Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

DIPLOMOVÁ PRÁCE

Plzeň, 2002

Jan Patera

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Metody extrakce iso-ploch pro volumetrická data a zobrazování skalárních veličin

Plzeň, 2002 Jan Patera

Zadání

Jméno diplomanta: Jan Patera

Název práce:

Metody extrakce iso-ploch pro volumetrická data a zobrazování skalárních veličin

Zásady pro zpracování:

1. Seznamte se s metodami extrakce iso-ploch ze strukturovaných i nestrukturovaných dat a metodami zobrazování.
2. Seznamte se s vnitřními datovými strukturami pro representaci iso-ploch, resp. modifikujte stávající datové struktury v projektu MVE.
3. Realizujte jednotlivé metody a porovnejte jejich vlastnosti, zejména s ohledem na přesnost extrakce iso-ploch. Pro porovnání použijte "syntetická" tělesa.
4. Vyhodnoťte vlastnosti jednotlivých metod, proveďte vzájemné porovnání vzhledem k různým kritériím, např. povrch a objem generovaného objektu apod.
5. Jednotlivé metody optimalizujte s ohledem na rychlost a zaměřte se na možnosti předzpracování a paralelizace k urychlení výpočtu.
6. Realizované algoritmy realizujte jako DLL moduly, začleňte do visualizačního systému MVE.
7. Ověřte činnost modulů na netriviálních scénách, včetně spolupráce s ostatními moduly visualizačního systému MVE.
8. Proveďte řádnou uživatelskou a programovou dokumentaci včetně komentářů v programových jednotkách.
9. Zhodnoťte dosažené výsledky.

Rozsah práce: min. 40 stran textu

Rozsah příloh: dle potřeby

Seznam literatury: dodá vedoucí diplomové práce

Datum odevzdání: dle vyhlášky Katedry informatiky a výpočetní techniky

Vedoucí diplomové práce: Prof. Ing. Václav Skala, CSc.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne *20.5.2002*

. . . . . .

Jan Patera

Poděkování

Na tomto místě bych rád poděkoval svým rodičům za jejich výraznou podporu po celou dobu mého studia. Svému bratrovi za to, že je. Psům Arturovi a Káče, za to, že mě skoro každý den dokázali za poslední půlrok zvednout od počítače a donutili mě tím alespoň čtvrthodinku denně strávit s nimi venku.

Vedoucímu diplomové práce panu Prof. Ing. Václavu Skalovi, CSc. za připomínky k této diplomové práci, za poskytnutí některých studijních materiálů a za umožnění zahraničního studijního pobytu ve Velké Británii.

Abstrakt

V této práci budou popsány různé metody extrakce iso-ploch ze strukturovaných i nestrukturovaných volumetrických dat, jejich výhody, nevýhody a přesnost aproximace hledané iso-plochy.

Klíčová slova: extrakce iso-ploch, Marching Cubes, Marching Tetrahedra, Centered Cubic Lattice, NOISE, kd-strom, intervalový prostor, zobrazování skalárních dat.

Abstract

In this work there will be described different methods for iso-surface extraction from structured and unstructured volumetric data, their advantages, their disadvantages and accuracy of iso-surface approximation.

Keywords: iso-surface extraction, Marching Cubes, Marching Tetrahedra, Centered Cubic Lattice, NOISE, kd-tree, span space, scalar data visualization.

Obsah

Přílohy

1 Úvod

Volumetrická data obsahují informace o vlastnostech uvnitř a na povrchu určitých objektů. Tyto vlastnosti jsou např. hustota, pnutí a proudění. Jinými slovy volumetrická data popisují určitým způsobem strukturu, objektů a to především i v místech, které nejsou lidskému oku přístupné. Například pří sledování pnutí v nějakém výrobku jsme s obtížemi schopni sledovat toto pnutí do jisté míry zvenčí, ale co se děje uvnitř výrobku už nejsme schopni pozorovat.

Jedním způsobem získávání volumetrických dat může být např. speciální třídimenzionální snímač rentgenového záření, který je schopen určit intenzitu odražených rentgenových paprsků od určitého bodu ve 3D. S použitím tohoto snímače lze vytvořit např. 3D mřížku v jejíchž vrcholech jsou naměřené hodnoty, takovou mřížku nazýváme strukturovaná volumetrická data. Velmi známé metody získávání volumetrických dat jsou CT – *Computed Tomography*, SPECT – *Single Photon Computed Tomography* nebo MR – *Magnetic Resonance.*

Prahování je jednoduchá metoda segmentace, v našem případě se jedná o segmentaci volumetrických dat. Prahováním volumetrických dat izolujeme ty části dat, jejichž hodnota je větší (popř. menší) než hodnota prahu. Jednou z možností jak zobrazit takovou segmentaci je zobrazení iso-plochy ve 3D (nebo pro 2D případ iso-čáry ve 2D). Tyto plochy (nebo křivky) spojují body se stejnou hodnotou zkoumané veličiny (prahová hodnota) a tím oddělují jednotlivé podmnožiny získané prahováním. Ze vzhledu iso-ploch (nebo iso-čar) vidíme určitou část informace obsažené ve volumetrických datech. Iso-plocha ve 3D je ekvivalentem iso-čáry, nebo-li vrstevnice ve 2D. Informace týkající se prahování barev (pojmy iso-foty a iso-colory) lze nalézt na [WWW1].

Extrakce iso-ploch je jednou z možností jak lze zobrazovat část informace uchovávané ve volumetrických datech, která nás právě zajímá. Tedy iso-plocha odstraňuje ty informace, jež jsou pro nás nepodstatné nebo přebytečné a charakterizuje určitým způsobem zkoumaná volumetrická data. Některé metody extrakce iso-ploch jsou použitelné pouze na určitý druh volumetrických dat (např. pouze na strukturovaná volumetrická data), jiné metody mají univerzální použití. V současné době, kdy dochází k rychlému vývoji nejen počítačů, ale i vstupních zařízení, dochází také k nárůstu rozlišení a velikosti volumetrických dat, která je potřeba interaktivně a co nejpřesněji zobrazovat. Z toho plynou vysoké nároky na metody extrakce iso-ploch.

Další možností jak zobrazovat volumetrická data jsou například metody založené na principu sledování paprsku (*ray tracing*).

Obecně lze metody používané pro extrakci iso-ploch popsat následujícím způsobem:

- Vyhledání všech aktivních buněk – tj. z množiny všech buněk (kapitola 2.1) volumetrických dat vybrat pouze ty, které jsou hledanou iso-plochou protnuty.
- Aproximovat hledanou iso-plochu v rámci každé buňky pomocí trojúhelníkové sítě – tento bod zahrnuje např. vyhledání hran, které iso-plocha protíná; aproximace vrcholů trojúhelníků na hranách buňky; aproximace normálových vektorů ve vrcholech trojúhelníků.
- Zobrazení nalezené trojúhelníkové sítě – např. *rendering* nebo zobrazení několika iso-ploch zároveň s probarvením jednotlivých iso-ploch v závislosti na prahové hodnotě.

V této práci se budeme zabývat především porovnáváním vlastností jednotlivých metod (doba extrakce, přesnost aproximace iso-plochy pomocí trojúhelníkové sítě). K porovnání budou použita generovaná tělesa koule a torus.

2 Metoda *Marching Cubes*

Jedná se o jednu z prvních metod pro extrakce iso-ploch z volumetrických dat. *Marching Cubes* je v podstatě rozšířením 2D metody *Marching Squares* do 3D. *Marching Squares* se používá při extrakci iso-čar. *Marching Cubes* je relativně jednoduchá metoda vzhledem k jiným metodám extrakce iso-ploch. Metoda je stále používána v praxi, ale především jako vzorová metoda, tedy jako metoda s níž jsou nově vyvinuté metody srovnávány co se týče paměťových nároků a algoritmické složitosti. Paměťové nároky metody jsou však i v současné době její silnou stránkou. Algoritmická složitost metody je *O(N)*, kde *N* je počet všech buněk ve volumetrických datech.

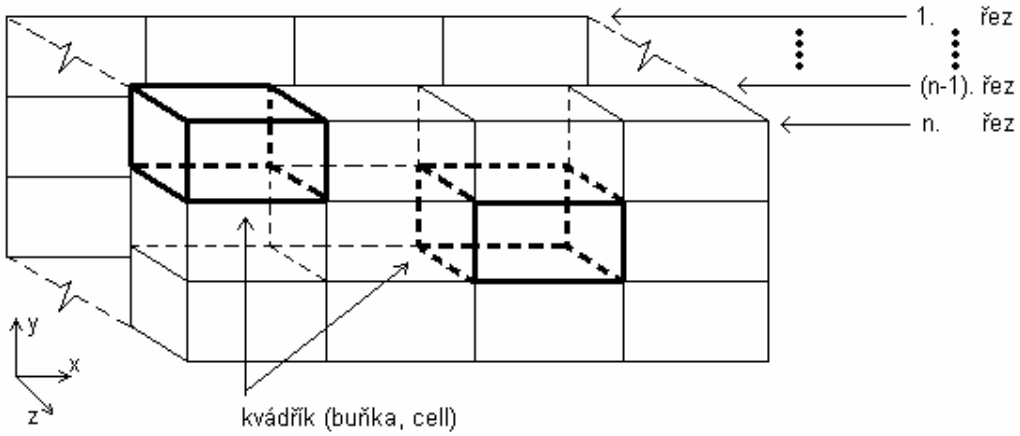
Marching Cubes je metoda pro generování iso-ploch z diskrétních prostorových (3D) strukturovaných volumetrických dat. Iso-plocha odděluje dva různé objemy. Vstupem algoritmu je uživatelem zadaná prahová hodnota (*threshold value*), která určuje hledanou iso-plochu. Dalším vstupem je tří rozměrná mřížka, v jejíchž vrcholech jsou uloženy hodnoty vzorků (volumetrická data). Z každých dvou sousedních řezů mřížky (ve směru

některé souřadné osy) lze sestrojit množinu buněk (buňka = kvádřík = čtyři hodnoty z obdélníku prvního řezu a další čtyři hodnoty z odpovídajícího obdélníku ve druhém řezu). Algoritmus postupně zpracovává vždy dva po sobě jdoucí řezy. Výstupem algoritmu je trojúhelníková síť, která aproximuje hledanou iso-plochu. Paměťové nároky pro algoritmus jsou minimální, pokud zpracováváme jednotlivě všechny buňky, které je možné z volumetrických dat sestrojit.

Jednotlivé kroky algoritmu jsou naznačeny na následujících řádcích (jejich detailní popis je na následujících stránkách):

1) Vytvoření buňky (*cell*) a určení osmi vrcholových hodnot
2) Porovnání těchto hodnot s prahovou hodnotou
3) Určení indexu do tabulky trojúhelníků
4) Výpočet normálových vektorů ve vrcholech buňky
5) Určení hran, na kterých leží vrcholy trojúhelníků
6) Výpočet přesné polohy vrcholů trojúhelníků na hranách buňky
7) Výpočet normálových vektorů ve vrcholech trojúhelníků

2.1 Vytvoření buňky a určení vrcholových hodnot



Obr. 2.1.1: Buňka ve strukturovaných volumetrických datech

Jak již bylo řečeno jsou volumetrická data uložena ve vrcholech pravoúhlé mřížky. Algoritmus zpracovává vždy buňky, které lze sestrojit ze dvou po sobě jdoucích řezů mřížky s volumetrickými daty, pak čtyři hodnoty z obdélníku v prvním řezu a další čtyři

hodnoty z odpovídajícího obdélníku v následujícím řezu tvoří buňku ve tvaru kvádru, viz obr. 2.1.1. Algoritmus postupně zpracovává všechny buňky, které lze vytvořit ze všech po sobě jdoucích řezů mřížky, kolmých na jednu ze souřadných os. Celkový počet buněk, které jsou zpracovávány je roven:

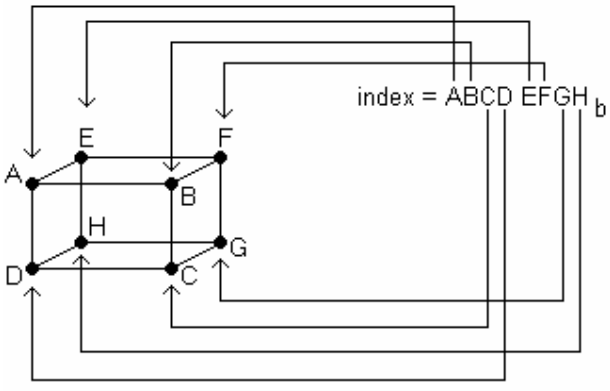
$$(\textit{rozlišení v } X - 1) * (\textit{rozlišení v } Y - 1) * (\textit{rozlišení v } Z - 1)$$

Například při rozlišení volumetrických dat *256*256*256* tak dostáváme přibližně *16* miliónů buněk.

2.2 Porovnání hodnot s prahovou hodnotou

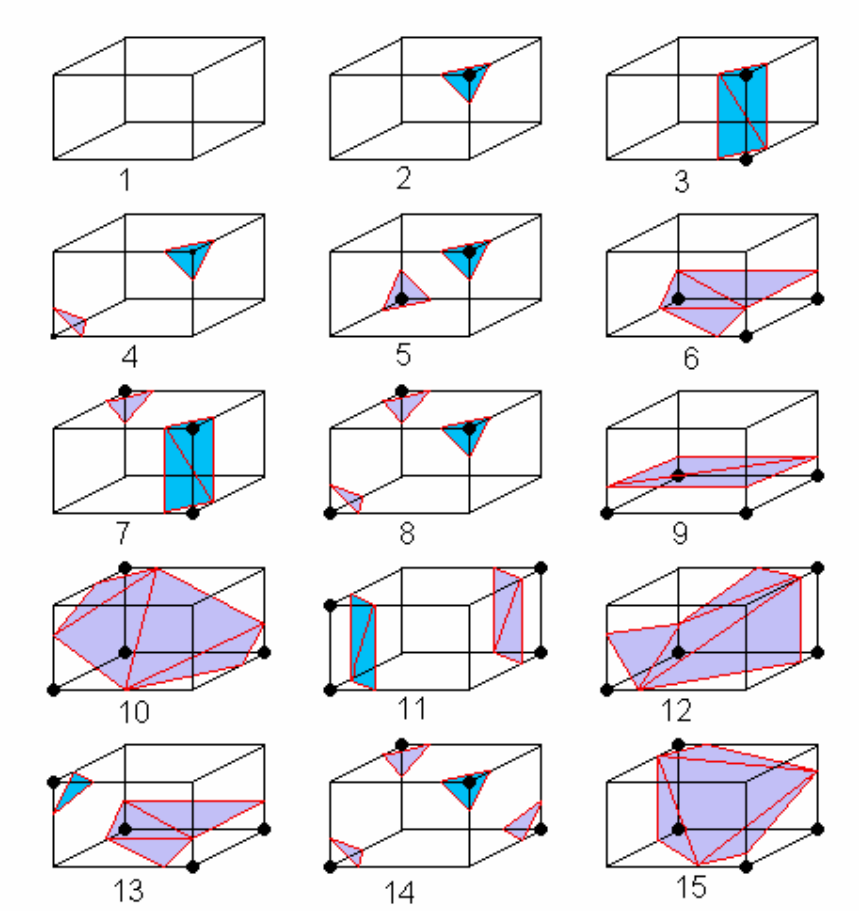
Nyní určíme zda hledaná iso-plocha prochází právě zpracovávanou buňkou. Prahová hodnota určuje hledanou iso-plochu. Pokud jsou hodnoty ve všech vrcholech buňky větší než nebo menší než zadaná prahová hodnota, pak hledaná iso-plocha touto buňkou neprochází (buňka leží mimo hledanou iso-plochu). Takové buňky pro nás nemají význam, protože nijak nesouvisí s hledanou iso-plochou. V takovém případě pokračujeme prvním krokem algoritmu a začneme zpracovávat následující buňku. Pokud však alespoň jeden vrchol (ne všechny!) buňky má hodnotu větší než prahová hodnota a zbývající vrcholy menší než prahová hodnota (nebo naopak), pak hledaná iso-plocha prochází touto buňkou a bude nutné ji v rámci buňky aproximovat pomocí trojúhelníkové sítě.

