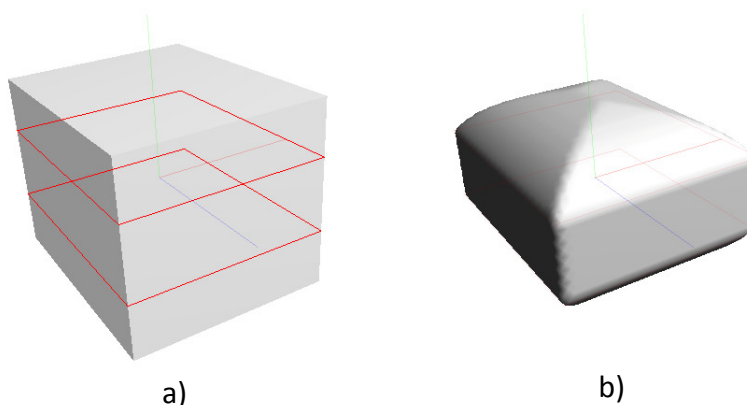
Obrázok
6-1 Rekonštruované teleso zobrazené rôznymi vizualizačnými metódami

Na obrázku 6-1 je zobrazené rekonštruované teleso zobrazené všetkými trom vizualizačnými metódami. Skokové prechody, ktoré akoby rozdeľujú teleso na časti medzi rezmi nie sú spôsobené samostatnou vizualizáciou, ale interpoláciou vstupných hodnôt. Na prvý pohľad je vidieť rozdiel v hladkosti povrchu pri použití RMT na rozdiel od ostatných metód. To je spôsobené z jednej časti odlišným spôsobom výpočtu normály a z druhej menším počtom trojuholníkov.

6.2 Rekonštrukcia z paralelných rezov

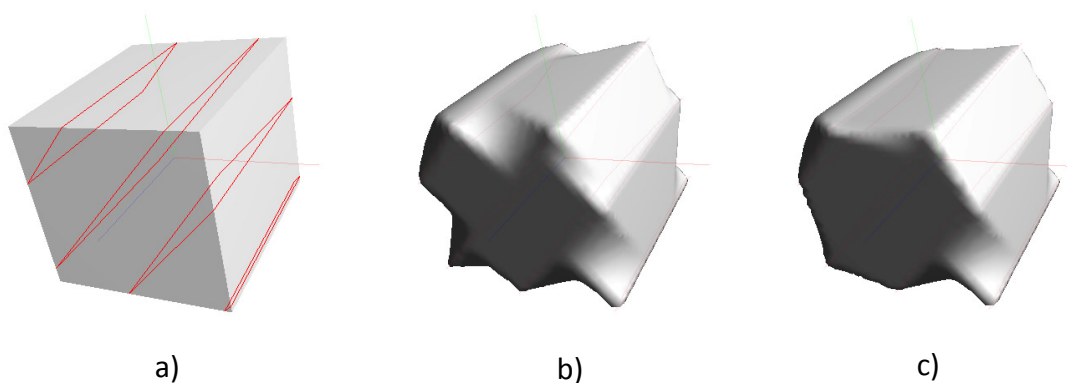
Keďže interpolačné metódy nerozlišujú zásadným spôsobom rekonštrukciu z paralelných a neparalelných rezov najprv boli preskúmané rekonštrukčné schopnosti pri použití rezov paralelných. Za zdrojové teleso boli postupne vyberané: kocka, guľa, cylinder, zložené torusy a model koňa. Vizualizácia bola zaobstaraná pomocou regularised marching tetrahedra, ktorá dáva v porovnaní s marching cubes a marching tetrahedra lepšie výsledky. Pre paralelné rezy nebolo potrebné skúmať zvlášť rozdiely medzi interpoláciou cez najbližšie body a priemernú normálu, pretože obidve metódy sa v takomto prípade zhodujú.

Model kocky rezaný paralelnými rezmi rovnobežnými s podstavou dával úplne rovnaké výsledky pre všetky použité metódy. Je to spôsobené tým, že schopnosť disc guided interpolácie prepájať časti modelov nie je v takomto prípade využitá. Model je rekonštruovaný výborne medzi rezmi. Problém nastáva iba nad a pod časťou priestoru, ktorú tieto dva rezy vymedzujú. V tejto oblasti však o zdrojovom modeli neexistujú žiadne informácie. Jednou možnosťou by bolo v takomto mieste nastaviť hodnotu distance function na nulu za účelom nedefinovanú oblasť vôbec nezobrazovať. V takom prípade by však pri vizualizácii vznikli artefakty spôsobené nespojitosťou distance function, a preto je z praktického hľadiska výhodnejšie použiť spojitý prechod (obrázok 6-2).



Obrázok 6-2 Rekonštrukcia kocky pomocou paralelných rezov.
a) zdrojový model s dvoma rezmi b) rekonštruovaný model

Zaujímavejší prípad nastáva pri použití rezov nerovnobežných s podstavou kocky. Markantný je najmä rozdiel medzi druhým až štvrtým rezom, kde v prípade disc guided interpolácie nevznikajú preliačiny (obrázok 6-3).

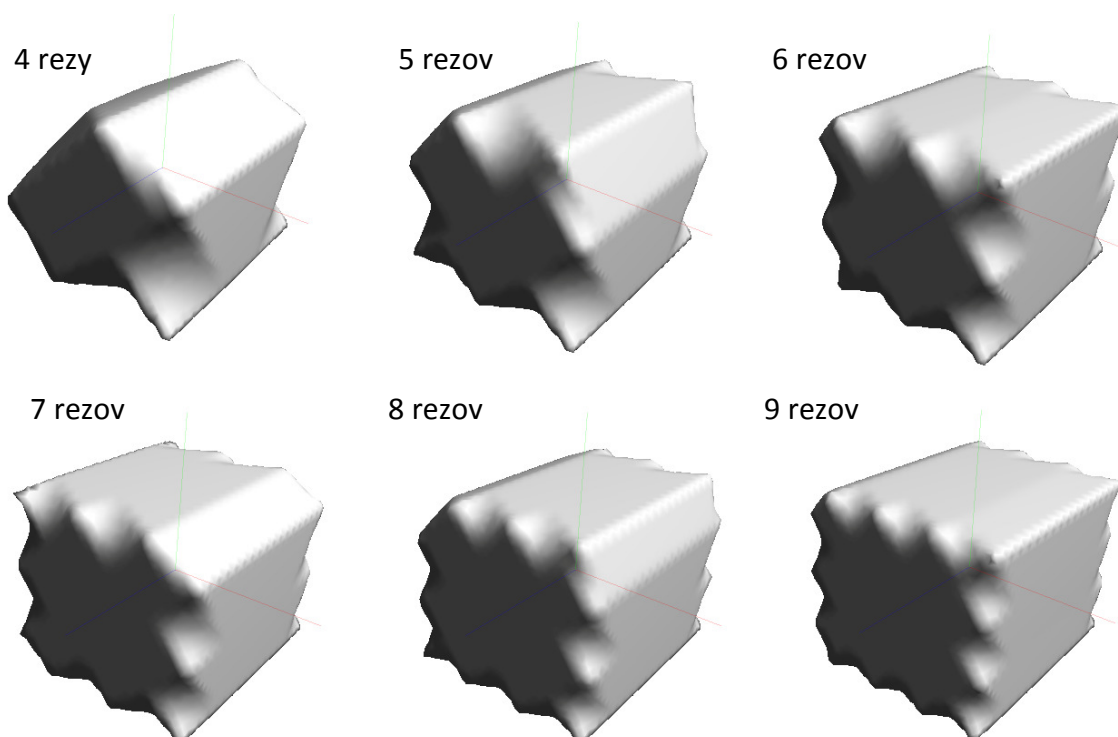


Obrázok

6-3 Rekonštrukcia kocky pomocou paralelných rezov nerovnoběžných z podstavou kocky.

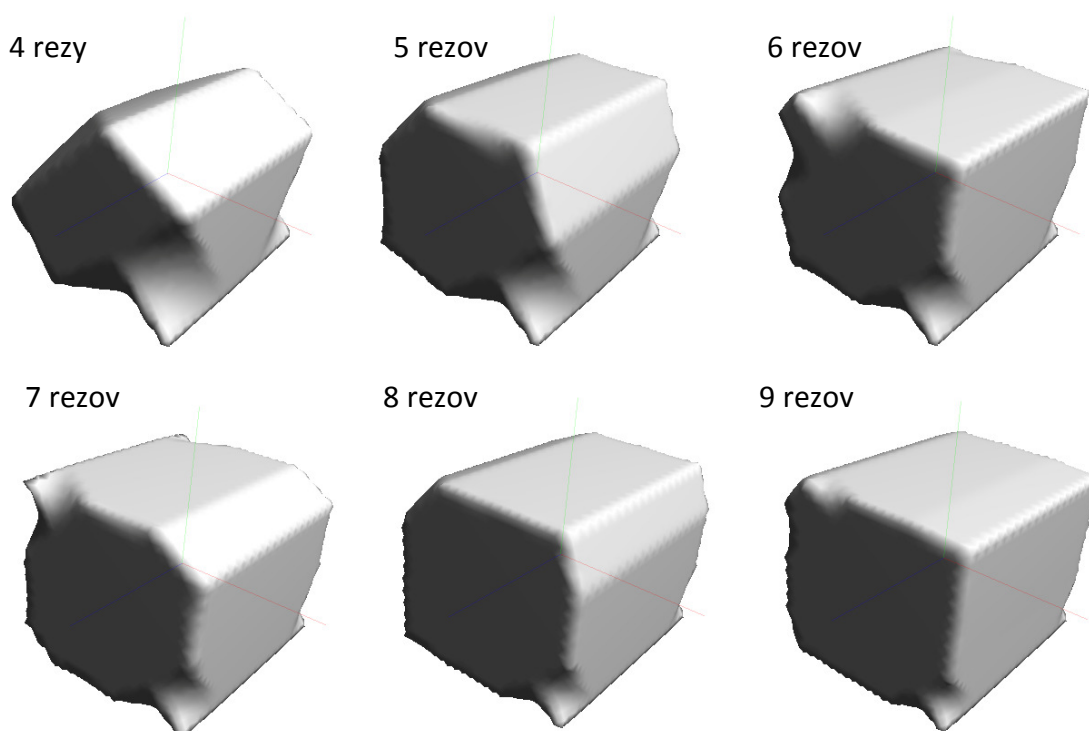
a) zdrojový model b) rekonštrukcia s použitím interpolácie najbližších bodov c) rekonštrukcia s použitím disc guided interpolácie

Podrobnejšie boli porovnané metódy disc guided s closest point pri postupnom raste počtu rezov. V programe bol nastavený reconstructor na parallel, pričom prvý parameter bol 1 čo znamenalo rezy naklonené o 45 stupňov, druhý parameter znamenal počet rezov. Visualisator bol nastavený na regularised marching tetrahedra a interpolator najprv na closest point, potom na disc guided. Postupne bola rekonštruovaná kocka s použitím štyroch až deviatich rezov. Na obrázkoch 6-4 a 6-5 je vidieť hlavné rozdiely medzi obidvoma metódami. Už na prvý pohľad je jasne vidieť schopnosť disc guided interpolácie lepšie sledovať jednotlivé rezy a vhodne určiť smer interpolácie.



Obrázok

6-4 Rekonštrukcia kocky pomocou closest point interpolácie pre rôzny počet rezov.

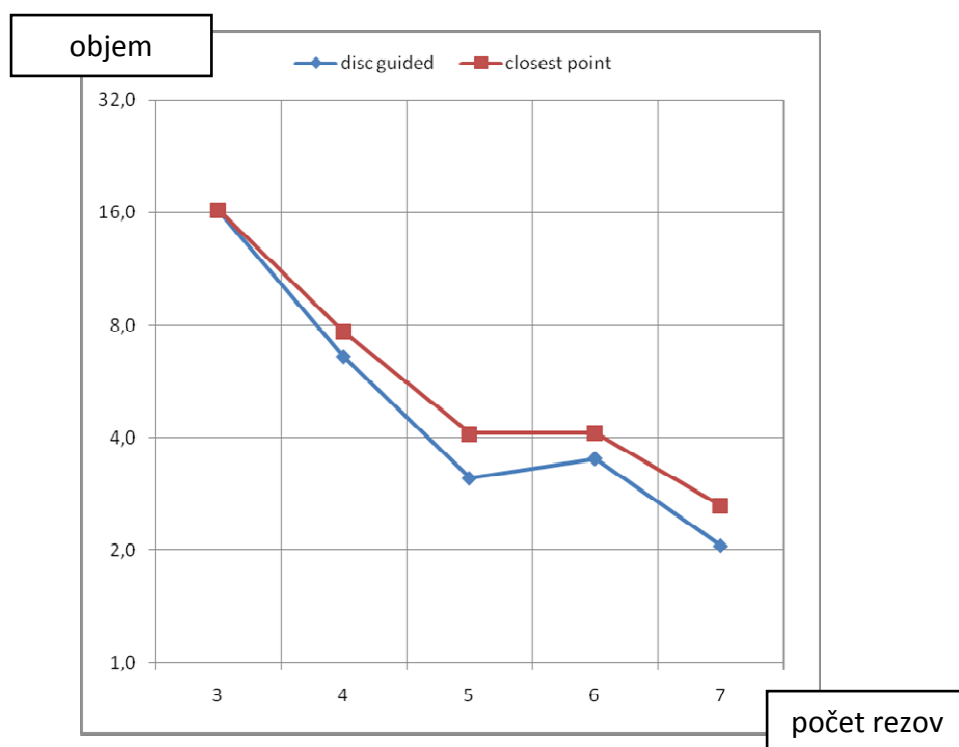


Obrázok
6-5 Rekonštrukcia kocky pomocou disc guided interpolácie pre rôzny počet rezov.

Na grafe 6-1 sú znázornené hodnoty objemu, ktorý odpovedá rozdielovému telesu medzi zdrojovým a rekonštruovaným povrchom. Čím menšia je táto hodnota, tým viac sa rekonštruované teleso podobá pôvodnému. Z grafu jasne vyplýva lepšie postavenie disc guided interpolácie. S pribúdajúcimi rezmi by mala teoreticky hodnota rozdielového objemu vždy klesať, pretože ďalší rez spresňuje rekonštrukčný proces. V tomto prípade sú však rezy rovnomerne rozdelené pričom začínajú v pravom prednom dolnom rohu kocky, a teda môže nastať určité zhoršenie. Toto sa ukázalo aj v prípade použitia siedmich rezov, kde nastalo zhoršenie pri použití obidvoch metód.

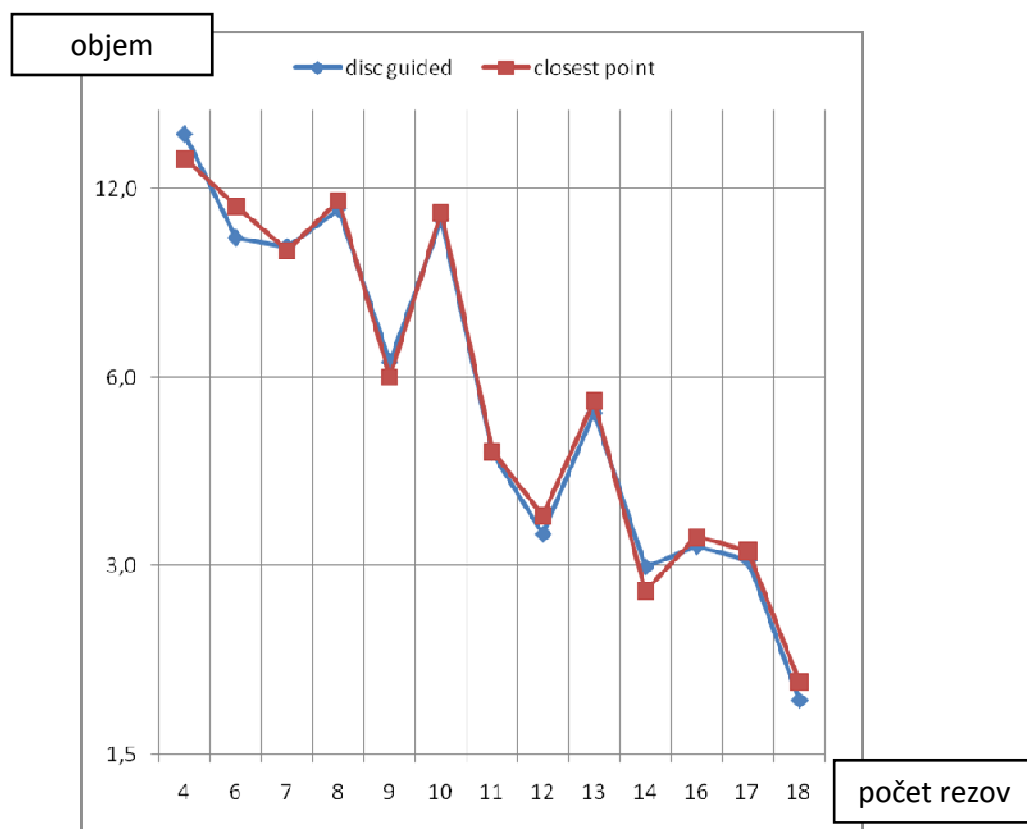
Graf 6-2 znázorňuje rovnaké hodnoty, tentokrát však pri rekonštruovaní spojených torusov. V tomto prípade je objekt rekonštruovaný takmer totožne pri použití obidvoch metód, aj keď disc guided interpolácia má vo väčšine prípadov malý náskok.

Veľmi dobré výsledky možno pozorovať pri rekonštrukcii cylindra. Ten je pri použití paralelných rezov usporiadaných tak ako vidno na obrázku 6-6 rekonštruovaný s pomocou disc guided interpolácie veľmi presne. Naproti tomu closest point interpolácia si z veľmi vzdialenými rezmi nedokáže poradiť, čo je dobre vidieť aj na grafe hodnôt rozdielových objemov 6-3.



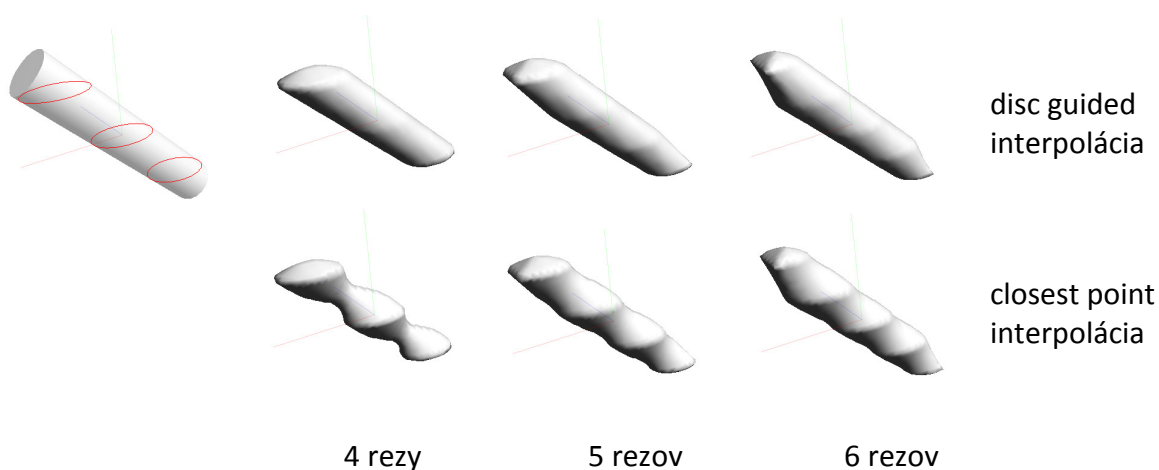
Graf

6-1 Objemy rozdielov medzi zdrojovým a rekonštruovaným telesom v závislosti od počtu rezov pri rekonštrukcii kocky s použitím paralelných rezov.