2.3 Určení indexu do tabulky trojúhelníků



Obr. 2.3.1: Vytvoření indexu pro jednu buňku

Vzhledem k tomu, že metoda *Marching Cubes* pracuje s celými buňkami je celkový počet možností jak může iso-plocha buňku protínat roven *256*. Jelikož každý vrchol může ležet uvnitř iso-plochy (hodnota ve vrcholu je větší než prahová hodnota, bit indexu je roven jedné) nebo vně iso-plochy (hodnota ve vrcholu je menší než prahová hodnota, bit indexu je roven nule). Dostáváme tedy dvě možnosti pro každý z osmi vrcholů buňky, tudíž $2*2*2*2*2*2*2*2 = 2^8 = 256$ možností. Označíme-li vrcholy buňky *A, B, C, D, E, F, G* a *H* pak index do tabulky trojúhelníků lze uvažovat jako osmibitové binární číslo $index=ABCDEFGH_b$, kde každý bit reprezentuje právě jeden vrchol buňky, viz obr. 2.3.1. Všech *256* způsobů jak může iso-plocha protínat buňku lze zredukovat na *15* základních způsobů (viz obr. 2.3.2), všech *256* způsobů pak lze odvodit rotací buňky, negací stavu vrcholů (indexu) a symetrií podle os.



Obr. 2.3.2: Patnáct základních způsobů jak může být buňka protnuta

Dvě sousední buňky vždy sdílí jednu stěnu a tedy i část indexu, tím je zajištěna návaznost ploch mezi jednotlivými buňkami. Ve speciálních případech může na sdílené stěně vzniknout nejednoznačnost, která se ve výstupní trojúhelníkové síti projeví vznikem díry na takové stěně, viz kapitola 2.9.

2.4 Výpočet normálových vektorů ve vrcholech buňky

Víme, že právě zpracovávanou buňkou prochází hledaná iso-plocha. Pro pozdější výpočet normálových vektorů ve vrcholech trojúhelníků, jimiž budeme hledanou iso-plochu aproximovat, je nutné nyní vypočítat normálové vektory ve vrcholech buňky. Tyto vektory odhadneme. Odhad vychází ze skutečnosti, že hledaný povrch (iso-plocha) se nachází mezi dvěma objemy, které se liší svojí hodnotou (dána volumetrickými daty). Směr normálového vektoru je totožný se směrem vektoru gradientu dat $\tilde{N}\mathbf{f} = (g_0, g_1, g_2)$. Jelikož neznáme přesný průběh funkce (= objekt z něhož jsou volumetrická data získána), ale známe pouze její vzorky (volumetrická data), je možné aproximovat gradient dat pomocí symetrické diference hodnot vzorků umístěných ve vrcholech mřížky. Označme obecný vrchol mřížky jako $X=[x_0, x_1, x_2]$, pak symetrické diference vzorků jsou dány matematickým vztahem:

$$g_0 = \frac{f(x_0 + a, x_1, x_2) - f(x_0 - a, x_1, x_2)}{2a}$$
$$g_1 = \frac{f(x_0, x_1 + b, x_2) - f(x_0, x_1 - b, x_2)}{2b}$$
$$g_2 = \frac{f(x_0, x_1, x_2 + c) - f(x_0, x_1, x_2 - c)}{2c}$$

Kde a, b, c jsou vzdálenosti jednotlivých vzorků v mřížce volumetrických dat ve směru souřadných os x, y, z od právě zpracovávaného vzorku. Pro aproximaci gradientu dat u buněk umístěných na okraji zpracovávaného objemu je nutné použít jednu nebo více jednostranných diferencí, viz následující vztahy (všechny diference jsou jednostranné => levý zadní roh volumetrických dat):

$$g_0 = \frac{f(x_0 + a, x_1, x_2) - f(x_0, x_1, x_2)}{a}$$
$$g_1 = \frac{f(x_0, x_1 + b, x_2) - f(x_0, x_1, x_2)}{b}$$
$$g_2 = \frac{f(x_0, x_1, x_2 + c) - f(x_0, x_1, x_2)}{c}$$

Pokud se jedná o krychličku (všechny strany jsou stejně dlouhé) je $a=b=c=k$. Vztahy se pak podstatně zjednoduší na:

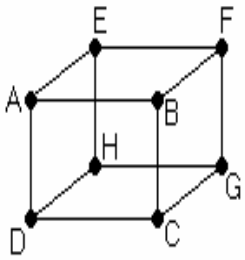
$$g_0 = f(x_0 + k, x_1, x_2) - f(x_0 - k, x_1, x_2)$$
$$g_1 = f(x_0, x_1 + k, x_2) - f(x_0, x_1 - k, x_2)$$
$$g_2 = f(x_0, x_1, x_2 + k) - f(x_0, x_1, x_2 - k)$$

Aproximace normálových vektorů je vhodná na vyhlazená data (data zbavená šumu). Jiný způsob výpočtu normálových vektorů je použití metody adaptivního gradientního stínování, základní informace viz [Zara98], str. 207.

2.5 Určení hran a přesné polohy vrcholů trojúhelníků na nich

V tabulce konfigurací trojúhelníků je 256 řádek, pro všech 256 možných průchodů iso-plochy skrze buňku. Každá řádka obsahuje hrany buňky, na kterých leží vrcholy trojúhelníků a trojúhelníky, jimiž aproximujeme hledanou iso-plochu v rámci jedné buňky. Na těchto hranách budeme interpolovat jednotlivé vrcholy trojúhelníků. Maximálně je pro aproximaci hledané iso-plochy v rámci jedné buňky použito čtyř trojúhelníků.

Hrany a trojúhelníky v tabulce trojúhelníků (index viz kapitola 2.3):



index=0 ... iso-plocha buňkou neprochází = *0* trojúhelníků
index=1 ... iso-pl. protíná *3* hrany *DH,GH,EH* = *1* trojúhelník
index=2 ... iso-pl. protíná *3* hrany *CG,FG,GH* = *1* trojúhelník
index=3 ... iso-pl. protíná *4* hrany *CG,DH,FG,EH* = *2* trojúhelníky
index=4 ... iso-pl. protíná *3* hrany *BF,EF,FG* = *1* trojúhelník
index=5 ... iso-pl. protíná *6* hran *BF,DH,EF,FG,GH,EH* = *2* trojúh.
... atd.

Jelikož budeme hledat souřadnice bodu mezi dvěma koncovými body hrany buňky, provedeme výpočet přesné polohy vrcholu trojúhelníka na hraně buňky pomocí lineární interpolace. Použijeme vztah pro lineární interpolaci:

$$X_P = X_A + \frac{X_B - X_A}{B - A} * (P - A)$$

Kde A, B, P, X_A, X_B jsou známé veličiny a X_P je souřadnice námi hledaného bodu, který leží mezi body X_A a X_B. A, B jsou hodnoty volumetrických dat. P je prahová hodnota. X_A, X_B jsou souřadnice koncových bodů hrany na níž interpolujeme. Výpočet plochy trojúhelníka a trojúhelníkové sítě je popsán v kapitole 11.1.

2.6 Výpočet normálových vektorů ve vrcholech trojúhelníků

Nyní potřebujeme určit normálové vektory pro všechny vrcholy trojúhelníků v rámci právě zpracovávané buňky. Vrcholy všech trojúhelníků leží na hranách buňky. Již máme spočtené normálové vektory ve všech vrcholech buňky a tyto nyní použijeme. Normálové vektory umístěné ve vrcholech trojúhelníků budeme interpolovat pomocí lineární interpolace vektorů ve vrcholech buňky, podobně jako jsme interpolovali souřadnice vrcholů trojúhelníků na hranách buňky. Budeme interpolovat každou souřadnici normálového vektoru zvlášť, viz následující vztahy:

$$\vec{Q}_X = \vec{A}_X + \frac{(\vec{B}_X - \vec{A}_X)}{B - A} * (P - A)$$

$$\vec{Q}_Y = \vec{A}_Y + \frac{(\vec{B}_Y - \vec{A}_Y)}{B - A} * (P - A)$$

$$\vec{Q}_Z = \vec{A}_Z + \frac{(\vec{B}_Z - \vec{A}_Z)}{B - A} * (P - A)$$

Kde $\vec{Q}$ je výsledný normálový vektor ve vrcholu trojúhelníka; $\vec{A}, \vec{B}$ jsou normálové vektory ve vrcholech buňky mezi nimiž se interpoluje; P je prahová hodnota; A,B jsou hodnoty volumetrických dat ve vrcholech buňky.

2.7 Výhody a nevýhody

Výhody:

- jednoduchá metoda pro extrakci iso-ploch z volumetrických dat
- nejmenší počet výstupních trojúhelníků aproximujících hledanou iso-plochu z metod *Marching Cubes*, *Marching Tetrahedra* a *Centered Cubic Lattice*
- minimální nároky na paměť potřebnou pro výpočet

Nevýhody:

- obecně velký počet výstupních trojúhelníků => pomalejší zobrazování iso-plochy
- ve výstupní trojúhelníkové síti se mohou vyskytovat díry, viz kapitola 2.9
- algoritmická složitost *O(N)*, kde *N* je počet buněk je pro data o větším rozlišení poměrně vysoká.

2.8 Vznik děr

V některých konfiguracích může být buňka protnuta více než jedním způsobem. Takový případ nastává tehdy, jsou-li na některé stěně buňky splněny následující podmínky:

- oba koncové body všech hran leží na opačných stranách iso-plochy (jeden bod má v indexu bit nula a druhý bit jedna)
- dva body ležící na diagonále stěny mají stejné hodnoty bitů v indexu

Pak takovou stěnu buňky nazýváme dvojznačná stěna. Buňky číslo 4, 7, 8, 11, 13 a 14 (je jich šest) z obr. 2.3.2 obsahují takovou stěnu. Pokud dvě sousední buňky sdílejí dvojznačnou stěnu a jedna z buněk je doplňkem (negace indexu) k jedné ze základních konfigurací, pak na takové stěně vznikne díra. Existuje několik způsobů jak tento problém řešit. Jedním řešením je rozšíření původní tabulky konfigurací buněk o doplňkové případy (je jich šest), které jsou sestaveny tak, aby zachovaly návaznost iso-plochy mezi sousedními buňkami a zároveň nedocházelo ke vzniku děr na dvojznačných stěnách. Doplňkové případy obsahují v některých konfiguracích i více než čtyři trojúhelníky na buňku. Jiným způsobem je z lokální koherence buněk odhadnout místo, kde vznikne díra. Díru pak pomocí dvou trojúhelníků zašít. Detailnější informace viz [Schroeder98].

3 *Marching Tetrahedra* s dělením na 6 tetrahedronů

Jednotlivé kroky algoritmu se téměř shodují s kroky algoritmu *Marching Cubes*. Jediný rozdíl spočívá v rozdělení nalezené buňky na šest čtyřstěnů (tetrahedronů) a zpracování každého čtyřstěnu zvlášť.

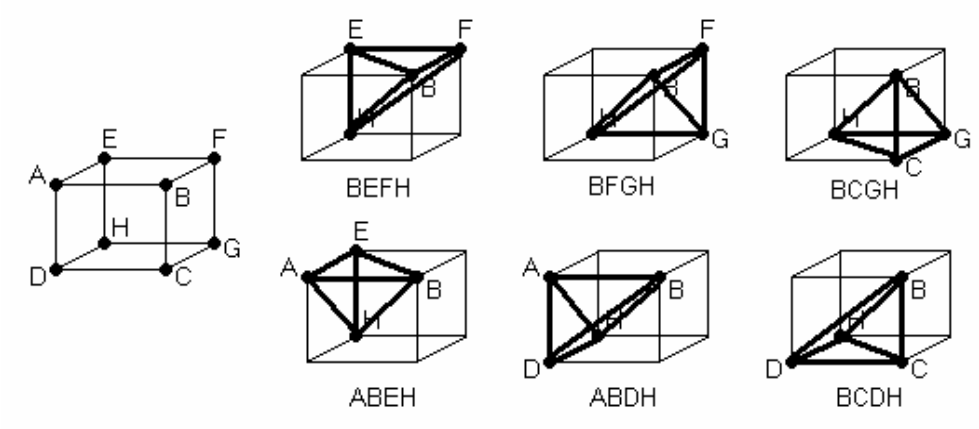
Kroky algoritmu jsou následující:

1) Nalezení buňky a určení osmi vrcholových hodnot
2) Porovnání těchto hodnot s prahovou hodnotou
3) Výpočet normálových vektorů ve vrcholech buňky
4) Rozdělení buňky na šest tetrahedronů
5) Pro každý tetrahedron, určení indexu do tabulky trojúhelníků
6) Pro každý tetrahedron určení hran, na nichž leží vrcholy trojúhelníků

7) Pro každý tetrahedron, výpočet přesné polohy vrcholů trojúhelníků na hranách tetrahedronu
8) Pro každý tetrahedron, výpočet normálových vektorů ve vrcholech trojúhelníků

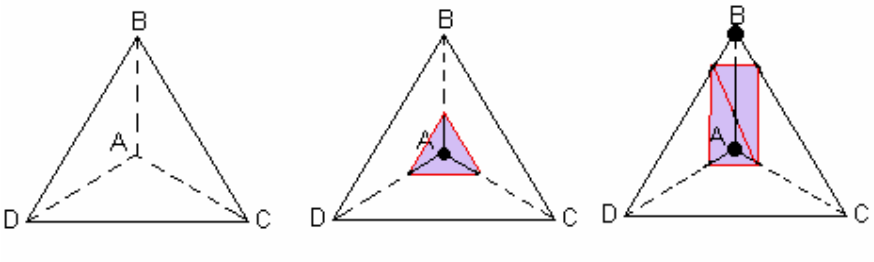
3.1 Rozdělení buňky na 6 tetrahedronů

Buňku lze rozdělit na šest tetrahedronů několika způsoby, viz [Krejza00] str. 16, kde je uvedeno šest způsobů nebo dva způsoby v [Terrades99] na str. 13. My jsme zvolili dělení z [Bourke97] a to způsobem na obr. 3.1.1.