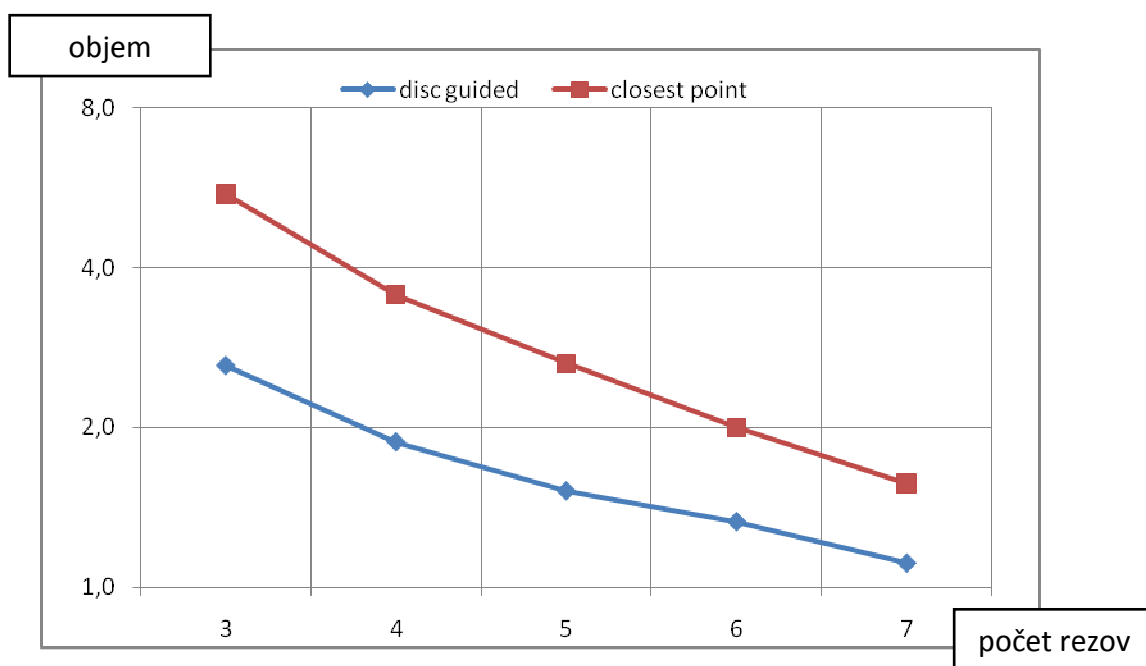
Graf

6-2 Objemy rozdielov medzi zdrojovým a rekonštruovaným telesom v závislosti od počtu rezov pri rekonštrukcii spojených torusov s použitím paralelných rezov.



Obrázok

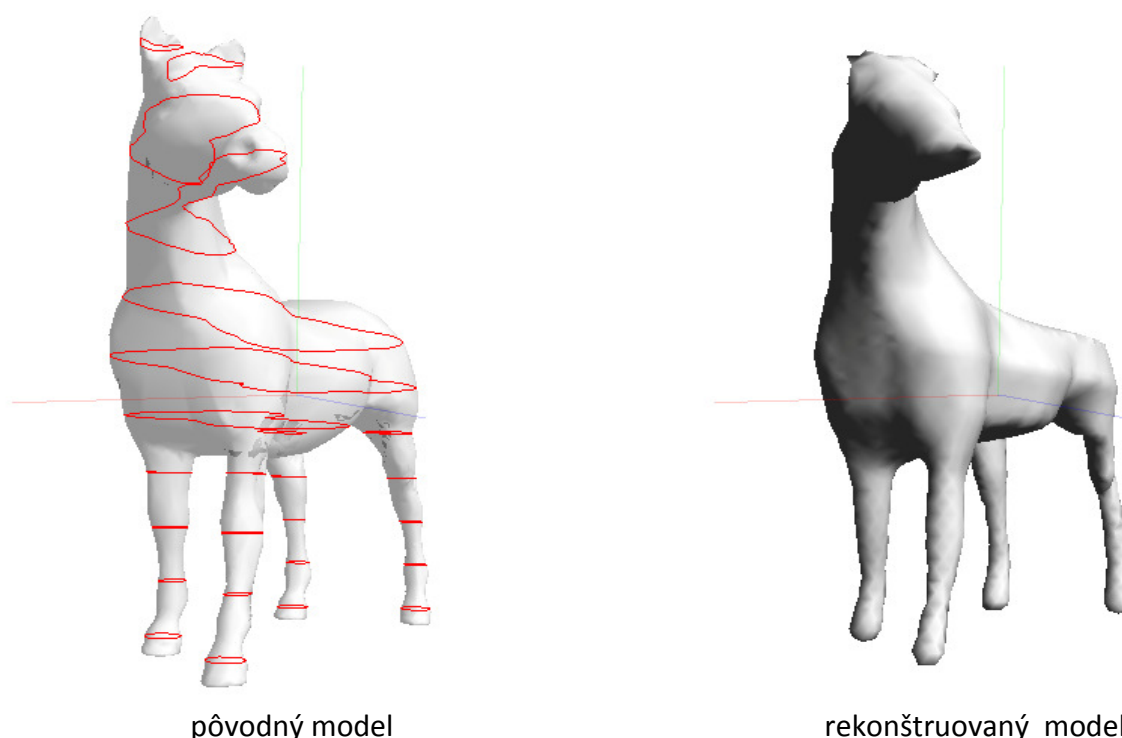
6-6 Rekonštrukcia cylindra pri použití interpolácie closest point a disc guided z rôzneho počtu rezov.



Graf

6-3 Objemy rozdielov medzi zdrojovým a rekonštruovaným telesom v závislosti od počtu rezov pri rekonštrukcii cylindra s použitím paralelných rezov.

Posledným rekonštruovaným modelom bol model koňa, ktorý dával veľmi podobné výsledky ako zložené torusy. Pôvodný model a rekonštrukciu využívajúcu disc guided interpoláciu je možno vidieť na obrázku 6-7.



Obrázok
6-7 Rekonštrukcia koňa s použitím paralelných rezov a disc guided interpolácie.

6.3 Rekonštrukcia z neparalelných rezov

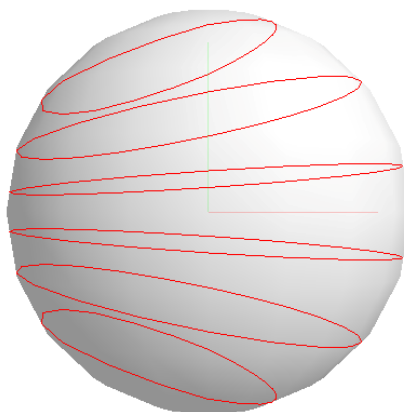
Podobne ako pre paralelné rezy, budú v tejto sekcii porovnané dve metódy. Kvôli neparalelnosti rezov však metódu closest point nahradí metóda average normal (priemerná normála). Closest point totižto pri použití v neparalelných rezoch častokrát vytvára veľké množstvo artefaktov (viď obrázok 3-7).

Najskôr boli rekonštruované telesá kocka, zložené torusy a model koňa z rezov rovnomerne rozložených po rekonštruovanom priestore, pričom ich normála mohla nadobúdať pre jednotlivé súradnice tieto hodnoty:

$$x \in (-0.2, 0.2); y = 1; z \in (-0.2, 0.2)$$

Takéto rezy sú sekvenčne zoradené a je možné rýchlejšie vypočítať, z ktorých dvoch je potrebné interpolovať. V praxi ich možno získať ak sa použije ručný ultrazvuk s polohovacím senzorom a operátor bude kopírovať povrch skúmaného telesa v jednom smere. Takto vzniknuté rezy sa môžu pretínať.

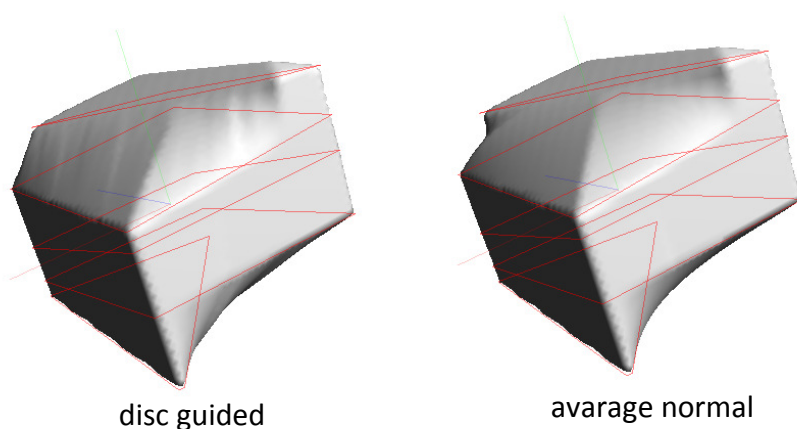
Druhým typom rezov bol takzvaný vejár. V takomto prípade sa všetky rezové roviny pretínajú v jednej priamke a rovnomerne rozdeľujú priestor obsahujúci zdrojové teleso (obrázok 6-8). Podobne ako v predchádzajúcom prípade, aj tieto rezy je možné veľmi ľahko vytvoriť v praxi.



Obrázok
6-8 Model gule rozrezaný pomocou vejárových rezov

6.3.1 Rekonštrukcia kocky

Prvým rekonštruovaným telesom bola kocka. Prvým spôsobom bola rekonštruovaná z ôsmich rezov, pričom samotné teleso pretlo vždy len päť rezových rovín. Výsledky objemov, rozdielových objemov a percentuálnej chyby sú vypísané v tabuľke 6-2. Objem pôvodného modelu kocky je 60.5. Chyba bola v tomto prípade vypočítaná vzhľadom na tento objem ako časť, ktorú zaberá rozdielový objem z celého objemu pôvodného telesa. Z tabuľky jasne vyplýva mierna výhoda disc guided interpolácie. Percentuálna chyba je však u oboch metód značne vysoká, čo vyplýva hlavne z malého počtu rezov. Na obrázku 6-9 je zobrazený výsledok rekonštrukcie, kde je vidno len malé rozdiely medzi jednotlivými metódami.