Obr. 3.1.1: Jedna z možností jak lze rozdělit buňku na šest tetrahedronů

3.2 Určení indexu do tabulky trojúhelníků



Obr. 3.2.1: Základní způsoby jak může iso-plocha protínat tetrahedron

Index je určován stejným způsobem jako u buňky v metodě *Marching Cubes*, viz kapitola 2. Tetrahedron má čtyři vrcholy, takže celkový počet způsobů jak může iso-plocha tetrahedron protínat je pouze $2*2*2*2 = 2^4 = 16$ na rozdíl od celé buňky, kde jich bylo

256. Těchto *16* způsobů lze zredukovat díky symetrii, rotaci a inverzi na tři základní případy, viz obr. 3.2.1.

3.3 Výhody a nevýhody

Výhody:

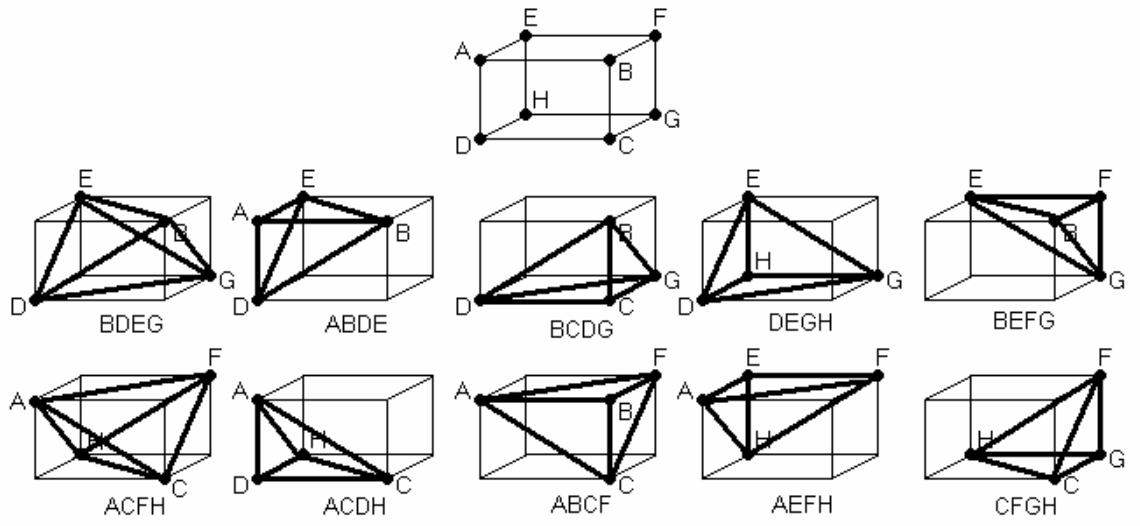
- odstraňuje nevýhodu algoritmu *Marching Cubes*, ve výstupní trojúhelníkové síti nejsou díry
- podstatně jednodušší implementace tabulky konfigurací trojúhelníků než u metody *Marching Cubes*, kde jich bylo *256*
- všechny tetrahedrony vzniklé rozdělením buňky jsou stejné (objem i rozměry, viz obr. 3.1.1)

Nevýhody:

- další nárůst počtu výstupních trojúhelníků oproti metodě *Marching Cubes* a mírný nárůst oproti metodě *Marching Tetrahedra* s dělením na pět tetrahedronů. *Marching Cubes* extrahuje maximálně čtyři trojúhelníky v rámci jedné buňky oproti této metodě, která extrahuje maximálně dva, ale v rámci jednoho tetrahedronu (v buňce jich je šest).
- nárůst času potřebného pro extrakci iso-plochy oproti metodě *Marching Cubes*. Důvodem je dělení buňky na tetrahedrony a zároveň nárůst počtu hran, které může iso-plocha protínat z *12* na *18*.

4 Marching Tetrahedra s dělením na 5 tetrahedronů

Dělení je podobné jako u *Marching Tetrahedra 6*, buňka je však rozdělena pouze na pět tetrahedronů. Existují dva způsoby jak buňku rozdělit na pět tetrahedronů. Při dělení buňky na pět tetrahedronů je nutné oba způsoby dělení pravidelně střídat z důvodu návaznosti trojúhelníkové sítě mezi sousedními buňkami (3D šachovnice, viz kapitola 6.4), jak je patrné z obr. 4.1.



Obr. 4.1: Dva způsoby dělení buňky na pět tetrahedronů

4.1 Výhody a nevýhody

Výhody:

- ve výstupní trojúhelníkové síti nejsou díry
- podstatně jednodušší implementace tabulky konfigurací trojúhelníků
- nižší počet výstupních trojúhelníků oproti metodě *Marching Tetrahedra 6*

Nevýhody:

- nárůst počtu výstupních trojúhelníků oproti metodě *Marching Cubes*. Důvod je stejný jako u *Marching Tetrahedra 6*.
- nárůst času potřebného pro extrakci iso-plochy oproti metodě *Marching Cubes*, důvodem je dělení buňky na tetrahedrony, nutnost střídání dvou různých dělení a nárůst maximálního počtu hran, které může iso-plocha protínat v rámci jedné buňky z 12 na 18 hran.
- tetrahedrony nemají stejný tvar

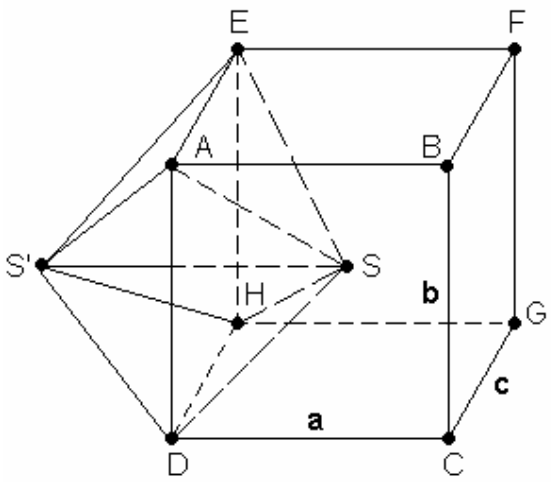
5 Centered Cubic Lattice

Jedná se o další metodu pro pravidelná volumetrická data. Dochází zde k dělení buňky na 24 tetrahedronů, přičemž všechny tetrahedrony jsou sdíleny i sousedními buňkami, viz obrázek 5.1.

Hrany tetrahedronů jsou buď krátké (vedoucí do vnitřního vrcholu AS, DS, viz obr. 5.1) nebo dlouhé (hrany buňky AE, AD, popř. hrany mezi středy sousedních buněk SS'). Vrcholy tetrahedronů jsou taktéž dvojího druhu. Vrcholy ležící na hranách buňky (A, B, C, D, ...) a vrcholy ležící ve středu buňky (S, S', ...). Hodnotu ve vrcholu uvnitř buňky lze spočítat např. prostým aritmetickým průměrem (použito v našem případě) nebo pomocí trilineární interpolace, viz např. [Zara98]. Jednotlivé normálové vektory počítáme pomocí vztahu (uveden pouze vztah pro první složku výsledného normálového vektoru, vztahy pro ostatní složky jsou obdobné):

$$n_0 = q*\left(\frac{f(x_0+a,x_1,x_2)-f(x_0-a,x_1,x_2)}{2*a}\right)+$$
$$\frac{1-q}{4}*\left(\frac{f\left(x_0+\frac{a}{2},x_1+\frac{b}{2},x_2+\frac{c}{2}\right)-f\left(x_0-\frac{a}{2},x_1+\frac{b}{2},x_2+\frac{c}{2}\right)}{a}\right)+$$
$$\frac{1-q}{4}*\left(\frac{f\left(x_0+\frac{a}{2},x_1+\frac{b}{2},x_2-\frac{c}{2}\right)-f\left(x_0-\frac{a}{2},x_1+\frac{b}{2},x_2-\frac{c}{2}\right)}{a}\right)+$$
$$\frac{1-q}{4}*\left(\frac{f\left(x_0+\frac{a}{2},x_1-\frac{b}{2},x_2-\frac{c}{2}\right)-f\left(x_0-\frac{a}{2},x_1-\frac{b}{2},x_2+\frac{c}{2}\right)}{a}\right)+$$
$$\frac{1-q}{4}*\left(\frac{f\left(x_0+\frac{a}{2},x_1-\frac{b}{2},x_2+\frac{c}{2}\right)-f\left(x_0-\frac{a}{2},x_1-\frac{b}{2},x_2+\frac{c}{2}\right)}{a}\right)$$

Kde *a, b, c* jsou rozměry buňky ve směru jednotlivých souřadných os, viz obr. 5.1. Písmeno *q* je váhový parametr pro výpočet normálového vektoru, přičemž *q* je z intervalu <0; 1>. Parametr *q* určuje vliv hodnot umístěných ve středu osmi okolních buněk (vrchol S na obr. 5.1) na právě počítanou souřadnici normálového vektoru ve vrcholu buňky (totéž platí pro středový vrchol).



Obr. 5.1: Centrované dělení buňky – jedna stěna (čtyři tetrahedrony)

5.1 Výhody a nevýhody

Výhody:

- ze sdílení tetrahedronů sousedními buňkami plyne lepší návaznost generované trojúhelníkové sítě mezi sousedícími buňkami
- všechny tetrahedrony mají stejný tvar

Nevýhody:

- každá buňka je dělena v průměru na 12 tetrahedronů, z toho plyne nárůst generovaných trojúhelníků
- nutnost aproximace volumetrických dat ve středu buňky
- náročnější výpočty v rámci jedné buňky

6 Urychlení výpočtu

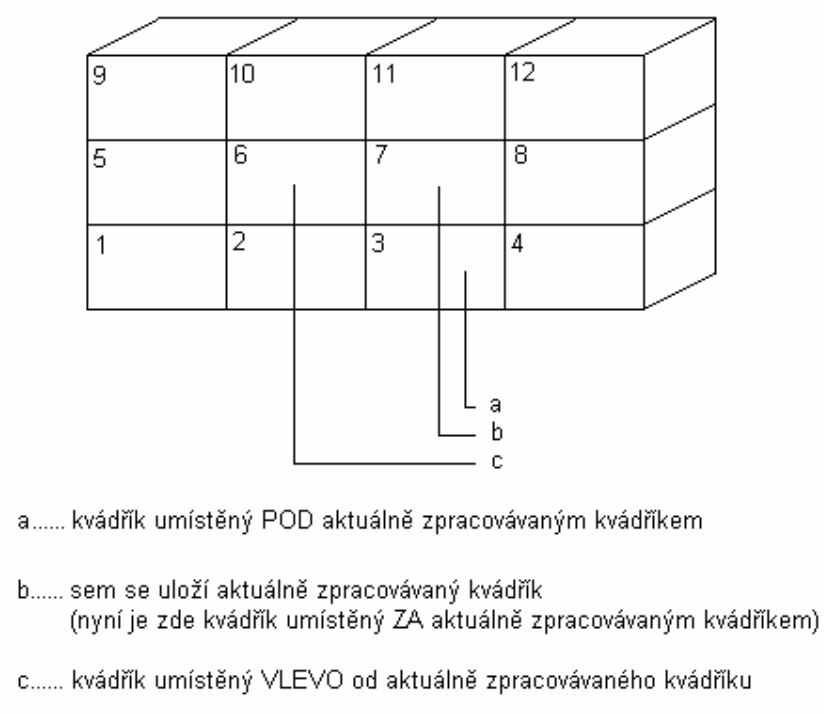
Na následujících stránkách jsou popsány různé způsoby urychlení extrakce iso-ploch ze strukturovaných i nestrukturovaných volumetrických dat. V našem případě bylo použito urychlení s využitím lokální koherence (kapitola 6.2) pro metody *Marching Cubes*, *Marching Tetrahedra 5* a *Marching Tetrahedra 6*. Urychlení s využitím redukce buněk

a lokální koherence (kapitola 6.4) pro metodu *Centered Cubic Lattice* a urychlení vyhledávání aktivních buněk (kapitola 6.5) v metodě *NOISE* (kapitola 6.5.3).

6.1 Urychlení vynecháním interpolace

Lze použít pro strukturovaná i nestrukturovaná volumetrická data. Pokud nám záleží více na rychlosti výpočtu než na přesnosti aproximace iso-plochy lze vynechat výpočetně náročnou interpolaci vrcholů trojúhelníků a gradientů na hranách buňky a místo interpolace vrchol jednoduše umístit do středu hrany. Jedná se o urychlení na úkor kvality výstupní trojúhelníkové sítě.

6.2 Urychlení s využitím lokální koherence mezi buňkami



Obr. 6.2.1: Eliminace množství interpolací pomocí bufferu

Vhodné pro strukturovaná volumetrická data. Urychlit výpočet lze použitím bufferu. Velikost bufferu je taková, aby se do něho vešel jeden řez buněk, tedy:

$$(rozlišení\ v\ X - 1) * (rozlišení\ v\ Y - 1) * elem$$

Kde *elem* je velikost paměti potřebné pro uchování dat o jedné buňce. Dochází tedy k nárůstu paměťové složitosti metody. Data uchovávaná v bufferu pro jednu buňku jsou normálové vektory ve vrcholech buňky, průsečíky s iso-plochou na všech hranách buňky, index do tabulky trojúhelníků. Buňky jsou zpracovávány postupně po řádcích od zadního řezu směrem k přednímu.

Jak je patrné z obr. 6.2.1, jsou buňky v rámci řezu zpracovávány podle očíslování od *1* do *12*. Urychlení výpočtu u vnitřních buněk je velmi výrazné. Např. u buňky, která se uloží do bufferu na pozici číslo *7* počítáme pouze část indexu do tabulky trojúhelníků, normálu v bodě *B* a průsečíky na hranách *AB, BC, BF*. Ostatní údaje, tj. část indexu, normály, průsečíky na zbývajících hranách jsou již spočteny a lze je získat z bufferu z pozic číslo *3, 6, 7*. U buněk na okraji objemu, např. *1* až *4* je urychlení méně výrazné.

6.3 Celkové předzpracování

Lze použít jak pro strukturovaná tak i pro nestrukturovaná volumetrická data. Dalším způsobem jak urychlit extrakci iso-ploch je předzpracování. Při každé změně prahu je nutné hledat nově vzniklou iso-plochu a celý objem je procházen znovu, což je časově velmi náročné. Pro všechny možné hodnoty prahu jsou získané trojúhelníkové sítě uschovány a při změně prahové hodnoty se pouze zamění výstupní trojúhelníková síť za jinou. Při výpočtu všech iso-ploch lze využít globální koherence mezi jednotlivými iso-plochami.

6.4 Urychlení s využitím redukce buněk a lokální koherence

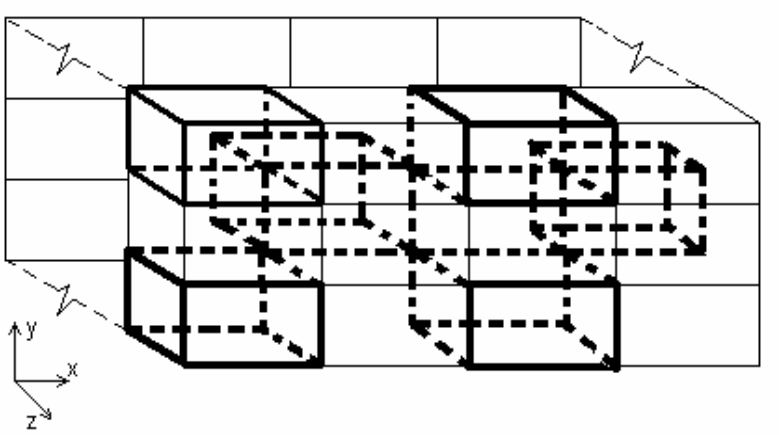
Vhodné pro strukturovaná volumetrická data. Jiný způsob využití lokální koherence mezi buňkami (podobný použití bufferu) je popsán v [Cignoni97]. Autoři redukují buňky ve vstupní pravidelné mřížce pomocí tzv. 3D šachovnice. Takto lze omezit počet všech buněk, u kterých zkoumáme zda jsou protnuty iso-plochou či nikoliv. 3D šachovnice je definována takto:

- Máme dánu pravidelnou mřížku *IxJxK* (tedy je dán objem složený z *(I-1)*(J-1)*(K-1)* buněk ve tvaru kvádru). Černá políčka (na obr. 6.4.1 vykreslena tučně) jsou pak na těchto pozicích *[2*i+k%2, 2*j+k%2, k]*, kde *i=0..(I-2-k%2)/2,*

j=0..(J-2-k%2)/2 a *k=0..(K-2)*. Symbol *%* značí dělení modulo, např. výraz *(5 % 2)=(5 mod 2)=1*.

- Jinými slovy máme-li černou buňku *X[i, j, k]* uvnitř objemu, pak okolní černé buňky (je jich osm) jsou ty buňky co s buňkou *X[i, j, k]* sdílejí právě jeden vrchol.