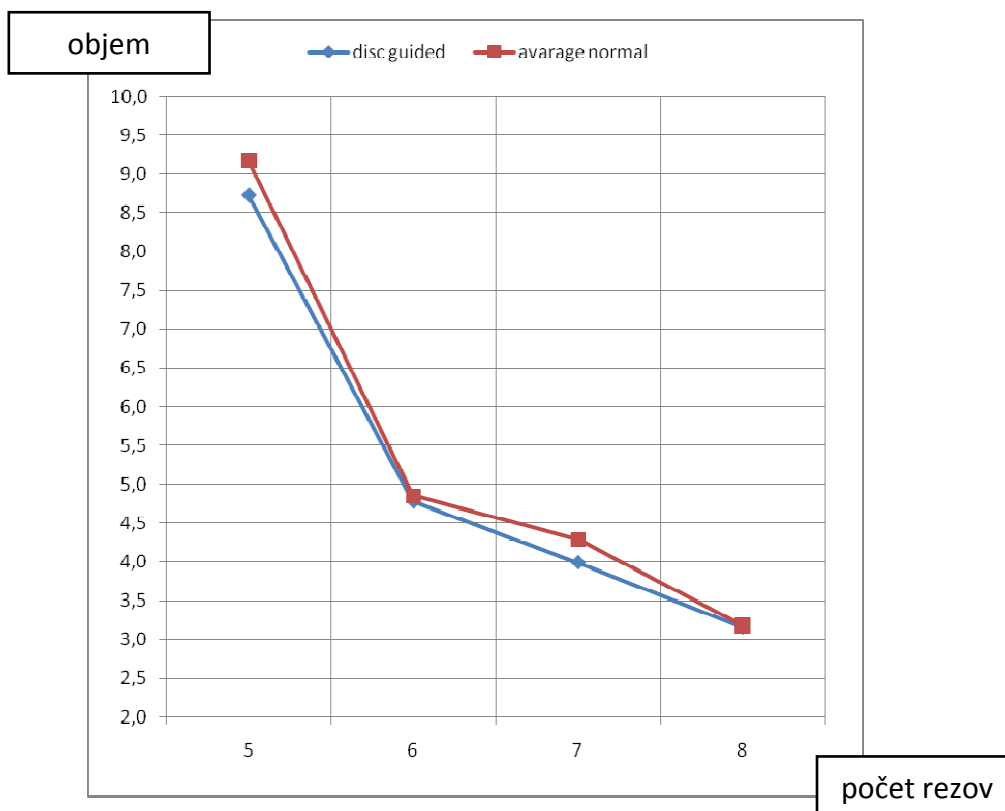
Obrázok
6-9 Porovnanie výsledku rekonštrukcie kocky z neparalelných rezov pomocou dvoch metód.

p.c.	DG xor objem	DG rec objem	chyba	AN xor objem	AN rec objem	chyba
1	10,26	52,4	17,0%	10,17	52,49	16,8%
2	7,82	54,9	12,9%	7,93	54,82	13,1%
3	11,61	52,65	19,2%	11,99	52,27	19,8%
4	10,29	53,73	17,0%	10,31	53,7	17,0%
5	8,27	55,81	13,7%	8,65	55,42	14,3%
6	11,11	54,07	18,4%	11,75	53,43	19,4%
priemer	9,89	53,93	16,4%	10,13	53,69	16,7%

Tabuľka

6-2 Hodnoty objemov pre rekonštruovanú kocku neparalelnými rezmi.

V druhom prípade sa rekonštruovalo pomocou vejárovitých rezov. Pri ich tvorbe neboli použité žiadne náhodné veličiny, tým pádom bolo možné pozorovať zmeny rozdielov pri použití rôzneho počtu rezov. Hodnoty rozdielových objemov sú zakreslené v grafe 6-4. Z tohto grafu je dobre vidieť, že pri použití ôsmich rezov prechádzajúcich cez teleso klesá rozdielový objem na hodnotu okolo 3.2, čo už predstavuje menej ako štvorpercentnú chybu.



Graf

6-4 Objemy rozdielov medzi zdrojovým a rekonštruovaným telesom v závislosti od počtu rezov pri rekonštrukcii kocky s použitím neparalelných vejárových rezov.

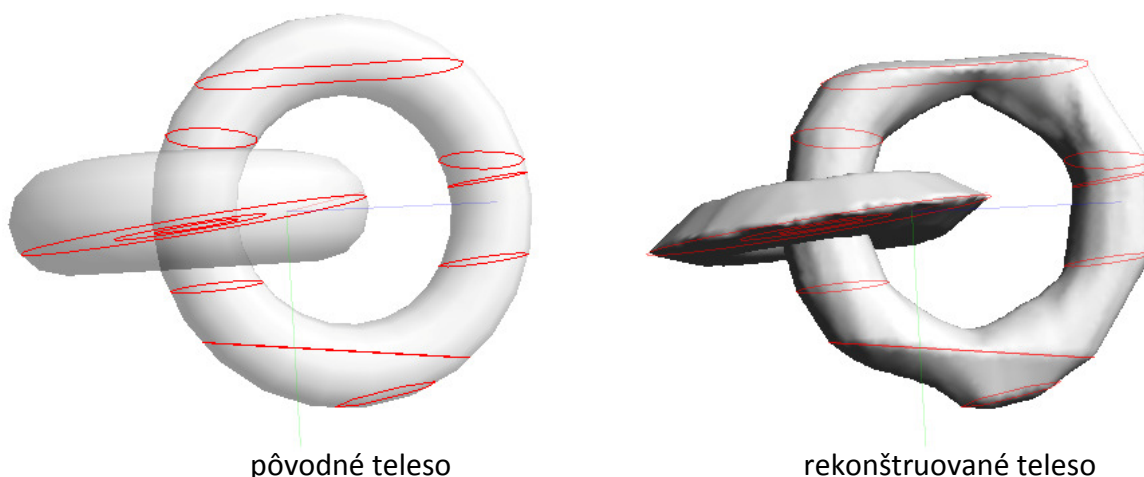
6.3.2 Rekonštrukcia spojených torusov

Druhým rekonštruovaným telesom boli dva spojené torusy, ktorých celkový objem je 23,75. Podobne ako v predchádzajúcom prípade sú hodnoty objemov pri rekonštrukcii z náhodných sekvenčných rezov zobrazené v tabuľke 6-3. V tejto menej syntetickej situácii ako v prípade rekonštrukcie kocky vidno výraznejší rozdiel medzi obidvoma metódami. Percentuálna chyba sa zdá byť na prvý pohľad obrovská. Je však spôsobená tým, že cez torus s väčším objemom prechádza najčastejšie len jedna rezová rovina zo šiestich použitých. Tým pádom je jeho objem značne zredukovaný. Pri zdvojnásobení rezov, prechádzajú cez tento torus v priemere tri rezové roviny a chyba klesne pod 15%.

p.c.	DG xor objem	DG rec objem	chyba	AN xor objem	AN rec objem	chyba
1	8,03	16,76	33,8%	8,4	16,28	35,4%
2	11,38	14,02	47,9%	11,47	12,81	48,3%
3	10,9	14,12	45,9%	11,05	13,51	46,5%
4	10,86	13,82	45,7%	11,74	12,46	49,4%
5	11,2	13,63	47,2%	11,63	12,65	49,0%
6	8,93	15,78	37,6%	9,55	14,84	40,2%
priemer	10,22	14,69	43,0%	10,64	13,76	44,8%

Tabuľka

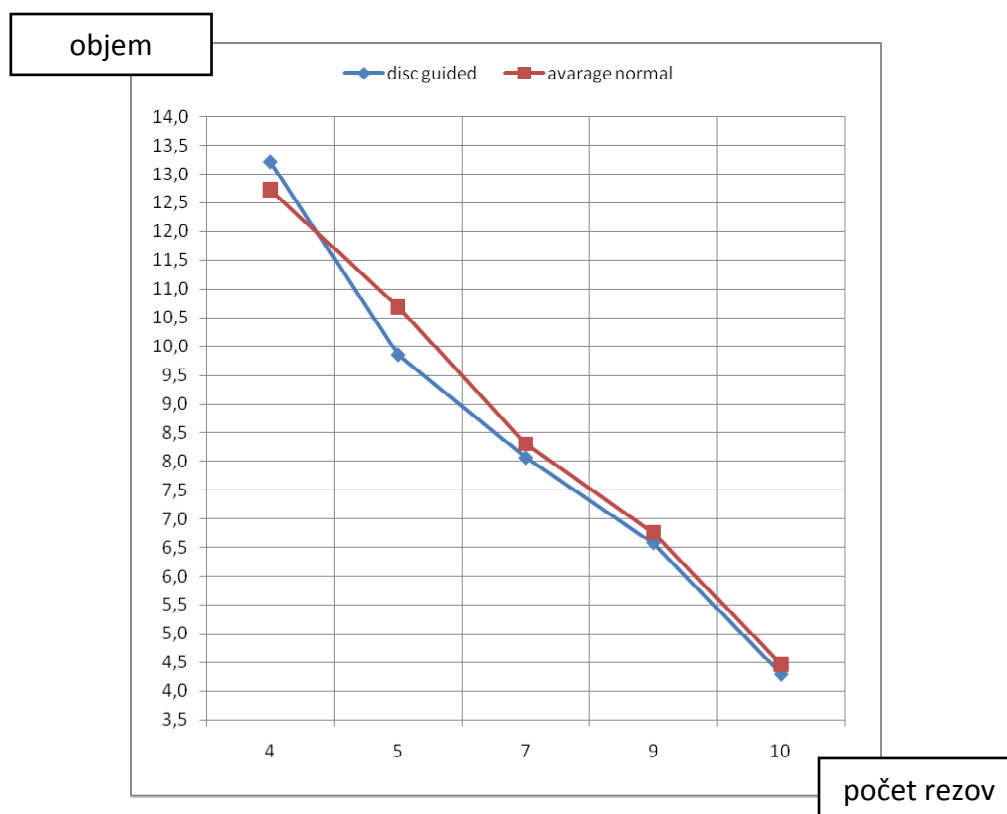
6-3 Hodnoty objemov pre rekonštrukciu zložených torusov neparalelnými rezmi.



Obrázok

6-10 Problém veľkej chyby pri malom počte rezov prechádzajúcich objemnejším torusom.

Podobne ako v prípade kocky boli merané hodnoty rozdielových objemov pri rekonštrukcii z rôzneho počtu vejárových rezov. Tieto sú vyznačené na grafe 6-5.



Graf

6-5 Objemy rozdielov medzi zdrojovým a rekonštruovaným telesom v závislosti od počtu rezov pri rekonštrukcii zložených torusov s použitím neparalelných vejárových rezov.

6.3.3 Rekonštrukcia modelu koňa

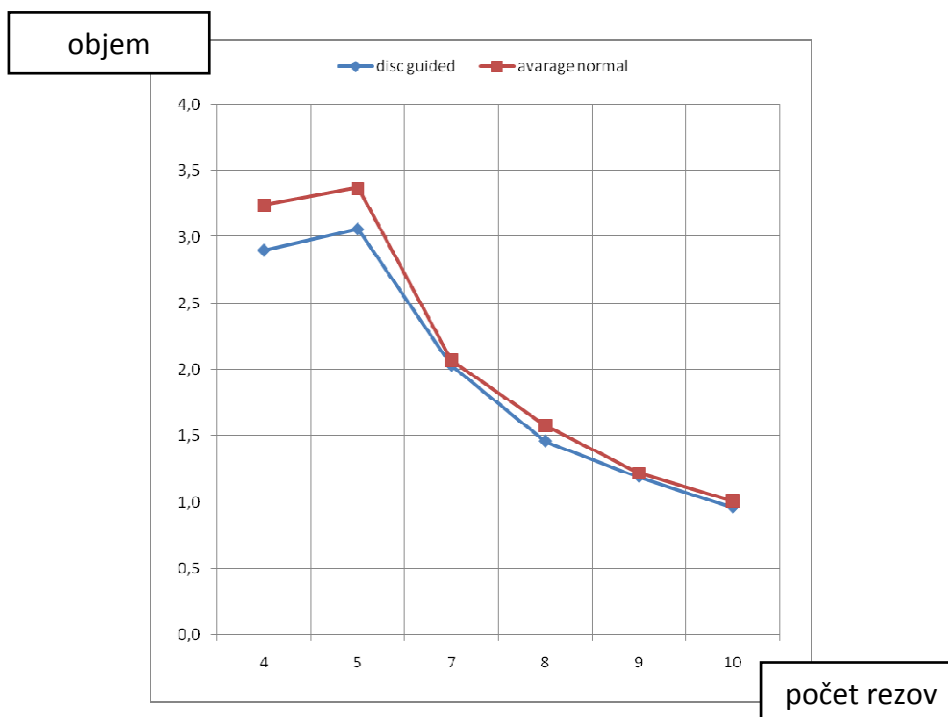
Rekonštrukcia modelu koňa zo sekvenčných neparalelných rezov dopadla veľmi podobne ako v prípade zložených torusov (tabuľka 6-4).

p.c.	DG xor objem	DG rec objem	chyba	AN xor objem	AN rec objem	chyba
1	1,16	6,11	17,2%	1,19	5,95	17,7%
2	1,19	5,96	17,7%	1,18	5,88	17,5%
3	1,41	5,89	21,0%	1,41	5,8	21,0%
4	1,21	6,16	18,0%	1,23	6,04	18,3%
5	1,13	5,94	16,8%	1,19	5,81	17,7%
6	0,92	6,05	13,7%	1,01	5,93	15,0%
priemer	1,17	6,02	17,4%	1,20	5,90	17,9%

Tabuľka

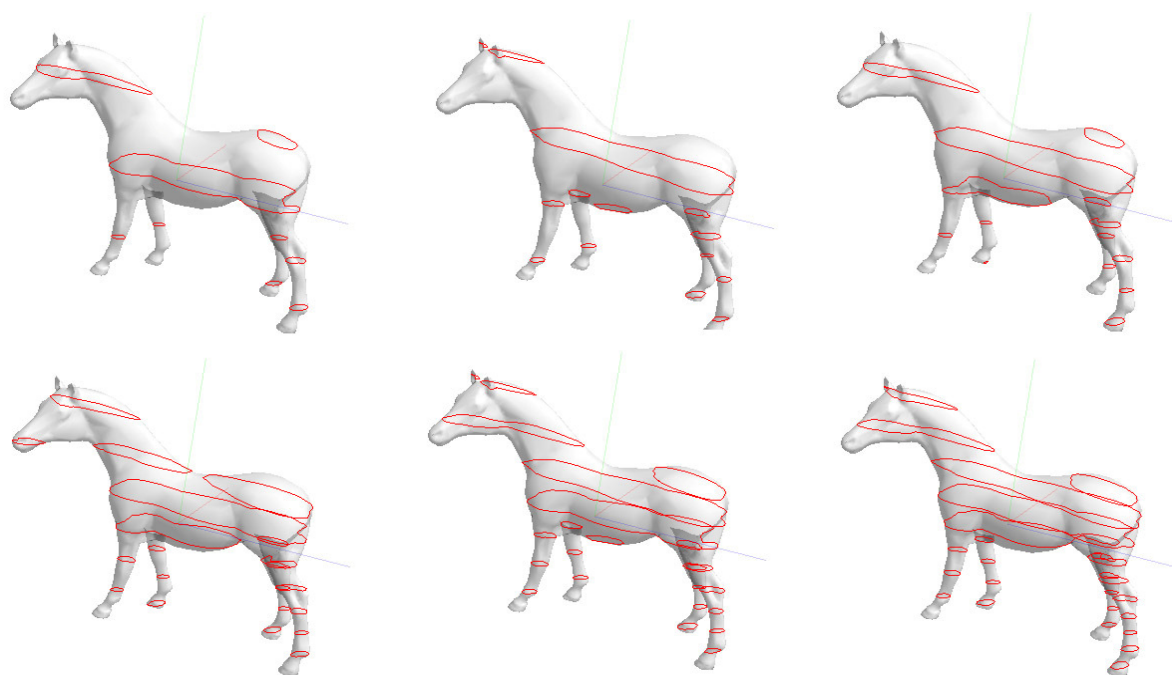
6-4 Hodnoty objemov pre rekonštruovaného koňa neparalelnými rezmi.

V prípade vejárových rezov je zaujímavá najmä rekonštrukcia z minimálneho počtu rezov. Na grafe 6-6 je vidno veľké rozdiely v rozdielových objemoch pre malý počet rezov. V takomto prípade je jasne lepšia metóda disc guided, ktorá je schopná napájať na seba jednotlivé časti rezov. Tento rozdiel s počtom rezov klesá. Na obrázkoch 6-11 až 6-13, sú zobrazené výstupy jednotlivých rekonštrukcií.