Počet černých buněk je přibližně 25% z celkového počtu všech buněk, tedy dochází zde k paměťové úspoře a urychlení celkového průchodu buňkami. Je však nutné speciálně ošetřit průchod buňkami na okraji objemu popsaného daty, protože u nich nelze zcela využít lokální koherence z okolních buněk (některé neexistují).



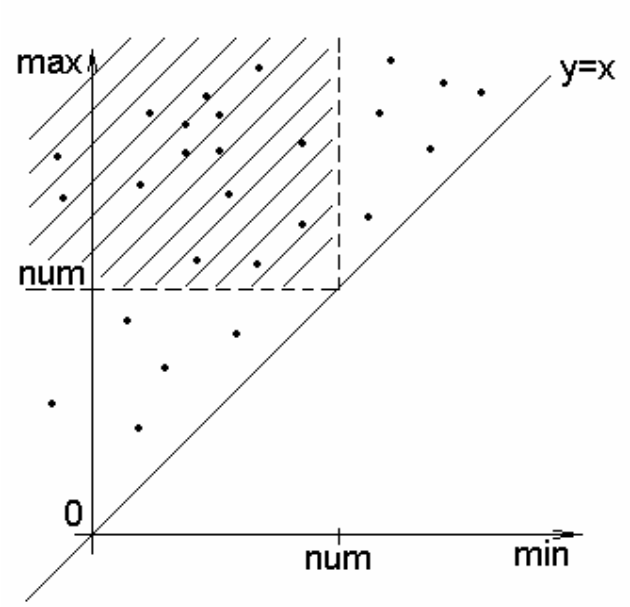
Obr. 6.4.1: Třírozměrná šachovnice

6.5 Urychlení vyhledávání aktivních buněk

Lze použít jak pro strukturovaná tak i pro nestrukturovaná volumetrická data. Metoda slouží k velmi rychlému nalezení aktivních buněk, viz [Livnat99]. Aktivní buňky jsou vyhledávány na základě *span space* (intervalový prostor) reprezentace. Tímto lze dosáhnout toho, že celý objem nemusí být při každé změně prahu prohledáván znovu. *Span space* reprezentaci lze implementovat pomocí kd-stromu s paměťovou složitostí *O(N)*, kde *N* je počet buněk. Kd-strom je stromová struktura vytvořená nad volumetrickými daty. Kd-strom je pro daná volumetrická data vytvořen pouze jednou v rámci *offline preprocesingu*. Jiný způsob implementace *span space* je uveden v [Cignoni97] a to pomocí intervalového stromu, který umožňuje mírně rychlejší vyhledání aktivních buněk, ale za cenu daleko vyšších paměťových nároků pro reprezentaci intervalového stromu.

6.5.1 Intervalový prostor (*Span Space*)

V [Livnat99] je představena *span space* reprezentace množiny intervalů. V této reprezentaci přísluší každému intervalu bod ve dvourozměrném prostoru. Na osu *x* nanášíme minimum a na osu *y* maximum z daného intervalu. Ve speciálním případě, kdy pro určitý interval je minimum daného intervalu rovno maximu, leží bod na přímce *y=x*. V ostatních případech leží body nad touto přímkou (minimum daného intervalu je tedy ostře menší než maximum), viz obr. 6.5.1.1.

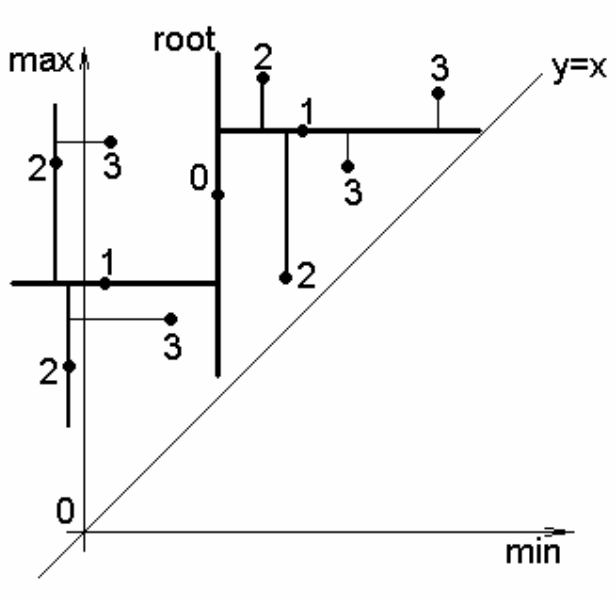


Obr. 6.5.1.1: Reprezentace intervalů jako body ve 2D

Hledáme-li, ve kterých intervalech leží např. číslo *num*, znamená to, že hledáme takové body (=intervaly), jejichž souřadnice na ose *max* je větší než číslo *num* (ve speciálním případě rovno) a zároveň souřadnice na ose *min* je menší než číslo *num* (popř. ve speciálním případě rovno), tedy číslo leží v intervalu reprezentovaném bodem. Intervaly obsahující číslo *num* jsou reprezentovány body ve vyšrafované oblasti na obr. 6.5.1.1. V našem případě bude každé buňce (kvádru, tetrahedronu) odpovídat jeden interval (maximální a minimální hodnota ve vrcholech buňky).

6.5.2 Kd-strom (*Kd-tree*)

Struktura kd-strom je v podstatě vícerozměrný binární vyhledávací strom. Každý uzel kd-stromu (otec) obsahuje data a dva podstromy (syny). V levém podstromu jsou ve všech uzlech hodnoty menší a naopak v pravém podstromu jsou ve všech uzlech hodnoty větší než je hodnota obsažená v otcovském uzlu. Binární vyhledávací stromy slouží k vyhledávání dat v jednorozměrném prostoru. Kd-stromy umožňují vyhledávat v *N* rozměrném prostoru, díky tomu, že se v každé úrovni kd-stromu pravidelně střídají jednotlivé dimenze dat. Např. body ve 2D, v sudých úrovních stromu budou rozděleny podle hodnoty souřadnice *x* a v lichých úrovních pak podle hodnoty souřadnice *y*. Kd-strom lze použít jako strukturu pro implementaci intervalového prostoru (*span space*), kde kd-strom bude sloužit k vyhledávání intervalů obsahujících určité číslo. Ukázka vyváženého kd-stromu vykresleného v intervalovém prostoru je na obr. 6.5.2.1.



Obr. 6.5.2.1: Kd-strom (čísla = úrovně kd-stromu)

V našem případě uzly kd-stromu reprezentují intervaly hodnot volumetrických dat ve vrcholech jednotlivých buněk. Pak po zadání iso-plochy (prahové hodnoty) nalezneme poměrně rychle aktivní buňky průchodem kd-stromu. Jak je kd-strom implementován je popsáno v kapitole 8.4. Na tomto principu je založena metoda *NOISE* (viz kapitola 6.5.3 na následující stránce).

6.5.3 Metoda *NOISE*

NOISE je zkratka vytvořená ze slov *Near Optimal Iso Surface Extraction* (téměř optimální extrakce iso-ploch). Princip metody spočívá v rychlém nalezení aktivních buněk za pomoci kd-stromu a aproximaci hledané iso-plochy pomocí trojúhelníkové sítě v rámci těchto buněk.

Jednotlivé kroky metody jsou následující:

1. Konstrukce kd-stromu nad danými volumetrickými daty (pokud strom již existuje, jeho načtení ze souboru)
2. Vyhledání aktivních buněk rekurzívním průchodem kd-stromu
3. Aproximace iso-plochy v rámci aktivních buněk

V kapitole 8.4 je uvedena implementace kd-stromu, jeho konstrukce a postup při jeho průchodu. Sestrojení kd-stromu je také naznačeno v kapitole 6.5.2. Iso-plocha je v rámci buňky aproximována již dříve popsaným způsobem, viz např. metoda *Marching Cubes*, která byla popsána v kapitole 2.

7 Zobrazování skalárních a neskalárních veličin

7.1 Zobrazování skalárních veličin

Třírozměrná pole skalárních veličin lze zobrazovat pomocí výše popsaných metod pro extrakce iso-ploch. Další možností je použití metod sledování paprsku (*ray tracing*) nebo vrhání paprsku (*ray casting*), které jsou detailně popsány v [Zara98] včetně nejčastěji používaných způsobů urychlení. Jinou možností je třírozměrné pole skalárních veličin převést na vektorové pole a to zobrazit (viz kapitola 7.2).

Pokud chceme zobrazovat dvourozměrná pole skalárních veličin, lze využít metody *Maching Squares*, ze které byla odvozena metoda *Marching Cubes* pro 3D případ. Výstupem algoritmu *Marching Squares* jsou iso-čáry (vrstevnice). Obdobně jako ve 3D případě lze dvourozměrné pole skalárních veličin převést na vektorové pole a to zobrazit pomocí způsobů uvedených v kapitole 7.2 (viz níže).

Méně často se setkáváme s potřebou zobrazovat jednorozměrné pole skalárních veličin. To lze zobrazit pomocí bodového grafu (popř. pomocí sloupcového grafu) ve 2D a to tak, že každé celočíselné hodnotě na ose x (1..N) odpovídá jedna skalární hodnota uložená v poli (výsledkem je bod ve 2D).

7.2 Vektorová pole

Vektorová data (nejčastěji 2D nebo 3D) reprezentují směr a velikost nějaké veličiny (ve 2D nebo 3D). Data jsou získávána např. studiem proudění kapalin, vzduchu, atd. V následujících kapitolách budou naznačeny základní způsoby zobrazování vektorových polí. Některé z níže uvedených přístupů jsou již implementovány v knihovně *VTK* (*Visualization Toolkit* – *www.kitware.com*), jako např. třídy `vtkGlyph3D`, `vtkStreamLine` (viz obrázky A.9 a A.10 v příloze A). Obecně lze vektorová data převést na skalární a ty zobrazovat pomocí metod pro extrakce iso-ploch.

7.2.1 Orientované úsečky

Nejednodušší možnost jak tyto data zobrazit je vykreslení úsečky pro každý vektor pole, jelikož známe umístění, směr a velikost vektorů. Tato technika bývá označována *hedgehog* (ježek). Můžeme také místo úseček zobrazovat orientované šipky, které mohou být obarveny na základě své velikosti nebo směru (popř. teploty nebo tlaku v daném bodě). Také lze místo úseček použít orientované glyfy (*glyph*), přičemž glyfem rozumíme 2D nebo 3D orientovaný objekt (trojúhelník, jehlan, atd.). Glyf může znázorňovat i zakřivení, zkroucení atp. Výsledek při použití knihovny VTK je na obrázku A.10 v příloze A.

Potíže nastávají při zobrazování většího množství vektorů, kdy vektory vystupují spíše jako body než jako orientované úsečky. Různě dlouhé vektory se po promítnutí na rovinu jeví jako vektory o stejné délce, viz [Zara96].

7.2.2 Deformování (*Warping*)

Vektorová data často znázorňují pohyb (vzduch, tekutina, atd.). Vektory například charakterizují rychlost toku v daném bodě nebo směr posunu částic. Toho lze využít a data zobrazit pomocí *warpingu* (deformace) geometrických útvarů (úsečka ve 2D, rovina

ve 3D, atp.). Například můžeme do vektorového pole, které znázorňuje tok tekutiny, vložit kolmo rovinu a tu na základě znalosti pole v daném místě deformovat (zakřivit, natáhnout, zkroutit, atp.). Jsou-li vektory dány pouze na povrchu nějakého tělesa, lze použít obdobu *warpingu* a to zobrazení změny (*displacment plots*).

Ve speciálních případech však může dojít k tomu, že se např. rovina zdeformuje tak, že bude protínat sama sebe nebo deformace nebude naopak vůbec znatelná. V takovém případě bude zcela určitě pozorovatel poněkud zmaten. Další informace lze nalézt v [Schroeder98].

7.2.3 Časová animace (*Time animation*)

Můžeme si představit pohyb bodu (nebo jiného objektu) ve vektorovém poli za malý časový úsek, např. zobrazení vektoru jako úsečky je v podstatě lineární aproximace trajektorie bodu (***x(t)***) pohybujícího se ve směru vektoru rychlosti (***v***) za malý časový úsek (*dt*), tedy:

$$\vec{x}(t) = \int_t \vec{v} * dt$$

Použitím animování bodů (popř. objektů, např. bublinek = koulí) přes mnoho časových úseků získá pozorovatel představu o tvaru vektorového pole.

Hlavním úskalím je zde volba délky časového úseku. Příliš dlouhý úsek může mít za následek přeskočení některých částí pole. Příliš krátký úsek pak zkrácení celkově uražené trasy oproti skutečnosti.

Modifikacemi tohoto způsobu zobrazení vektorových polí jsou např. techniky *particle tracing*, *streamlines* (viz obrázek A.9 v příloze A), *streaklines* a další. Podrobněji viz [Schroeder98] a [Zara98].

8 Systém MVE a použité datové struktury

8.1 Úvod do MVE

Písmena MVE jsou zkratkou slov *Modular Visualization Environment* (modulové visualizační prostředí). Systém byl vyvinut v rámci projektu na Západočeské univerzitě

v Plzni. Tento systém dovoluje uživateli vizuálně sestavit vlastní aplikaci (schéma) pomocí modulů na načítání, zobrazování, ukládání dat a výpočty. Některé moduly nemusí mít vstup, jiné zase výstup. Takto sestavené aplikace jsou pak v systému MVE spustitelné. Systém se skládá ze dvou základních částí:

- **Editor** – v něm lze poskládat různé moduly a vzájemně je propojit mezi sebou do výsledné aplikace
- **Runtime** – koordinuje činnost jednotlivých modulů (spouští je sériově, popř. paralelně, atd.)

Modulem rozumíme objekt, který je schopen ze vstupních dat a svého nastavení produkovat data výstupní. Moduly jsou v podstatě DLL (Dynamic Link Library) knihovny s definovanou strukturou. Každý modul musí pro Runtime část poskytovat minimálně následující funkce:

- jménoModulu_MAIN_MODULE_FUNC – hlavní funkce modulu
- jménoModulu_FREE_DATA – uvolnění paměti alokované modulem
- jménoModulu_SETUP_FUNC – nastavení modulu (setup)
- jménoModulu_FREE_SETUP_DATA – uvolnění paměti alokované v setup
- jménoModulu_FREE_STATE – uvolnění stavu modulu
- Get_Modules – pro získání informací o modulech v této DLL knihovně
- Free_DLL_Descr – uvolnění dat alokovaných v Get_Modules

Podrobné informace o tom jak vytvořit modul pro MVE systém lze získat ze stránek projektu MVE a to na [WWW3]. Na následujících řádcích jsou popsány použité datové struktury pro implementaci dříve popsaných metod jako modulů v MVE systému.

V modulech vytvořených v rámci této diplomové práce byly použity vstupní a výstupní datové struktury, které jsou popsány v následujících kapitolách. Jedná se o moduly VolDataGenerator (generuje volumetrická data nebo je načte ze souboru, popř. umí načíst ze souboru do paměti i kd-strom), IsoSurfaceExtraction (umožňuje volbu metody pro extrakci iso-plochy z volumetrických dat a dokáže uložit výstupní data do souboru) a SimpleRenderer (zobrazuje trojúhelníkovou síť). Výstupem modulu VolDataGenerator a vstupem modulu IsoSurfaceExtraction jsou volumetrická data (kapitola 8.3). Výstupem modulu IsoSurfaceExtraction a vstupem modulu SimpleRenderer

je trojúhelníková síť (kapitola 8.2). Moduly jsou z uživatelského hlediska podrobněji popsány v příloze A (anglicky).