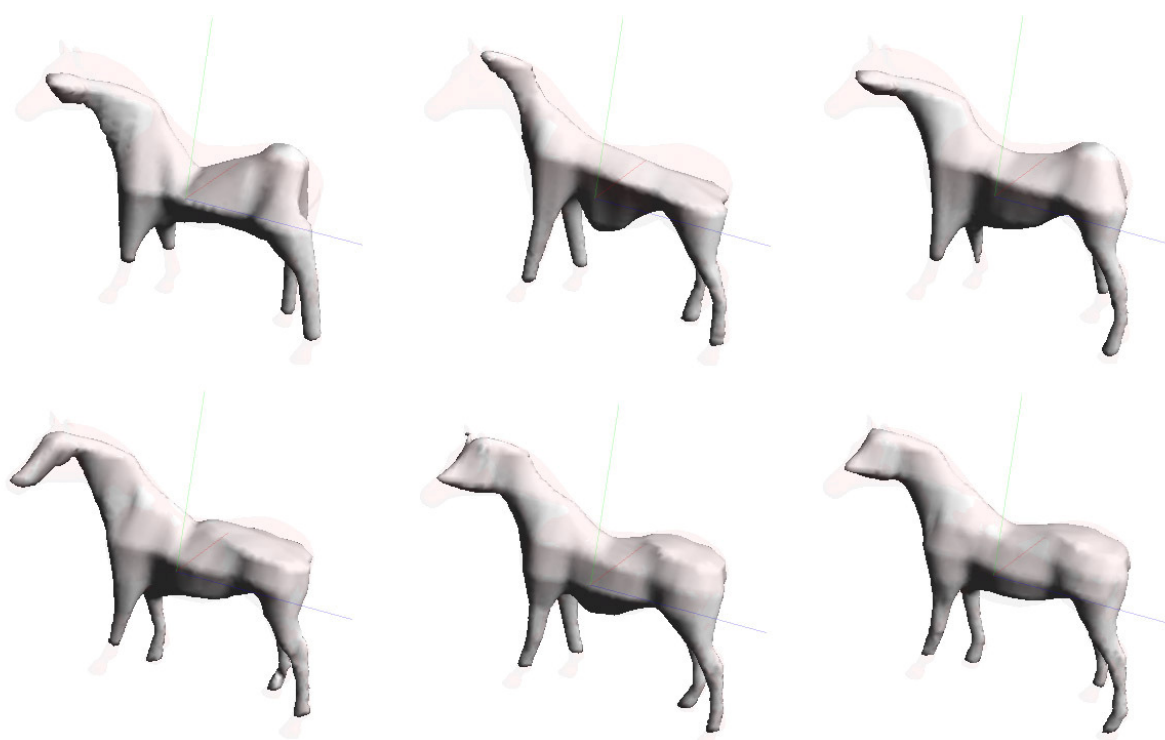


Graf

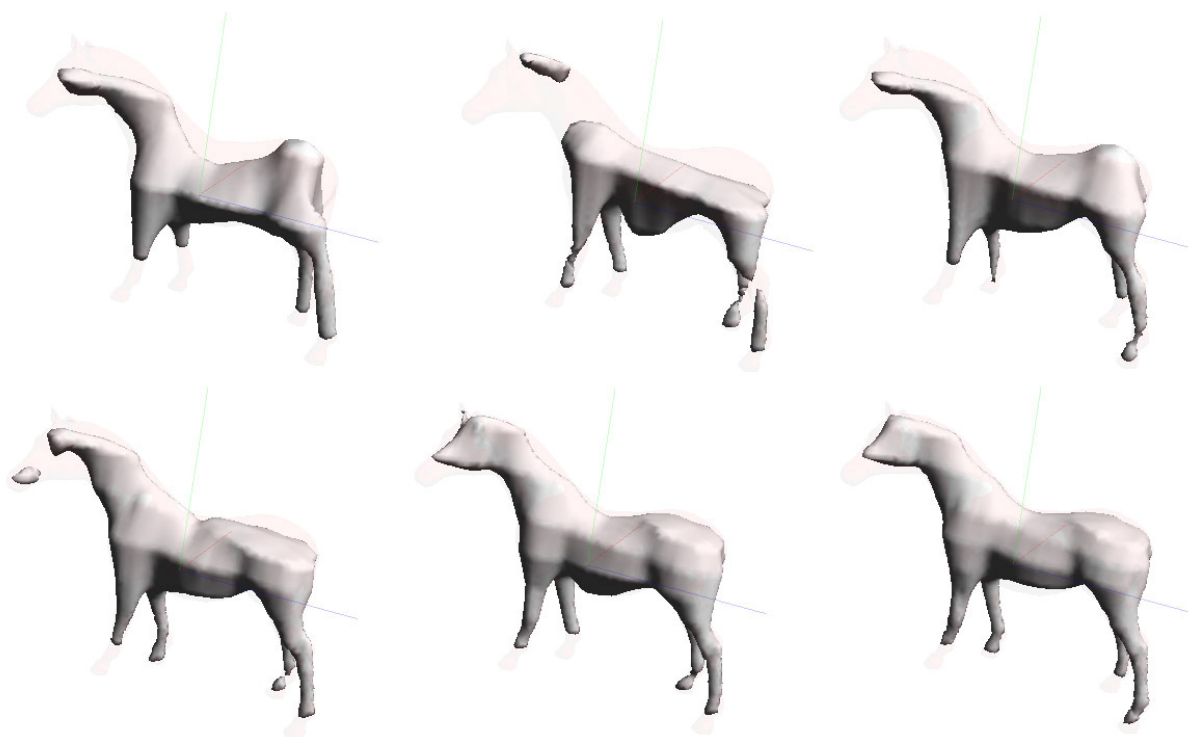
6-6 Objemy rozdielov medzi zdrojovým a rekonštruovaným telesom v závislosti od počtu rezov pri rekonštrukcii modelu koňa s použitím neparalelných vejárových rezov.



Obrázok
6-11 Vejárové rezy modelu koňa.



Obrázok
6-12 Rekonštrukcia koňa z vejárových rezov s použitím disc guided interpolácie.



Obrázok
6-13 Rekonštrukcia koňa z vejárových rezov s použitím average normal interpolácie.

7 Záver

Táto práca poskytla širší prehľad v metódach rekonštruujúcich povrch pomocou interpolačných algoritmov. Zameraná bola na algoritmy interpolujúce hodnoty distance function z dvoch rezov. Takisto sa okrajovo dotkla priamych metód vytvárajúcich trojuholníkovú sieť bez použitia skalárneho poľa. Z vizualizačných metód boli spomenuté základné princípy fungovania priamych metód, ktoré sa vďaka výkonu súčasného hardware dostávajú do popredia a podrobne boli popísané nepriame metódy.

Implementované boli algoritmy vytvárajúce rezy zo zdrojového 3D modelu, interpolačné algoritmy interpolujúce pomocou najbližšieho bodu (closest point), priemernej normály (average normal) a metódy disc guided. Súčasťou priloženého programu sú aj vizualizačné metódy marching cubes, marching tetrahedra a regularised marching tetrahedra a algoritmus merania objemu rozdielov. Aplikácia teda poskytuje ucelený systém, v ktorom je možné porovnávať rozdiely jednotlivých metód, hlavne vďaka grafickému zobrazeniu pôvodného aj rekonštruovaného telesa v jednej scéne.

Najkomplikovanejšími algoritmami pre implementáciu sa ukázali byť disc guided interpolácia a regularised marching tetrahedra napriek tomu, že sú publikované ich detailné popisy. Implementácia druhého z nich vyžaduje použitie viacerých konštánt popísaných v kapitole 8. Nezanedbateľne náročné bolo aj získanie rezov z 3D modelu uloženého vo formáte ASE.

Aplikácia je navrhnutá tak, aby ju bolo možné kedykoľvek rozšíriť o ďalšie metódy, pričom môžu byť použité rovnaké vstupné dáta. Nie je teda problém naprogramovať novú vizualizačnú alebo interpolačnú triedu a porovnať jej výsledky s pôvodnými. Pri zmene triedy zo skupiny reconstructors, môže byť vytvorený dokonca úplne nový systém rekonštrukcie.

Pretože nebol kladený veľký dôraz na rýchlosť so základným nastavením, môže trvať rekonštrukcia niektorých telies z desiatich neparalelných rezov pri použití najlepších metód až desať sekúnd na počítači s procesorom Core2 Duo T7100 1.8 Ghz.

7.1 Diskusia a plány do budúcnosti

Aplikácia bola navrhnutá tak, aby skúmala výsledky jednotlivých metód nad syntetickými dátami. Jedným z možných rozšírení by bolo pridanie segmentačnej časti, ktorá by dokázala spracovávať reálne dáta z ultrazvuku používajúceho pozičný senzor. Z praktického hľadiska je však nutné použiť dostatočné množstvo rezov, pretože inak môžu vznikáť rôzne artefakty, ktoré môžu byť doktorom mylne klasifikované. Vhodným riešením by v takomto prípade mohlo byť zobrazovanie jednotlivých rezov v 3D priestore, pričom by rekonštruovaný model poskytoval iba určité vodítko umožňujúce lepšiu orientáciu.

Interpolačné metódy popísané v tejto práci vypočítavajú hodnoty distance function vždy z dvoch rezov. Všeobecne však možno rozšíriť interpoláciu na celé bunky obkolesujúce skúmaný bod.

Základným prípadom by bolo použitie ortogonálnych, alebo takmer ortogonálnych rezov. V takomto prípade by mohli byť interpolované hodnoty zo štyroch alebo až šiestich rezov, čím by sa teoreticky mohla zväčšiť presnosť rekonštrukcie. Zaujímavá otázka je, či by v takomto prípade bolo v praxi výhodnejšie použiť rovnaké množstvo paralelných alebo ortogonálnych rezov. Nižší počet snímkov je výhodný najmä zo zdravotného hľadiska pri používaní CT, MRI a podobne.

Pri použití obecných rezových rovín a možnosti interpolovať zo všetkých rezov obkolesujúcich daný bod, je zaujímavou otázkou v akých smeroch interpolovať. Najjednoduchším spôsobom by bolo použitie najbližších bodov pre každú rovinu. V takomto prípade by však šlo o ekvivalentný prípad k prípadu zobrazenom na obrázku 3-7, kde by pri určitých rezoch mohli vznikať nepríjemné artefakty. Riešením by mohlo byť použitie algoritmu disc guided. Jeho rozšírenie na viaceré roviny však nie je priamočiare.

Z hľadiska časovej náročnosti nie sú implementované algoritmy príliš optimalizované. Veľké množstvo operácií je možné paralelizovať. Skúmaný priestor, v ktorom sa nachádzajú rezy, je možné rozdeliť na ľubovoľne veľa podpriestorov, pričom v každom z nich môže byť rekonštrukcia vypočítavaná samostatne. Ešte výhodnejší by bol paralelný výpočet distance function, ktorý by prebiehal podobným rozdelením priestoru. V takom prípade by navyše mohli ostať od seba oddelené časti visualisatora a interpolatora. Ďalšie zrýchlenie by mohlo poskytnúť presunutie niektorých výpočtov na grafické karty. V súčasnosti ich výkon pri spracovaní určitého typu dát značne prevyšuje výkon procesorov.

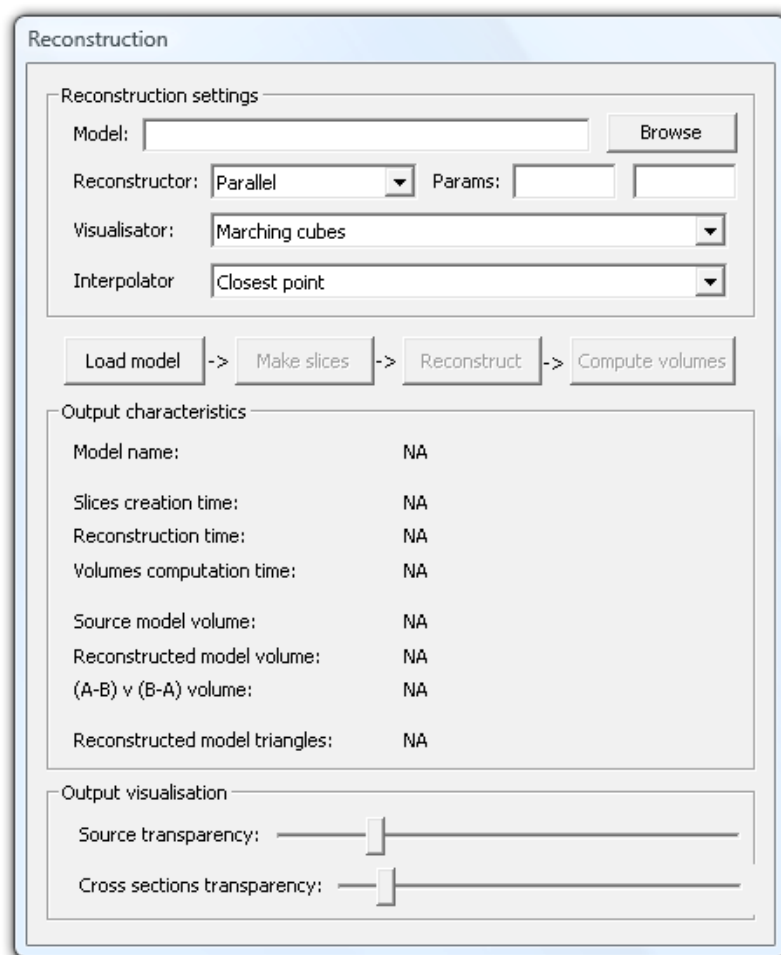
Na grafickú kartu je možné previesť aj takmer celú priamu vizualizáciu skalárneho poľa. Pri rozlíšení 800 x 600 pixlov je možné dosiahnuť niekoľko desiatok obrázkov za sekundu. Priama zobrazovacia metóda má pritom niekoľko výhod a jej implementácia môže byť o dosť jednoduchšia ako implementácia sofistikovaných nepriamych metód.