8.2 Iso-plocha v MVE

Iso-plocha je reprezentována jako trojúhelníková síť touto datovou strukturou (výpis je v jazyce ANSI C, struktura je umístěna v souboru "`triangle_types.h`"):

```
typedef struct{
       T_Float xmin,xmax, ymin,ymax, zmin,zmax;   //bounding box
       T_sFloat Res_x, Res_y, Res_z;              //resolution
       T_Float Pos_x, Pos_y, Pos_z;               //position
       T_Float Rot_x, Rot_y, Rot_z;               //rotation
       T_Float Dis_x, Dis_y, Dis_z;               //distortion

       T_Index NV_M; //memory space allocated for vertices
       T_Index NT_M; //memory space allocated for triangles
       T_Index NV;   //number of truly stored vertices
       T_Index NT;   //number of truly stored triangles

       P_Coord P_VCoor[3];         //vertices coordinates (x,y and z)
       P_Coord P_VNorm[3];         //normal vector for each vertex
       P_Index P_TV[3]; //triangle vertices (3 pointers to P_VCoor)
       P_Index P_TT[3];            //triangle neighbors if any
       P_Coord P_TNorm[3];         //triangles normal vectors

       P_Vertex_Status P_VStatus;  //state flags of vertices
       P_Triangle_Status P_TStatus; //data status flag
       T_Byte z_valid;             //data valid flag
} T_Triangle_Mesh, *P_Triangle_Mesh;  //triangle mesh data structure
```

Následuje popis nejdůležitějších prvků výše uvedené datové struktury. `P_VCoor[J]` je dynamicky alokované pole, v němž jsou pro každý vrchol A_J trojúhelníkové sítě uloženy jeho souřadnice A_J=(x, y, z). V poli `P_VNorm[J]` jsou pro každý vrchol A_J z `P_VCoor[J]` uloženy souřadnice jeho normálového vektoru N_J = (a, b, c). A konečně v poli `P_TV[K]` jsou pro každý trojúhelník T_K uloženy tři ukazatele na jeho vrcholy do pole `P_VCoor[]`, T_K = (u, v, w).

8.3 Volumetrická data v MVE

Volumetrická data jsou reprezentována touto datovou strukturou (výpis je v jazyce ANSI C, struktura je umístěna v souboru "`volume_types.h`"):

```
typedef struct{
       char Caption[255];          //describes title or name of a data
       char Source[255];           //describes source of a data
       T_sInt  Res_x, Res_y, Res_z;  //resolution
       T_sInt  Pos_x, Pos_y, Pos_z;  //position
       T_sInt  Rot_x, Rot_y, Rot_z;  //rotation
       T_sFloat Dis_x, Dis_y, Dis_z; //distortion
       P_sInt Data;         //vector of volumetric data values
       T_Int Min, Max;      //min and max intensity of voxels
       T_Int StartTreshold; //not implemented yet
} T_Volume_Data, *P_Volume_Data;  //volumetric data data structure
```

Nejdůležitější prvky jsou popsány na následujících řádcích. `Res_x`, `Res_y` a `Res_z` popisují rozlišení volumetrických dat ve směru souřadných os. `Min` a `Max` konstanty obsahují minimální a maximální hodnotu ze všech vzorků uložených ve volumetrických datech. Samotná volumetrická data jsou uložena v jednorozměrném poli `Data[]`. Na první pozici je vzorek, který se v objemu z něhož volumetrická data pocházejí nalézá vlevo dole a vzadu. Poslední vzorek se v původním objemu nalézá vpravo nahoře a vepředu. Vzorky jsou tedy uloženy po řezech postupně od zadního řezu dopředu (v kladném směru osy z), kde každý řez je složen z jednotlivých řádek (v kladném směru osy x) postupně odzdola nahoru (v kladném směru osy y). Pro data je tedy použit pravotočivý souřadný systém, jak je patrné z obr. 2.1.1.

8.4 Struktura pro kd-strom

Jak již bylo řečeno kd-strom uchovává následující hodnoty v každém ze svých uzlů:

- minimální a maximální hodnotu volumetrických dat pro danou buňku
- odkaz na levý a pravý podstrom
- odkaz na samotnou buňku

Pro metodu *NOISE* (kapitola 6.5.3) je používán vyvážený kd-strom, tzn. že levý a pravý podstrom má vždy stejnou hloubku pro všechny listy. Toto kritérium však nelze vždy splnit, protože by celkový počet uzlů musel být 2^X-1, což je speciální případ. Například pokud je celkový počet uzlů kd-stromu roven čtyřem, pak hloubka listů u dvou podstromů kořene se bude lišit o jedna. Pro uchování takového stromu lze s výhodou použít jednorozměrné pole.

Datová struktura je pak definována jako počet prvků v poli a samotné pole s uzly kd-stromu. Podstrom je v takovém poli definován jako jeho část, tedy interval <*dolní_mez; horní_mez*>. Kořen každého podstromu se pak nalézá na pozici *dolní_mez+(počet_uzlů_v_podstromu)/2*. Tím se zbavíme nutnosti uchovávat odkazy na levý a pravý podstrom a dojde k výrazné úspoře paměti za cenu mírného zvýšení výpočetních nároků (nutnost počítat pozici kořene a meze intervalů pro jednotlivé podstromy pomocí operací +, – a dělení dvěma).

Uzel kd-stromu je v programu definován následující datovou strukturou:

```
//one node of kd-tree
typedef struct kd_node{
  T_sInt min; //minimum vol. data value in this cell
  T_sInt max; //maximum vol. data value in this cell
  T_uInt cell; //pointer to whole cell informations
} KD_NODE;
```

Kd-strom je pak reprezentován speciálně uspořádaným jednorozměrným polem těchto prvků (kapitola 8.4.1). Struktura kd-strom je definována v souboru "`kd_tree.h`".

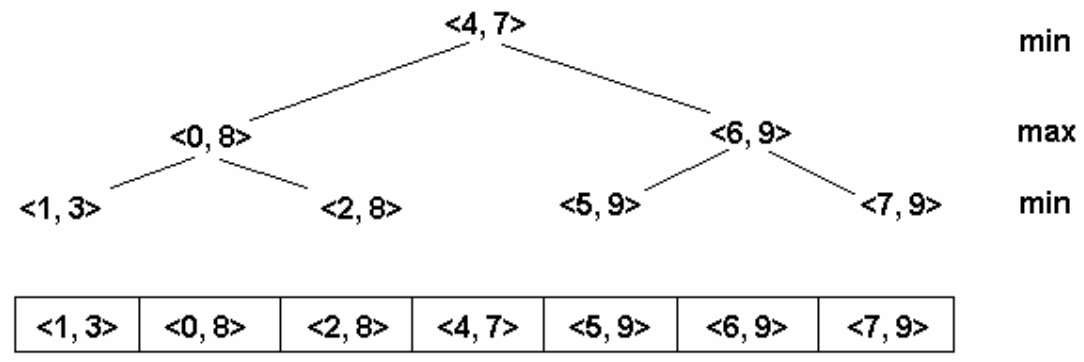
8.4.1 Konstrukce kd-stromu

Na uspořádání prvků v poli do kd-stromu je použit *Wirthův* (viz [Wirth76]) rekurentní algoritmus pro nalezení mediánu (obecně ho lze použít i pro nalezení k-tého nejmenšího prvku), který je několikanásobně rychlejší než použití algoritmu *Quick Sort* (implementován v knihovnách ANSI C), který řadí celou posloupnost.

Postup při konstrukci vyváženého kd-stromu je následující (podobný konstrukci vyváženého binárního stromu):

1. Vytvoříme dynamicky alokované jednorozměrné pole uzlů kd-stromu (typ `KD_NODE`), pole má tolik prvků, kolik je ve volumetrických datech buněk. Každé buňce tedy odpovídá jeden uzel stromu. Pole uzlů naplníme potřebnými údaji: minimální (`KD_NODE.min`) a maximální (`KD_NODE.max`) hodnota dat v buňce, odkaz na kompletní informace o buňce (`KD_NODE.cell`).
2. Nyní uspořádáme prvky pole tak, aby utvářely vyvážený kd-strom. Nalezneme medián (= kořen kd-stromu) v závislosti na hodnotě `KD_NODE.min` a umístíme ho do středu pole. Nalevo od mediánu jsou v poli pouze prvky s menší hodnotou `KD_NODE.min` (levý podstrom), napravo pak prvky s větší hodnotou `KD_NODE.min` (pravý podstrom). Tím jsme v poli vytvořili první úroveň kd-stromu.
3. Rekurzívním způsobem určíme kořeny u podstromů z předchozího kroku, nyní však v závislosti na hodnotě `KD_NODE.max`. Tím získáme sudou (např. druhou) úroveň kd-stromu. Všechny sudé úrovně kd-stromu jsou vytvářeny na základě hodnoty `KD_NODE.max`. Skok na krok 4, tak dlouho dokud není vytvořena poslední úroveň kd-stromu (dokud lze podstromy dále dělit).
4. Rekurzívním způsobem určíme kořeny u podstromů z předchozího kroku, nyní však v závislosti na hodnotě `KD_NODE.min`. Tím získáme lichou (např. třetí) úroveň kd-stromu. Všechny liché úrovně kd-stromu jsou vytvářeny na základě hodnoty `KD_NODE.min`. Skok na krok 3, tak dlouho dokud není vytvořena poslední úroveň kd-stromu.

Vyvážený kd-strom pro intervaly: <7, 9>, <2, 8>, <5, 9>, <1, 3>, <4, 7>, <6, 9>, <0, 8> je na obr. 8.4.1.1.



Obr. 8.4.1.1: Ukázka kd-stromu pro osm intervalů

8.4.2 Průchod kd-stromem

Postup při průchodu kd-stromu je následující:

1. Nalezneme kořen kd-stromu v jednorozměrném poli – kořen je na pozici *0+(počet_uzlů_ve_stromu)/2*, pole je v jazyce C indexováno od nuly. Kořen nám rozdělí pole na dvě části (podstromy). První podstrom je v poli uzlů uložen na pozicích od *0* do *(počet_uzlů_ve_stromu)/2-1*. Druhý podstrom pak od *(počet_uzlů_ve_stromu)/2+1* do *počet_uzlů_ve_stromu*. Údaje uložené v kořenu porovnáme s prahovou hodnotou a rozhodneme, zda buňka odpovídající tomuto uzlu je aktivní či nikoliv. Dále rozhodneme, zda je nutné procházet oba podstromy či nikoliv.
2. Rekurzívním způsobem prohledáme podstromy z předchozího kroku. Nalezneme tedy kořeny obou podstromů, rozhodneme zda jsou jim odpovídající buňky aktivní, atd. Krok 2 opakujeme dokud není prohledán celý kd-strom.

Jak již bylo řečeno liché úrovně stromu jsou řazeny podle hodnoty `KD_NODE.min` a sudé úrovně podle hodnoty `KD_NODE.max`. Při rozhodování zda prohledávat oba podstromy daného kořene (uzlu) na úrovni min (viz obr. 8.3.1) může nastat jedna z následujících možností:

- prahová hodnota je menší než dolní mez intervalu – víme, že v pravém podstromu se aktivní buňky nenacházejí. V pravém podstromu jsou totiž všechny dolní meze intervalů větší než prahová hodnota. Prohledáváme pouze levý podstrom.
- prahová hodnota je větší než dolní mez intervalu – víme, že v levém podstromu je dolní mez vždy menší než prahová hodnota. Můžeme tedy pro celý levý podstrom vynechat testování dolní meze intervalů. Prohledáváme oba podstromy.

Při rozhodování zda prohledávat oba podstromy daného kořene (uzlu) na úrovni max (viz obr. 8.3.1) může nastat jedna z následujících možností:

- prahová hodnota je větší než horní mez intervalu – víme, že v levém podstromu se aktivní buňky nenacházejí. V levém podstromu jsou totiž všechny horní meze intervalů menší než prahová hodnota. Prohledáváme pouze pravý podstrom.

- prahová hodnota je menší než horní mez intervalu – víme, že v pravém podstromu je horní mez vždy větší než prahová hodnota. Můžeme tedy pro celý pravý podstrom vynechat testování horní meze intervalů. Prohledáváme oba podstromy.

V nižších úrovních kd-stromu pak může poměrně snadno nastat případ, kdy jsou vynechány testy na dolní i horní meze intervalů zároveň a jsou tedy zpracovávány pouze aktivní buňky. Výše uvedené rozhodování vynechává ty části kd-stromu, kde se nenacházejí aktivní buňky. Pro 10^9 buněk dostáváme maximální hloubku rekurze pouze 30. Maximální hloubka rekurze je dána následujícím vztahem, kde desetinná místa neuvažujeme:

$$hloubka = \log_2(pocet\_bunek) + 1 = \frac{\ln(pocet\_bunek)}{\ln(2)} + 1$$

8.4.3 Degenerovaná buňka

Degenerovaná buňka je definována jako buňka, která má více než jednu hodnotu rovnu maximální nebo minimální hodnotě ze všech hodnot ve vrcholech buňky. Pak při prahové hodnotě rovné jednomu z extrémů může nastat jedna z následujících situací:

- jedna hodnota je rovna extrému – buňka sdílí s iso-plochou jeden bod
- dvě hodnoty jsou rovny extrému – buňka s iso-plochou sdílí jednu hranou
- více hodnot je rovno extrému – buňka se dotýká iso-plochy jednou stěnou, popř. celá buňka je součástí iso-plochy

První dva případy můžeme ignorovat. Při ignorování třetího případu vznikne ve výsledné iso-ploše díra. Pokud se rozhodneme vykreslit celou stěnu buňky, pak bude vykreslena dvakrát, protože stěna je sdílena dvěma tetrahedrony. Je-li celá buňka součástí iso-plochy pak při jejím vykreslení iso-plocha obsahuje bubliny a jedna stěna je opět vykreslena dvakrát.

V [Livnat99] je jako řešení navrhnuto mírně změnit shodné hodnoty u dvou sousedních buněk, takovým způsobem, že iso-plocha po té bude procházet pouze jedním z tetrahedronů. Jiným řešením může být využití lokální koherence a rozhodnutí co vykreslit na základě porovnání dvou a více sousedních buněk.

9 Správa paměti

Nespornou výhodou operačního systému MS Windows XP je lepší správa paměti než tomu bylo u předchozích verzí. MS Windows XP umožňují totiž dynamicky měnit velikost alokované paměti (pomocí funkce `_expand()`), což u minulých verzí nebylo možné. Jediným způsobem jak změnit velikost již alokované paměti byla možnost realokace (`realloc()`, která alokovala nový úsek paměti a do něho zkopírovala data, po té se zrušila stará alokovaná paměť. Tato zpráva paměti snižuje výrazně dobu potřebnou pro extrakci iso-plochy z rozsáhlých volumetrických dat, kdy dochází k realokaci paměti potřebné pro výstupní trojúhelníkovou síť.

V našem programu je pro realokaci paměti pro výstupní data nejprve použita funkce `_expand()`. Pokud se nepodaří tímto způsobem změnit velikost paměťového bloku, je použita funkce `realloc()`, která funguje i pod staršími operačními systémy. Realokace paměti probíhá po 150 tisících trojúhelníků. Pro měření v kapitole 11.4 bylo alokováno dostačující množství paměti, takové aby nedocházelo v průběhu měření časů k realokacím, které značně zpomalují dobu výpočtu. Paměť alokovaná pro vstupní data se v průběhu výpočtu nemění.

10 Generování těles

Generování volumetrických dat probíhá následovně. Postupně jsou procházeny všechny vrcholy mřížky volumetrických dat (o uživatelem zadaném rozlišení) a pro každý je rozhodnuto zda je uvnitř daného tělesa či mimo něj (na základě implicitní rovnice). Podle toho je zkoumaný vrchol mřížky ohodnocen. Vnitřek tělesa je ohodnocen maximální hodnotou (255 na jednobytových volumetrických datech), objem mimo těleso minimální hodnotou (nula).

10.1 Koule

Body na povrchu koule musí splňovat rovnici: $(x-s_X)^2+(y-s_Y)^2+(z-s_Z)^2-r^2=0$

Body uvnitř koule (bez povrchu) pak nerovnici: $(x-s_X)^2+(y-s_Y)^2+(z-s_Z)^2-r^2<0$

A body vně koule (bez povrchu koule) nerovnici: $(x-s_X)^2+(y-s_Y)^2+(z-s_Z)^2-r^2>0$

Kde *x, y, z* jsou souřadnice bodů ve 3D, s_X, s_Y, s_Z jsou souřadnice středu koule, *r* je poloměr generované koule.

10.2 Torus

Generován stejným způsobem jako koule. Body na povrchu torusu (se středem v počátku a symetrickým podle osy *z*) musí splňovat následující rovnici, viz [WWW2]:

$$\left(c-\sqrt{(x-s_X)^2+(y-s_Y)^2}\right)^2+(z-s_Z)^2-a^2=0$$

Kde *x, y, z* jsou souřadnice bodů ve 3D, *c* je hlavní poloměr torusu (poloměr prstence), *a* je vedlejší poloměr torusu (tloušťka prstence).

11 Porovnání metod

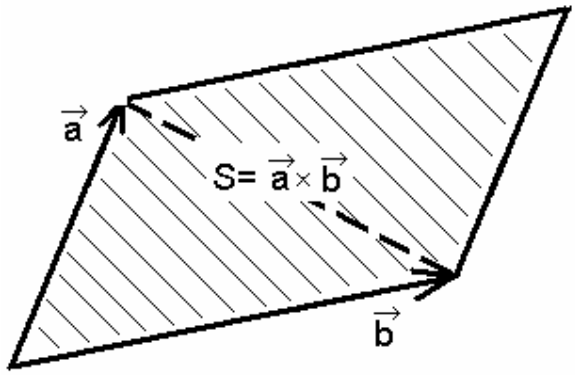
V následující kapitole budou porovnány některé z výše uvedených metod sloužících pro extrakci iso-ploch. Porovnání se bude týkat zejména následujících vlastností metod a výsledné trojúhelníkové sítě:

- doby potřebné pro extrakci iso-plochy (doba zahrnuje vyhledání aktivních buněk a generování trojúhelníků v rámci těchto buněk)
- počtu vygenerovaných trojúhelníků výsledné iso-plochy
- přesnosti aproximace hledané iso-plochy na synteticky generovaných objektech jako je např. koule nebo torus. Kritériem pro porovnání přesnosti extrakce bude např. objem a povrch objektu reprezentovaného výslednou iso-plochou v porovnání s objemem a povrchem synteticky generovaného tělesa.

11.1 Povrch trojúhelníkové sítě

Povrch je roven součtu povrchů jednotlivých trojúhelníků (= námi nalezené iso-plochy). Povrch jednoho trojúhelníka je dán vztahem: $S=\frac{a*v}{2}$

V tomto vztahu je *a* libovolná základna a *v* jí odpovídající výška trojúhelníka.



Obr. 11.1.1: Vektorový součin

Korektnost povrchů sítě byla kontrolována pouze vizuálně. Povrch jednoho trojúhelníka lze také určit pomocí vektorového součinu. Velikost vektoru vzniklého vektorovým součinem dvou vektorů, které odpovídají stranám rovnoběžníka, je rovna obsahu rovnoběžníka. Obsah trojúhelníka je potom polovina z obsahu rovnoběžníka, viz obr. 11.1.1 a tento vztah:

$$S = \left| \frac{\vec{a} \times \vec{b}}{2} \right|$$

11.2 Koule (*Sphere*)

Pro generovanou kouli a jí odpovídající vygenerovanou iso-plochu budeme porovnávat povrch. Povrch koule spočteme ze vztahu:

$$S = 4 * p * r^2$$

Další vlastností metody bude přesnost objemu uzavřeného iso-plochou vzhledem k objemu koule. Objem koule je dán vztahem:

$$V = \frac{4}{3} * p * r^3$$

Objem objektu daného iso-plochou, která aproximuje kouli je roven součtu objemů jednotlivých aktivních buněk (tetrahedronů) a jejich částí, které jsou uvnitř objemu, viz [Terrades99].

11.3 Torus

Objem je dán vzorcem, viz [Rektorys95]:

$$V = 2 * p^{2} * a^{2} * c$$

Kde *c* je hlavní poloměr, *a* je vedlejší poloměr.

Povrch je pak určen vztahem, viz [Rektorys95]:

$$S = 4 * p^{2} * a * c$$

Kde *c* je hlavní poloměr, *a* je vedlejší poloměr.

11.4 Srovnání metod - koule

V následující kapitole jsou pro popis grafů použity zkratky MC (*Marching Cubes*), MT5 (*Marching Tetrahedra 5*), MT6 (*Marching Tetrahedra 6*), CCL (*Centered Cubic Lattice*) a *NOISE* (*Near Optimal Iso Surface Extraction*).

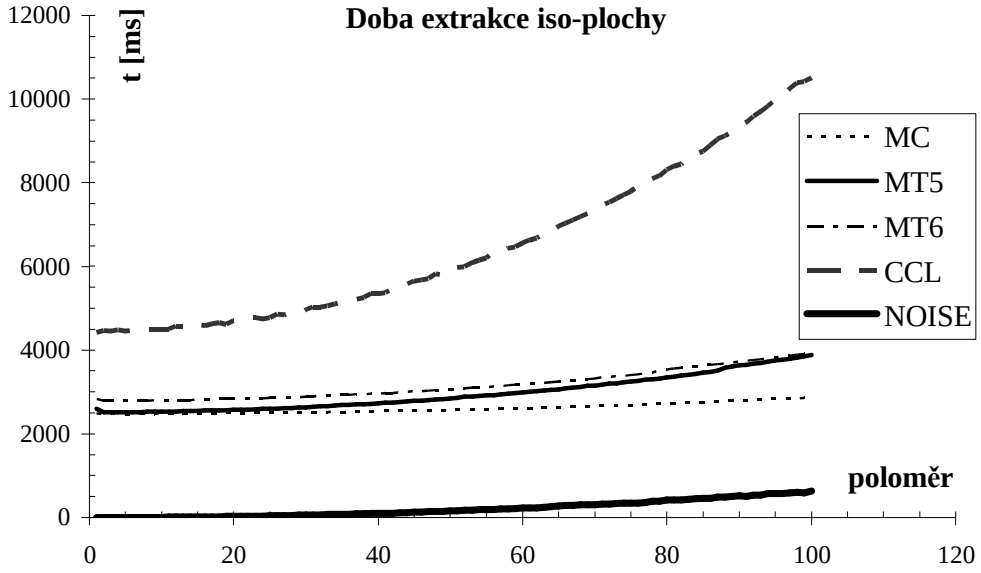
Koule je ve volumetrických datech reprezentována hodnotami 255 a prostor mimo kouli má hodnotu 0. Prahová hodnota je pro všechny metody 40% z rozsahu mezi minimální a maximální hodnotou dat (v tomto případě 0 a 255). Rozlišení volumetrických dat je ve všech směrech rovno 210, koule je umístěna ve středu volumetrických dat, poloměr byl měněn postupně od 1 do 100 s krokem 1. Naměřené hodnoty jsou pospojovány úsečkami. Grafy pro torus viz příloha D.

11.4.1 Doba extrakce iso-plochy

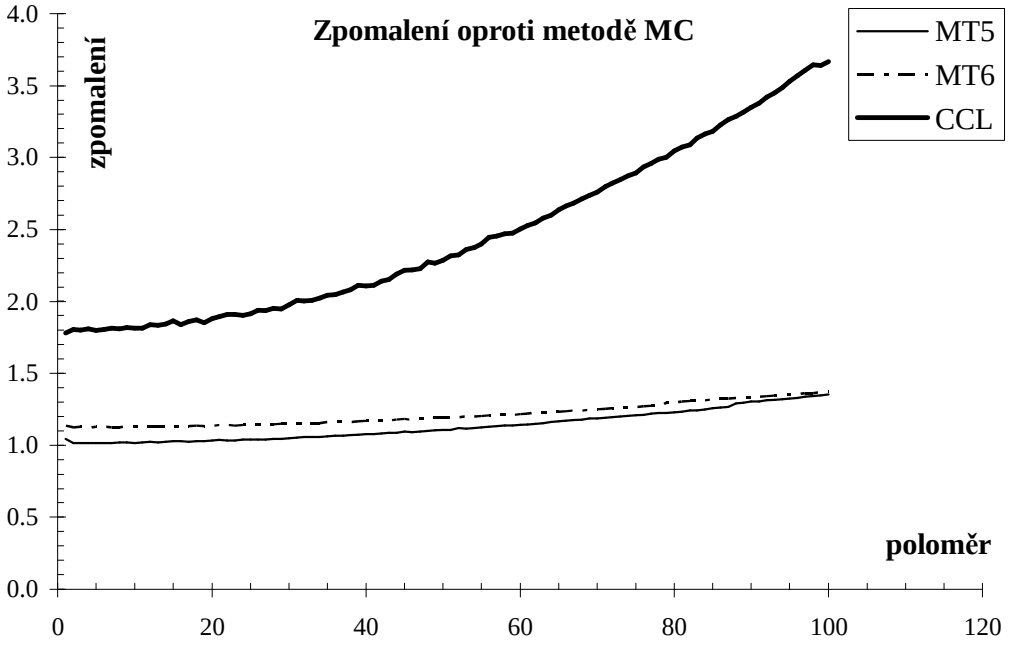
Časy byly naměřeny na PC s procesorem AMD Athlon 1GHz, 256MB RAM a základní deskou MSI, pod operačním systémem Windows XP s využitím hardwarového časovače *performance counter* o kmitočtu 3GHz. Doba zahrnuje čas potřebný pro nalezení aktivních buněk a generování trojúhelníků v rámci těchto buněk.

Metody MC, MT5 a MT6 procházejí vždy všechny buňky volumetrických dat. Metoda CCL prochází pouze cca. 25% ze všech buněk. Pokud žádná buňka není iso-plochou protnuta je vždy zapotřebí konstantní čas k prohledání všech buněk (jeho hodnota závisí na metodě), viz graf 11.4.1.1 pro nulovou hodnotu poloměru. Metoda

NOISE však využívá kd-stromu k nalezení aktivních buněk, postupně jsou ignorovány ty části stromu, kde se nenacházejí aktivní buňky, čímž dochází ke značnému urychlení doby extrakce iso-plochy, viz graf 11.4.1.1. Výsledná doba také značně závisí na počtu generovaných trojúhelníků a tím i na počtu interpolací.



Graf 11.4.1.1: Doba extrakce iso-plochy

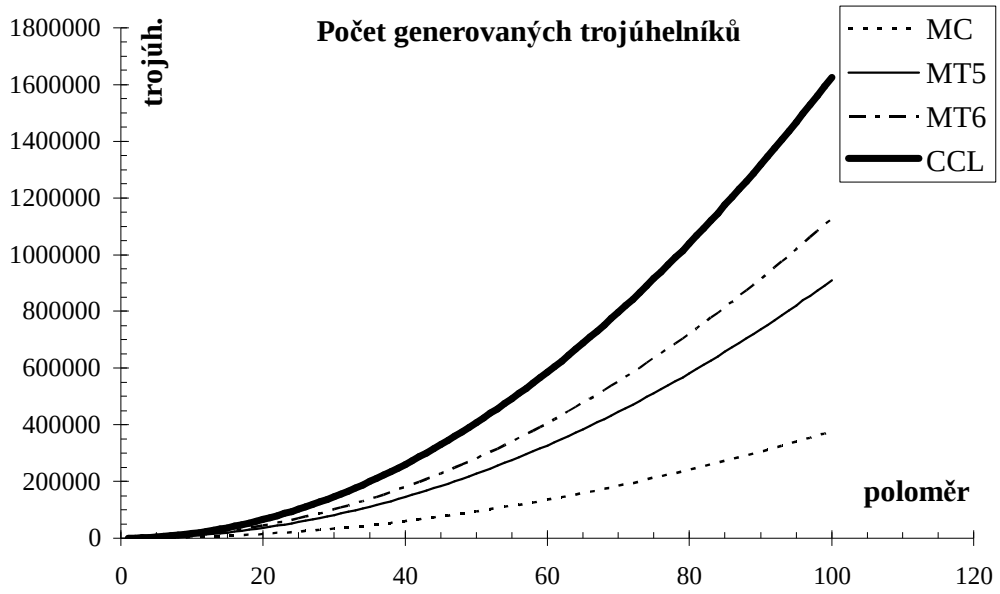


Graf 11.4.1.2: Zpomalení oproti metodě MC

Zpomalení (graf 11.4.1.2) doby extrakce iso-plochy vzhledem k metodě MC je pro jednotlivé metody definováno vztahy:

$$Zpomalení\_MT5 = \frac{T_{MT5}}{T_{MC}}, \quad Zpomalení\_MT6 = \frac{T_{MT6}}{T_{MC}}, \quad Zpomalení\_CCL = \frac{T_{CCL}}{T_{MC}}$$

11.4.2 Počet vygenerovaných trojúhelníků

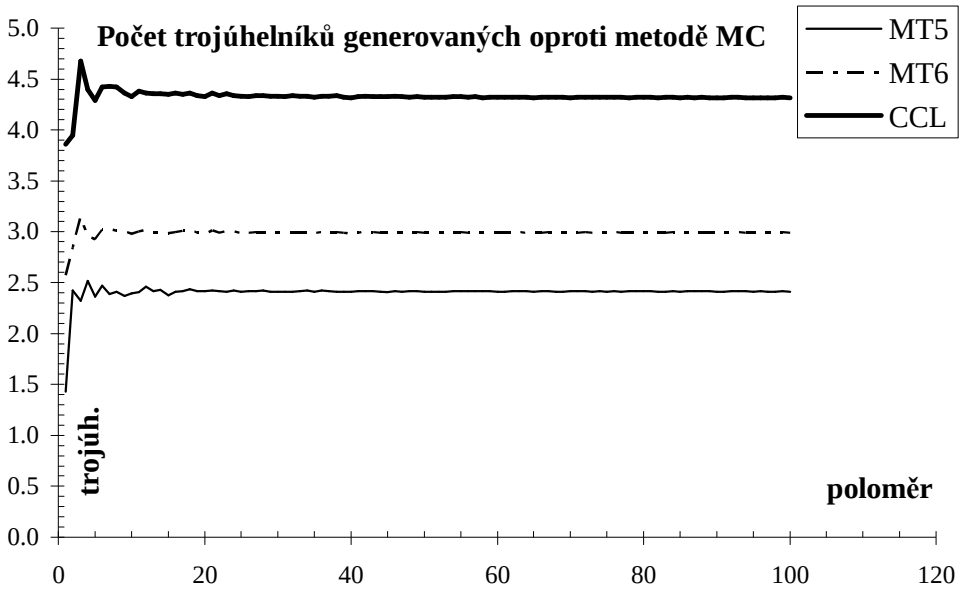


Graf 11.4.2.1: Počet generovaných trojúhelníků

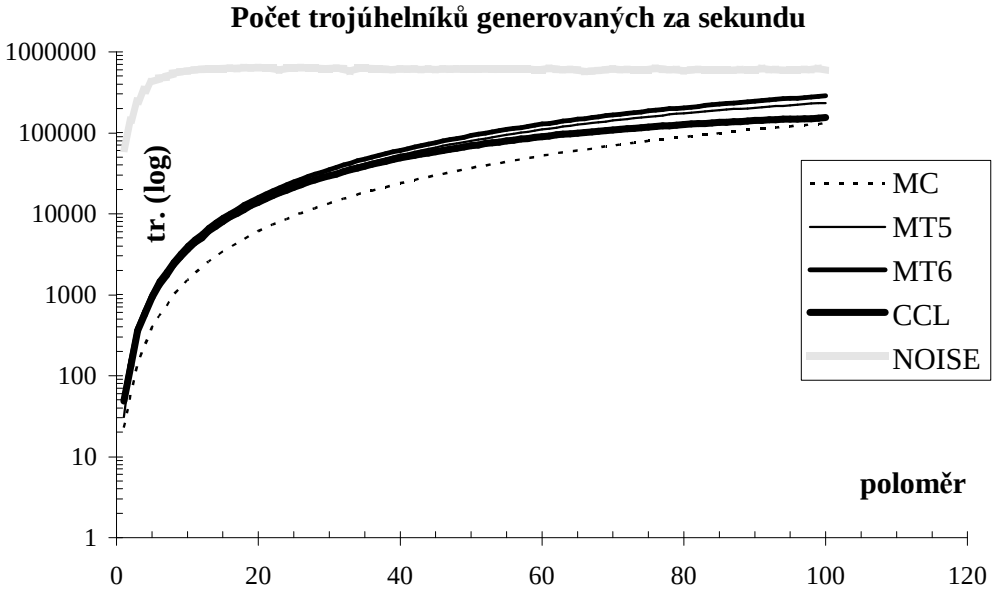
Počet generovaných trojúhelníků závisí především na způsobu jakým daná metoda dělí aktivní buňku. Metoda MC pracuje s buňkou jako s celkem a dále ji nedělí. Metoda MT5 dělí buňku na pět tetrahedronů, metoda MT6 na šest tetrahedronů a metoda CCL v průměru na 12 tetrahedronů.