8 Apendix A – Ovládanie aplikácie RPZNR

Táto kapitola stručne vysvetľuje použitie programu RPZNR.

8.1 Nastavenie parametrov

Aplikácia pozostáva z dvoch dialógových a jedného hlavného okna. Nastavenia a spúšťanie jednotlivých procesov sa vykonáva výlučne pomocou dialógového okna s názvom „reconstruction“ (obrázok 8-1).



Obrázok
8-1 Dialógové okno reconstruction

Toto okno je rozdelené na štyri logické časti. Prvá s názvom „reconstruction settings“ slúži na počiatočné nastavenie všetkých parametrov. Pomocou tlačidla „Browse“ sa zvolí vstupný model vo formáte ASE a pomocou ďalších výberových zoznamov sa určí postupne druh rezov, vizualizačná metóda a spôsob interpolácie. Rezy môžu byť v tomto prípade

vytvorené buď paralelne (Parallel), alebo neparalelne (Random). V oboch prípadoch je potrebné nastaviť aj hodnoty parametrov. Prvý parameter určuje akým spôsobom budú rezy rozložené (tabuľka 8-1) a druhý počet rezov.

parameter	paralel	random
1	Rezy rovnobežné z osami x a z.	Rezy sekvenčne zoradené najmenej vychýľujúce sa od osí x a z.
2	Rezy natočené od roviny x a z o 45 stupňov v oboch smeroch.	Vejárové rezy s centrom naľavo od scény.
3	--	Vejárové rezy s centrom pred scénou.

Tabuľka
8-1 Parametre vytvárania rezov.

Po kompletom nastavení môže začať samotná rekonštrukcia. V strednej časti sa nachádzajú štyri tlačidlá (obrázok 8-2). Tieto je možné stláčať postupne zľava doprava. Vykonalie akcie na jednom z nich vždy odblokuje nasledujúce susediace tlačidlo, a zablokuje všetky ďalšie na pravo. „Load model“ načíta vstupný model, ktorý sa začne okamžite zobrazovať v hlavnom okne. „Make slices“ vytvorí rezy podľa nastavení. „Reconstruct“ spustí samotnú rekonštrukciu. Táto môže trvať pri použití vizualizácie marching cubes a disc guided interpolácie z neparalelných rezov viac ako minútu. Preto sa v takýchto prípadoch odporúča využívať iba regularised marching tetrahedra vizualizáciu. Posledné tlačidlo „Compute volumes“ spustí výpočet približných objemov telies a ich rozdielov. Po ich dopočítaní sa zobrazia výsledky v tretej časti dialógového okna.



Obrázok
8-2 Tlačidlá riadiace rekonštrukciu.

Posledná časť s názvom „Output visualisation“ slúži na nastavenie priesvitnosti rezov a zdrojového modelu v scéne modelu rekonštruovaného.

9 Apendix B – Implementačné dáta a algoritmy

Táto kapitola obsahuje výpis tabuliek používaných v implementáciách použitých algoritmov a časti programov, ktoré tieto dáta využívajú. Všetky deklarácie sú zapísané pomocou syntaxe z C++.

9.1 Dáta použité v implementácii funkcie GetPoint

Jednotlivé tabuľky určujú hodnoty prírastkov, potrebných pripočítať k súradniciam centrálného bodu. Napríklad pre párne **z** a **y** a suseda s indexom **i** získame nové hodnoty súradníc po pripočítaní hodnôt z `SPOEE[i].x`, `SPOEE[i].y`, `SPOEE[i].z`.

```
struct SOffset
{
    int x, y, z;
};

SOffset SPOEE[15] = // prírastky susedných bodov pre párne z a párne y
{
    {-1, 0, -1}, // 0
    {-1, 0, 0}, // 1
    {-1, 0, 1}, // 2

    {0, -1, -1}, // 3
    {0, -1, 0}, // 4
    {0, -1, 1}, // 5
    {1, -1, 0}, // 6

    {0, 0, 1}, // 7
    {1, 0, 0}, // 8
    {0, 0, -1}, // 9

    {0, 1, 1}, // 10
    {1, 1, 0}, // 11
    {0, 1, -1}, // 12
    {0, 1, 0}, // 13

    {0, 0, 0} // 14
};

SOffset SPOOE[15] = // prírastky susedných bodov pre nepárne z a párne y
{
    {0, 0, -1}, // 0
    {-1, 0, 0}, // 1
    {0, 0, 1}, // 2

    {1, -1, -1}, // 3
    {0, -1, 0}, // 4
    {1, -1, 1}, // 5
    {1, -1, 0}, // 6
```

```

{1, 0, 1},          // 7
{1, 0, 0},          // 8
{1, 0, -1},         // 9

{1, 1, 1},          // 10
{1, 1, 0},          // 11
{1, 1, -1},         // 12
{0, 1, 0},          // 13

{0, 0, 0}           // 14
};

SOffset SPOEO[15] = // prírastky susedných bodov pre párne z a nepárne y
{
    {-1, 0, -1},     // 0
    {-1, 0, 0},      // 1
    {-1, 0, 1},      // 2

    {-1, -1, -1},    // 3
    {-1, -1, 0},     // 4
    {-1, -1, 1},     // 5
    {0, -1, 0},       // 6

    {0, 0, 1},        // 7
    {1, 0, 0},        // 8
    {0, 0, -1},       // 9

    {-1, 1, 1},       // 10
    {0, 1, 0},        // 11
    {-1, 1, -1},      // 12
    {-1, 1, 0},       // 13

    {0, 0, 0}         // 14
};

SOffset SPOOO[15] = // prírastky susedných bodov pre nepárne z a nepárne y
{
    {0, 0, -1},       // 0
    {-1, 0, 0},       // 1
    {0, 0, 1},        // 2

    {0, -1, -1},      // 3
    {-1, -1, 0},      // 4
    {0, -1, 1},       // 5
    {0, -1, 0},       // 6

    {1, 0, 1},        // 7
    {1, 0, 0},        // 8
    {1, 0, -1},       // 9

    {0, 1, 1},        // 10
    {0, 1, 0},        // 11
    {0, 1, -1},       // 12
    {-1, 1, 0},       // 13

    {0, 0, 0}         // 14
};

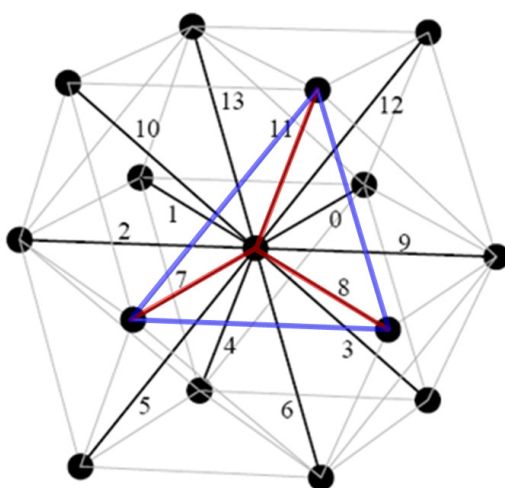
```

9.2 Hodnoty konštantných polí `EdgeReference` a `THEdges`

Pole `EdgeReference` obsahuje dva stĺpce dát po troch riadkoch. Táto štruktúra sa využíva, ak je potrebné získať odkazy na priesečníky hranami, pričom tieto odkazy nie sú uložené v poli `EdgeVertex` centrálného bodu. Prvý stĺpec v takomto prípade obsahuje číslo susedného bodu mriežky, v ktorom sa takýto odkaz nachádza a druhý stĺpec obsahuje index do štruktúry `EdgeVertex`.

Pole `THEdges` obsahuje priamo indexy do štruktúry `EdgeVertex`. Spoločne s použitím obidvoch polí získa program odkazy na všetkých šesť možných priesečníkov ležiacich na hranách jedného štvorstenu.

Na obrázku 9-1 je zobrazený príklad. K priesečníkom ležiacim na červených hranách sa dostaneme pomocou poľa `THEdges`, ktoré poskytne indexy 7, 8 a 11. K modrým sa dostaneme postupne cez jednotlivé vrcholy. Na priesečník s hranou medzi vrcholmi 7 a 8 odkazuje `EdgeVertex` vrcholu 7 s indexom 9. Tieto dva čísla sú uložené na prvom riadku v poli `EdgeReference`.