U metody CCL je buňka dělena na 24 tetrahedronů, tetrahedrony však zahrnují i části okolních buněk. Pokud sečteme objem 24 tetrahedronů, dostaneme objem dvou buněk, tedy v průměru 12 tetrahedronů na jednu buňku. Z grafu 11.4.2.2 je patrný nárůst extrahovaných trojúhelníků, u metod dělících buňku na několik tetrahedronů, oproti metodě MC. Nárůst trojúhelníků je definován vztahy:

$$Trojúh\_MT5 = \frac{Trojúh_{MT5}}{Trojúh_{MC}}, \quad Trojúh\_MT6 = \frac{Trojúh_{MT6}}{Trojúh_{MC}}, \quad Trojúh\_CCL = \frac{Trojúh_{CCL}}{Trojúh_{MC}}$$



Graf 11.4.2.2: Počet trojúhelníků generovaných oproti metodě MC



Graf 11.4.2.3: Počet trojúhelníků generovaných za sekundu

Průměrný počet trojúhelníků generovaných za sekundu (graf 11.4.2.3) je dán poměrem celkového počtu vygenerovaných trojúhelníků a celkové doby extrakce iso-plochy. Metoda *NOISE* extrahuje nejvíce trojúhelníků za sekundu. Důvodem je to, že *NOISE* metoda prochází minimum neaktivních buněk (díky kd-stromu), tedy pouze buňky, ze kterých nejsou generovány trojúhelníky. Naproti tomu metoda CCL, která generuje

celkově nejvíce trojúhelníků jich generuje nejméně za sekundu. To může být dáno složitějším rozhodováním, zda je buňka aktivní či nikoliv. Je totiž nutné zkontrolovat všech 24 tetrahedronů, přičemž tetrahedrony zasahují i mimo buňku.

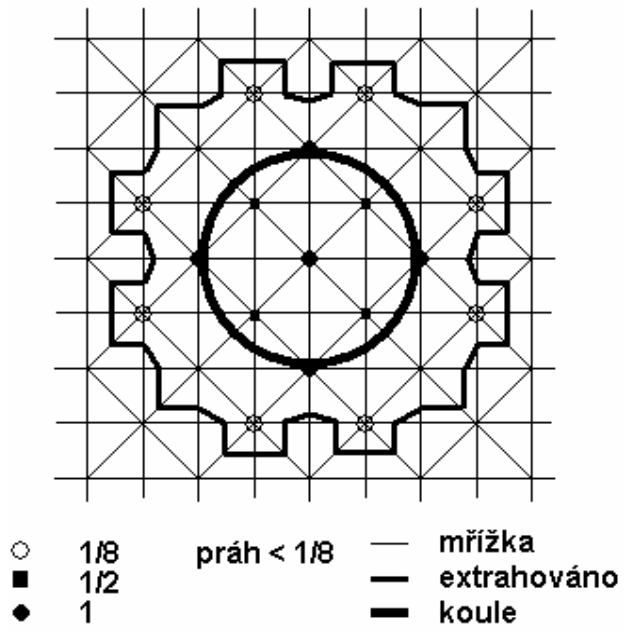
11.4.3 Chyba aproximace poloměru

Chyba je dána rozdílem mezi vzdáleností těžiště trojúhelníka od počátku koule a poloměrem koule, pro celou trojúhelníkovou síť pak:

$$Chyba = \sum_{K=1}^{N} \frac{d_K - r}{r}$$

$$d_K = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}$$

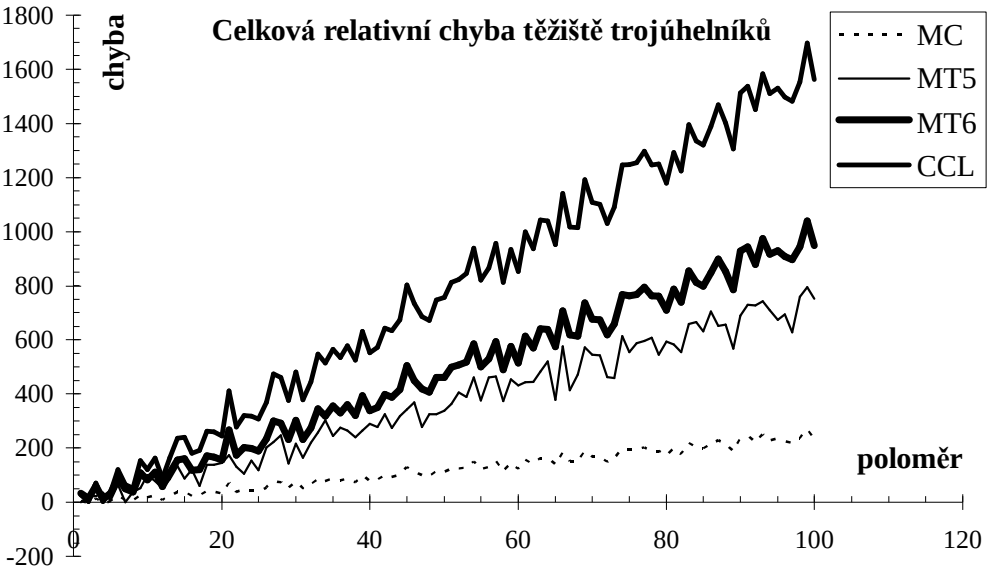
Kde d_K je vzdálenost těžiště trojúhelníka od středu koule, r je poloměr koule a N je celkový počet trojúhelníků aproximujících danou kouli. Dále x, y, z jsou souřadnice těžiště trojúhelníka a x_0, y_0, z_0 jsou souřadnice středu koule.



Obr. 11.4.3.2: 2D nákres pro kouli o r=1 a metodu CCL

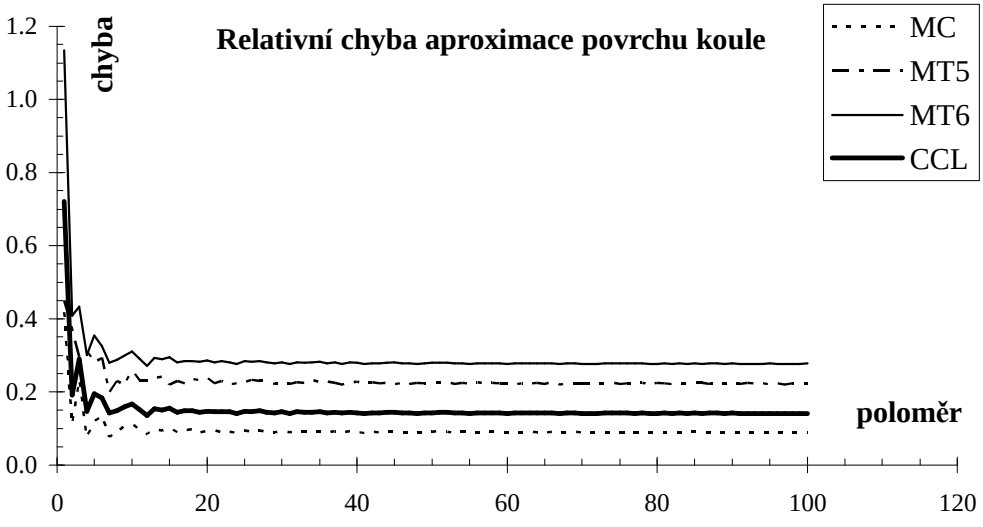
Chyby jsou u všech metod kladné, většina těžišť trojúhelníků je od středu koule dále než je poloměr koule. U metody MC se interpoluje pouze na hranách buňky (kvádříku). U metod MT5 a MT6 se interpoluje i na stěnových a tělesových úhlopříčkách, jejichž délka je větší než délka hrany, tím je dána i větší chyba. U CCL metody se

interpoluje na hranách, tělesových úhlopříčkách a navíc ještě mezi středy dvou sousedních kvádříků (přes stěnu). CCL odhaduje hodnoty ve středu buněk a navíc pro určité hodnoty prahu (menší než 1/8 maximální hodnoty 255) dochází k mírnému zvětšení koule, viz graf 11.4.3.2.



Graf 11.4.3.1: Celková relativní chyba těžiště trojúhelníků

11.4.4 Chyba aproximace povrchu



Graf 11.4.4.1: Relativní chyba aproximace povrchu koule

Chyba je dána rozdílem mezi skutečným povrchem generované koule a povrchem iso-plochy, která tuto kouli aproximuje (trojúhelníková síť), viz následující vztah:

$$Chyba = \frac{S_{TR} - S}{S}$$

Kde S_{TR} je povrch trojúhelníkové sítě a S je povrch koule.

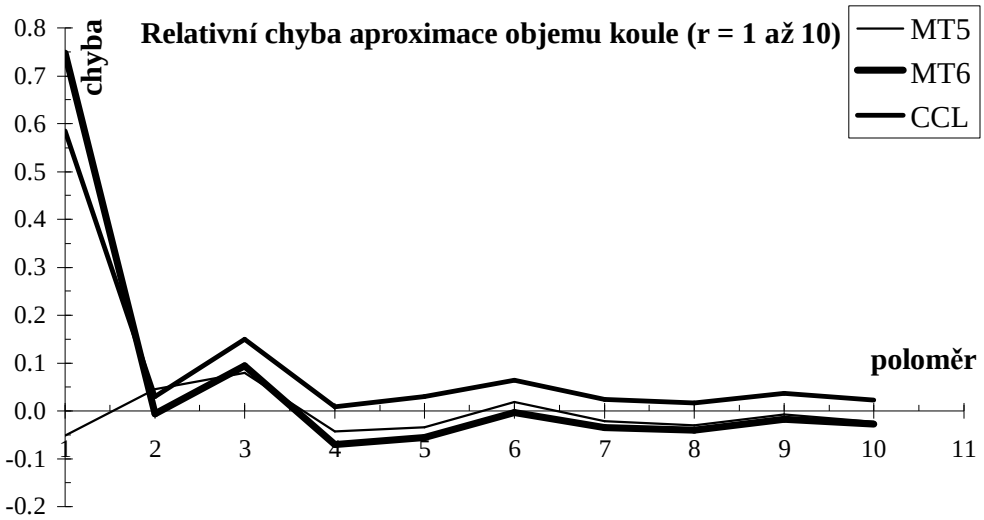
CCL metoda se zde oproti předchozí kapitole posunula na druhou nejlepší pozici. Důvodem je dělení buňky na 12 tetrahedronů s jedním vrcholem vždy ve středu buňky, tedy generovaná iso-plocha lépe přiléhá ke skutečnému povrchu koule. MT5 metoda má zase výhodu oproti MT6 metodě v tom, že střídá pravidelně dvě různá dělení buňky na pět tetrahedronů, výsledná chyba je tedy pro každé dělení jiná. MT6 metoda dává nejhorší výsledky z uvedených metod.

11.4.5 Chyba aproximace objemu

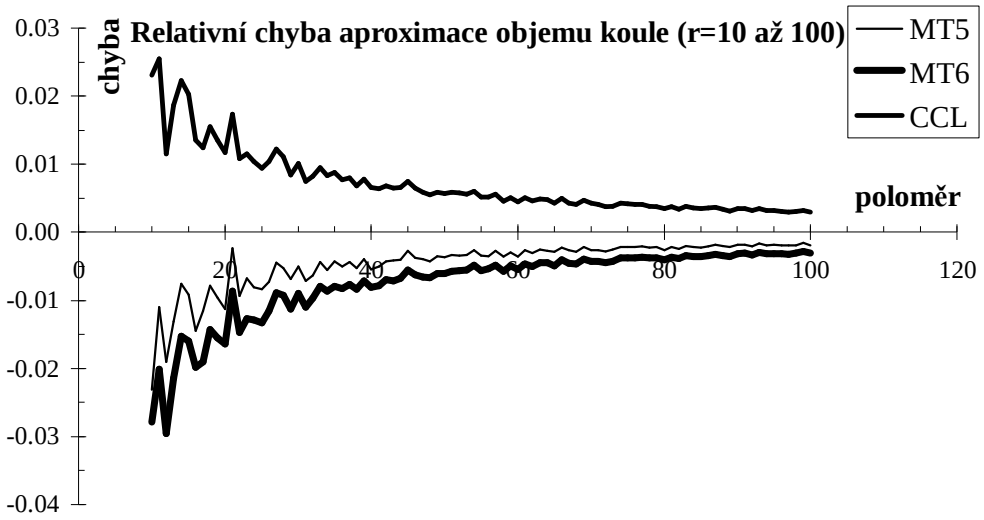
Chyba je dána rozdílem mezi objemem, který je uzavřen iso-plochou a skutečným objemem generované koule, viz vztah:

$$Chyba = \frac{V_{TR} - V}{V}$$

Kde V_{TR} je objem uzavřený trojúhelníkovou sítí a V je skutečný objem koule.