Obrázok
9-1 Využitie konštánt `EdgeReference` a `THEdges`

```
int THEdges [6][3] =
{
    {7, 8, 11},
    {8, 9, 11},
    {9, 11, 12},
    {11, 12, 13},
    {10, 11, 13},
    {7, 10, 11}
};
```

```
int EdgeReference [6][3][2] =
{
    {
        {7, 9},
        {7, 12},
        {8, 13}
    },
    {
        {8, 13},
        {9, 7},
        {9, 10}
    },
    {
        {9, 10},
        {9, 13},
        {12, 7}
    },
    {
        {12, 7},
        {13, 9},
        {13, 8}
    },
    {
        {10, 9},
        {13, 7},
        {13, 8}
    },
    {
        {10, 9},
        {7, 13},
        {7, 12}
    }
};
```

9.3 Test na zhlukovanie v algoritme RMT

Toto je implementácia algoritmu slúžiaceho na detekciu možnosti zhlukovať vrcholy pri zhlukovacej časti vizualizačného algoritmu regularised marching tetrahedra popísaného v sekcii 4.2.3. V trojrozmernom poli `Planes.DataPlane[y % 3][x][z].iEdgeIndex` sú uložené bitové informácie o susedných priesečníkoch. Výsledný zoznam vrcholov, ktoré môžu byť spojené do jedného je uložený v každej položke poľa `surface` pomocou rovnakých bitových značiek ako v prípade `iEdgeIndex`.

```
// koniec v prípade žiadneho priesečníka
if ( Planes.DataPlane[y % 3][x][z].iEdgeIndex == 0x0000 ) continue;

surfaces = 0;
for (int s = 0; s < 6; s++) surface[s] = 0;

cluster = true; // predpoklad že nastane zhlukovanie

if (Planes.DataPlane[y % 3][x][z].iEdgeIndex == 0x3FFF )
    cluster = false; // topologický prípad A
else
{
    // vyhľadávame diery v povrchu
    all_edges = Planes.DataPlane[y % 3][x][z].iEdgeIndex ^ 0x3FFF;
    // prechádzaním vrcholov
    done_edges = 0x3FFF;
    todo_edges = 0x0001;

    while ( !(all_edges & todo_edges) ) todo_edges <<= 1;

    while ( todo_edges )
    {
        int i = 0;
        // nájdi prvý vrchol v zozname
        while ( !((1 << i) & todo_edges) ) i++;
        // tento vrchol už viackrát nekontroluj
        done_edges &= ~(1 << i);
        // pridaj najbližšie vrcholi do zoznamu
        todo_edges |= (all_edges & nearest_edges[i]);
        // ale nie ak už boli spracované
        todo_edges &= done_edges;
    }
    // v prípade že sme neprešli všetky vrcholi, musí existovať viac ako jedna
    // diera

    if ( all_edges & done_edges )
        cluster = false; // topologický prípad B

    if ( cluster )
    {
        all_edges = Planes.DataPlane[y % 3][x][z].iEdgeIndex;

        while ( all_edges )
        {
            done_edges = 0x3FFF;
            todo_edges = 0x0001;
            while ( !(all_edges & todo_edges) ) todo_edges <<= 1;
            while ( todo_edges )
            {
```

```

        int i = 0;
        while ( !( (1 << i) & todo_edges ) ) i++;
        done_edges &= ~(1 << i);
        todo_edges |= (all_edges & nearest_edges[i]);
        todo_edges &= done_edges;
    }
    all_edges &= done_edges;
    surface[surfaces++] = done_edges ^ 0x3FFF;
}

if ( surfaces == 1 ) // test na otevřený povrch
{
    // výpočet vzdialenosti
    for (int i = 0; i < 14; i++)
    {
        SPlanePoint *pPP;
        if ((pPP = Planes.GetPoint(x, y, z, i)) != NULL)
        {
            distance[i] = pPP->fVal;
        }
        else
        {
            distance[i] = 1000.0f;
        }
    }
    distance[14] = Planes.DataPlane[y % 3][x][z].fVal;

    // pre všetky hrany
    for (int i = 0; i < 36; i++)
    {
        if (!(adjacent_edges[i] & ~done_edges) &&
            !(opposite_edges[i] & done_edges) )
        {
            // otestujeme všetky susedné a protiľahlé vrcholy
            int a1 = 0; // nájdi prvý susedný vrchol
            while ( (1 << a1) & ~adjacent_edges[a1] ) a1++;
            int a2 = a1 + 1; // a druhý susedný vrchol
            while ( (1 << a2) & ~adjacent_edges[a2] ) a2++;
            int o1 = 0; // nájdi prvý protiľahlý vrchol
            while ( (1 << o1) & ~opposite_edges[o1] ) o1++;
            int o2 = o1 + 1; // nájdi druhý protiľahlý vrchol
            while ( (1 << o2) & ~opposite_edges[o2] ) o2++;
            if (((distance[a1] >= 0) ^ (distance[a2] >= 0)) ||
                (abs(distance[a1]) <= abs(distance[o1]) &&
                 abs(distance[a1]) <= abs(distance[o2])) ) &&
                (((distance[o1] >= 0) ^ (distance[o2] >= 0)) ||
                 (abs(distance[o1]) <= abs(distance[a1]) &&
                  abs(distance[o1]) <= abs(distance[a2])) ) )
            {
                cluster = FALSE; // topologický prípad C
                break;
            }
        }
    }
}
}

```

10 Apendix C – Obsah DVD

Súčasťou diplomovej práce je aj DVD s takýmto obsahom:

BIN – spustiteľný súbor programu RPZNR

DATA – zdrojové modely vo formáte ASE použité pri testoch rekonštrukčných algoritmov

DOC – PDF verzia tohto dokumentu

SRC – zdrojové súbory potrebné na skompilovanie aplikácie

Resources – visual studio resources súbory

11 Použitá literatúra

1. The Visible Human Project. *National Institutes of Health*. [Online] [posledná návšteva: 4.august, 2008.] <http://www.nlm.nih.gov/research/visible/animations.html>.
2. Meyers, David, Skinner, Shelley and Sloan, Kenneth. *Surface from contours*. s.l. : AMC Trans. Graph, 1992. strany 228-258.
3. Desheng Wang, Oubay Hassan, Kenneth Morgan and Nigel Weatherill. Efficient surface reconstruction from contours based on two-dimensional Delaunay triangulation. *International journal for numerical methods in engineering ISSN 0029-5981*. s.l. : Wiley, Chichester, ROYAUME-UNI, 2006. strany 734-751.
4. Jean-Daniel Boissonnat, Pooran Memari. Shape Reconstruction from Unorganized Cross-sections. *Proceedings of the fifth Eurographics symposium on Geometry processing*. s.l. : Eurographics Association, júl 2007.
5. B. A. Payne, A. W. Toga. Surface reconstruction by Multiaxial Triangulation. s.l. : Computer Graphics and Applications, 1994. strany 28-35.
6. Levin, D. Multidimensional reconstruction by set-valued approximations. *IMA Journal of Numerical Analysis. Clarendon Press Institute Of Mathematics And Its Applications Conference Series Algorithms for approximation*. s.l. : Clarendon Press, April 1986. strany 421 - 431.
7. Udupa, S. P. Raya and J. K. Shape-based interpolation of multidimensional objects. *Medical Imaging, IEEE Transactions on*. 1990. Vol. 9, 1, strany 32–42.
8. Treece, Graham Michael. Volume measurement and surface visualisation in sequential freehand 3D ultrasound. s.l. : University of Cambridge, 2000.
9. A.L. Bogush, A.V. Tuzikov. 3D Object Reconstruction from Non-parallel Cross-sections. *Pattern Recognition*. 2004. Vol. 3, strany 542-545.
10. Bailey, D. G. An Efficient Euclidean Distance Transform. *Lecture Notes in Computer Science*. s.l. : Springer-Verlag, 2004. strany 394-408.
11. Cuisenaire, Olivier. Distance transformations: Fast algorithms and applications to medical image processing. [Online] October 1999. [posledná návšteva: december 5, 2007.] <http://www.tele.ucl.ac.be/PEOPLE/OC/these/node12.html>.
12. Parker S, Shirley P, Livnat Y, Hansen C, Sloan P. Interactive Ray Tracing for Isosurface Rendering. *Proceedings of the conference on Visualization '98*. s.l. : IEEE Computer Society Press, 1998. strany 233-238.

13. John Amanatides, Andrew Woo. A fast voxel traversal algorithm for ray tracing. *Eurographics '87*. 1987.
14. Schwarze, Jochen. Cubic and Quartic Roots. *Graphics gems*. s.l. : Academic Press Professional, 1990. strany 404-407.
15. Gerd Marmitt, Andreas Kleer, Heiko Friedrich, Philipp Slusallek. Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing. *Vision, modeling, and visualization 2004*. s.l. : Akademische Verlagsgesellschaft Aka, 2004. strany 429-435.
16. Purisima, S. L. Chan and E. O. A new tetrahedral tessellation scheme for isosurface generation. *Computers and Graphics*. s.l. : Elsevier, 1999. Vol. 22, strany 83-90.
17. G.M. Treece, R.W. Prager and A.H. Gee. Regularised marching tetrahedra: improved iso-surface extraction. *Computers and Graphics*. s.l. : Elsevier, 1999. Vol. 23, strany 583-598.
18. Grant, Terence J. Nehe Lesson 48. [Online] [posledná návšteva: 17.marec, 2008.] <http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=48>.
19. Frédéric Cazals, Joachim Giesen. Delaunay Triangulation Based Surface. [Online] 2004. [posledná návšteva: 13.jún, 2008.] <http://www.ann.jussieu.fr/~frey/papers/reconstruction/Cazals%20F.,%20Delaunat%20triangulation%20based%20surface%20reconstruction.pdf>.
20. J. Jeong, K. Kim, H. Park and M. Jung. A new method for solving branching problems in surface reconstruction. *International journal, advanced manufacturing technology*. 2000. strany 259-264.
21. Medical ultrasonography. *Wikipedia*. [Online] [posledná návšteva: 15.Apríl, 2008.] http://en.wikipedia.org/wiki/Medical_ultrasonography.
22. Tomography. *Wikipedia*. [Online] [Cited: 15.Apríl, 2008.] <http://en.wikipedia.org/wiki/Tomography>.
23. Thomas Lewiner, Hélio Lopes, Antônio Wilson Viera, Geovan Tavares. Efficient implementation of Marching Cubes' cases with topological guarantees. *Journal of Graphics Tools*. 2003. Vol. 8, strany 1-15.
24. Zimányi, Marek. Metódy získavania medicínskych dát. s.l. : Katedra aplikovanej informatiky OPGSO, FMFI UK, Bratislava, 2005.