Graf 11.4.5.1: Relativní chyba aproximace objemu koule (r=1 až 10)

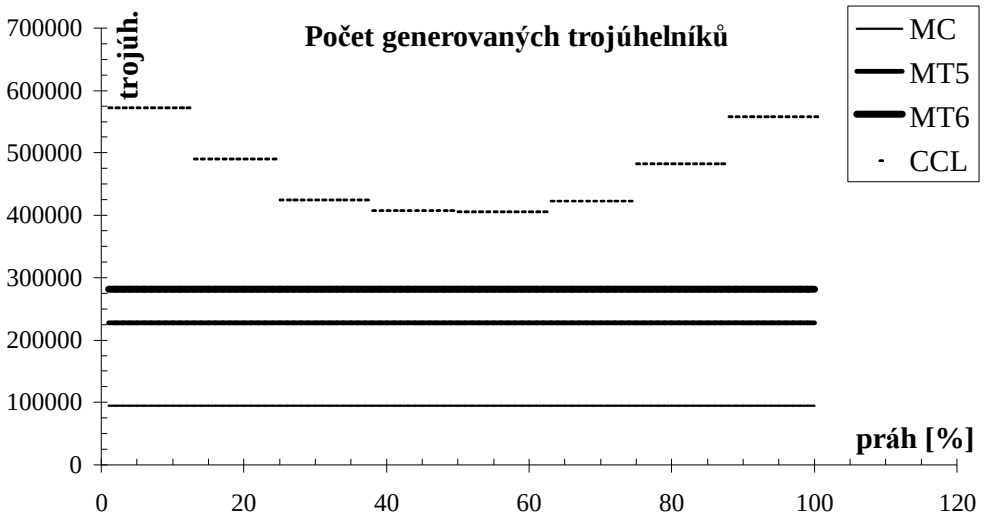


Graf 11.4.5.2: Relativní chyba aproximace objemu koule (r=10 až 100)

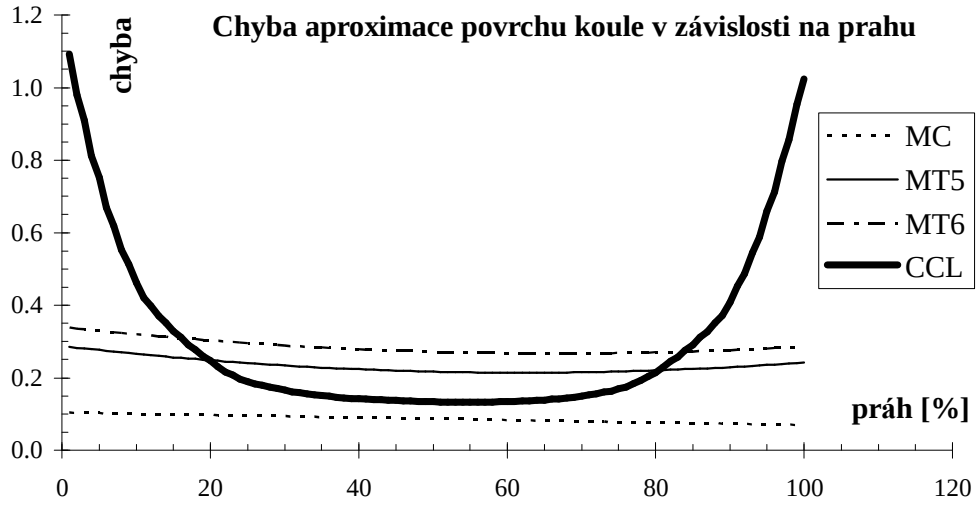
CCL metoda dělí buňku v průměru na 12 tetrahedronů. V určitých případech může dojít i ke zvětšení objemu koule. Dopočítáváním hodnoty volumetrických dat i ve středu buňky dochází k mírné expanzi volumetrických dat, to je důvod proč se CCL blíží ke skutečnému objemu shora. MT5 má opět oproti MT6 výhodu ve střídání dvou různých dělení, přičemž se obě metody blíží ke skutečnému objemu zdola.

11.4.6 Změna prahu

Měření byla provedena na kouli o poloměru 50 ve volumetrických datech o rozlišení 210. Počet extrahovaných trojúhelníků se pro námi generovanou kouli v závislosti na změně prahu nemění při použití metod MC, MT5, MT6. Pro CCL se mění skokově (viz graf 11.4.6.1). V CCL se dopočítává hodnota ve středu buňky, viz obr. 11.4.3.2. Změnou prahu se tedy v tomto případě mění i počet tetrahedronů, ze kterých jsou extrahovány trojúhelníky. Pro buňky ve tvaru kvádříku je zde osm intervalů pro prahovou hodnotu, jelikož se středová hodnota počítá pomocí aritmetického průměru z osmi okolních vrcholů buňky. Jeden až osm vrcholů může mít hodnotu 255 (v našem případě), pak procentuální meze intervalů, kde dochází ke změně počtu extrahovaných trojúhelníků, pro prahovou hodnotu jsou: 0, 12.5, 25, 37.5, 50, 62.5, 75, 87.5 a 100, jak je patrné z grafu 11.4.6.1.

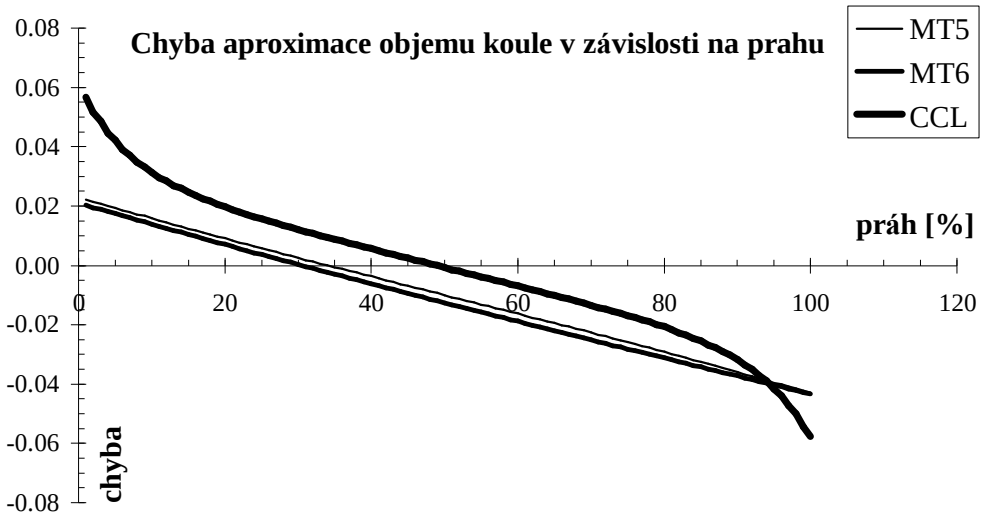


Graf 11.4.6.1: Počet generovaných trojúhelníků



Graf 11.4.6.2: Chyba aproximace povrchu koule v závislosti na prahu

Z grafu 11.4.6.2 je patrná závislost chyby aproximace povrchu koule na změně prahové hodnoty (v procentech). Prahová hodnota je minimální pro vrcholy buněk, které leží mimo kouli a maximální po vrcholy buněk uvnitř a na povrchu koule. Prahová hodnota je zadána v procentech na intervalu <*min; max*>. Pro práh 0% iso-plocha přiléhá k minimálním hodnotám protnutých buněk a naopak.



Graf 11.4.6.3: Chyba aproximace objemu koule v závislosti na prahu

12 Závěr

Byly popsány různé metody pro extrakce iso-ploch z volumetrických dat. V kapitole 6 jsou popsány základní způsoby urychlení extrakce iso-ploch. Dále bylo provedeno porovnání těchto metod vzhledem k různým kritériím (povrch, objem, atd.).

Popisované algoritmy byly realizovány ve formě DLL knihovny a začleněny do MVE systému. Implementace byla provedena v prostředí MS Visual C++ 6.0. Program a všechny zdrojové kódy jsou psány v anglickém jazyce. Spolupráce vytvořených modulů s ostatními moduly zahrnutými v MVE byla testována na modulech IsoExtractor, VolumeLoader, Renderer. Činnost modulů byla dále testována na reálných volumetrických datech (soubory: `bentum.vol`, `syn_64.vol`, `cthead.vol`, `ctmayo.vol`, `engine.vol`, `hplogo.vol`).

Z hlediska doby potřebné pro extrakci iso-plochy je nejlepším řešením použít metodu *NOISE* a sestrojit k daným volumetrickým datům kd-strom v rámci předzpracování. A po té již pouze extrahovat iso-plochy pro různé prahové hodnoty. Vytvoření kd-stromu je celkem časově náročné, ale stačí ho pro daná data sestrojit pouze jednou a po té ho uložit např. do souboru na pevný disk.

Pokud bude kritériem volby metody počet generovaných trojúhelníků, pak je jednoznačně nejvýhodnější použít metodu *Marching Cubes,* která z výše uvedených metod

obecně generuje nejméně trojúhelníků. Výstupem metody je trojúhelníková síť, která navíc nejlépe aproximuje povrch objektu.

Chceme-li naopak co nejpřesněji aproximovat objem objektu je nejvýhodnější použít metodu *Marching Tetrahedra* s dělením na pět tetrahedronů i za cenu nárůstu počtu výstupních trojúhelníků a doby extrakce iso-plochy.

13 Použitá literatura (v abecedním pořadí)

[Bourke97] Bourke,P.: *Polygonising a Scalar Field Using Tetrahedrons*, 1997, http://astronomy.swin.edu.au/pbourke/modelling/polytetr

[Cignoni97] Cignoni,P., Marino,P., Montani,C., Puppo,E., Scopigno,R.: *Speeding Up Isosurface Extraction Using Interval Trees*, 1997, http://www.citeseer.com

[Krejza00] Krejza,M. (vedoucí práce Skala,V.): *Metody zobrazení iso-ploch a metody extrakce iso-ploch z volumetrických dat*, diplomová práce, Západočeská Univerzita - Fakulta aplikovaných věd, 2000

[Livnat99] Livnat,Y., Parker,S.,G., Johnson,Ch.,R.: *Fast Isosurface Extraction Methods for Large Imaging Data Sets*, Center for Scientific Computing and Imaging, Dept. of Comp. Science, University of Utah, Salt Lake City, USA, UT 84112, http://www.cs.utah.edu/~sci, 1999

[Rektorys95] Rektorys,K.: Přehled užité matematiky I a II, 6. přepracované vydání, ISBN 80-85849-92-5, Prometheus, Praha,1995

[Schroeder98] Schroeder,W., Martin,K., Lorensen,B.: *The Visualization Toolkit*, 2nd Edition, Prentice Hall PTR, 1998, ISBN 0-13-954694-4

[Terrades99] Terrades,A.,B. (supervisor Skala,V.): *Comparsion of Tetrahedron Subdivision Schemes of Volumetric Data*, Diploma Thesis, University of West Bohemia – Faculty of Applied Sciences, 1999

[Wirth76] Wirth,N.: Algorithms + data structures = programs, Englewood Cliffs (Prentice-Hall), 1976

[Weisstein99] Weisstein,E.,W.: *World of Mathematics*, CRC Press LLC, Wolfram Research Inc., 1999, Internet: http://mathworld.wolfram.com

[WWW1] Internet: http://beety.kgb.cz/skola/dp/iso.html

[WWW2] Internet: http:// http://mathworld.wolfram.com/Torus.html

[WWW3] http://herakles.zcu.cz/research/mve/progrdoc/index.htm

[Zara98] Žára,J., Beneš,B., Felkel,P.: *Moderní počítačová grafika*, Computer Press, ISBN 80-7226-049-9, 1998

Příloha A

(Appendix A)

MVE Modules User Guide

These appendixes should be used as a user guide for MVE modules which were created as a part of this diploma thesis. On figure A.1 you can see a scheme which can be created with these modules in MVE system. Whole MVE system and other modules are described in Research section at website: `http://herakles.zcu.cz`.

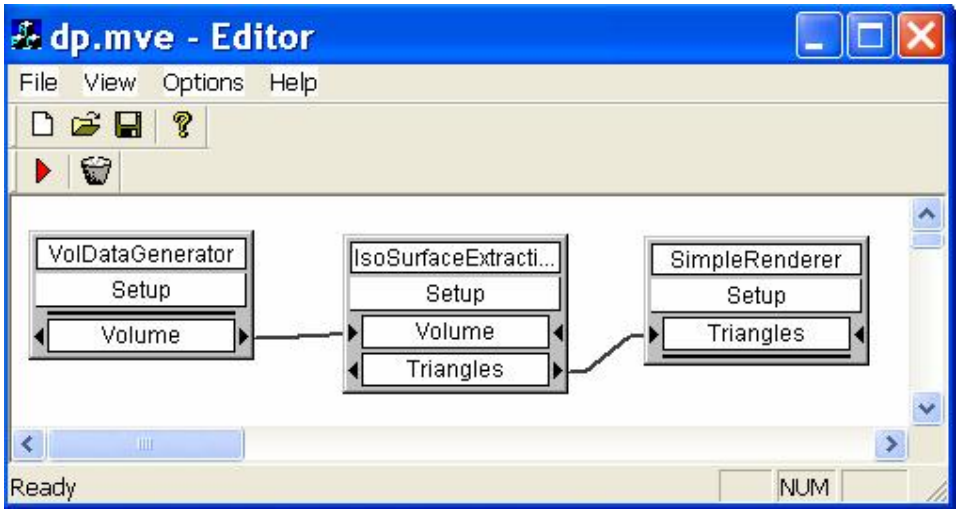


Figure A.1: MVE and user modules scheme for iso-surface extraction

Module VolDataGenerator - Introduction

This module provides generating of structured volumetric data (cells with a cube shape), as well as loading of volumetric data from files. The input file has to have an extension `*.vol` (volumetric data file) and in the same directory there has to be stored corresponding `*.vif` text file (description of volumetric data in `*.vol` data file). This module is able to load a corresponding kd-tree `*.kdt` to `*.vol` data file from the same directory. Kd-tree is used in *NOISE* isosurface extraction method, see IsoSurfaceExtraction module description bellow. For use of kd-tree you have to use VolDataGenerator and IsoSurfaceExtraction modules together. Module does not have any input. As output it has Volume MVE data structure for volumetric data.

Module VolDataGenerator – Setup

- **Resolution** – here you can select resolution of generated regular volumetric data, implicit value is *128*. This value is taken for all three directions along axis (x, y, z).
- **Chosen Object** – Here you can choose an object you want to generate, there are these choices: Sphere (R), Torus (C, A). In brackets behind an object there are parameters which you can use to modify shape of an object. To modify these parameters you can use edit boxes under Chosen Object (see figure A.2).
- **R, C and A edit fields** – you can use these fields as a parameters for generator of a chosen object. R stands for radius, C is a main radius of torus (ring radius) and A is a secondary radius for torus (ring thickness).
- **Load** – to load volumetric data from the file tick this checkbox and select a filename to load by Choose Input File Name button. Remember that in the same directory there has to be a file with the same name as selected file and extension `*.vif` instead of `*.vol`.
- **Load Kd-tree as well** – tick this checkbox and corresponding kd-tree will be loaded into memory. Kd-tree file has to have the same name and path as data file (previously selected `*.vol` file) and extension `*.kdt` instead of `*.vol`.

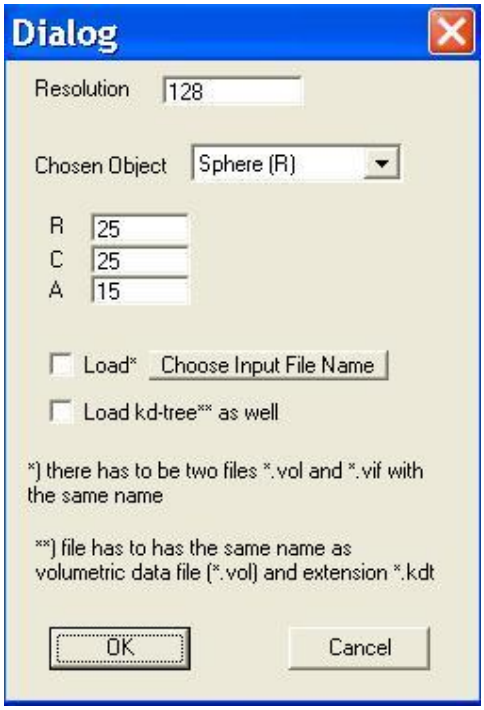


Figure A.2: Setup Dialog for VolDataGenerator module

Now click the OK button to accept these settings or click the Cancel button (or a dialog close system cross) to refuse changes. The object is generated with values 255 and the rest of volumetric data has the value 0.

Module IsoSurfaceExtraction – Introduction

This module extracts isosurface from regular volumetric data. Module provides few different methods for isosurface extraction, see next chapter for more details. It allows save output triangle mesh to a specified file on your hard drive. For NOISE method it provides a possibility to save resulting kd-tree data structure to a file. Some information about generated mesh can be saved to a log file (for use of log file facility you are expected to use VolDataGenerator and IsoSurfaceExtraction modules together).

Module IsoSurfaceExtraction – Setup

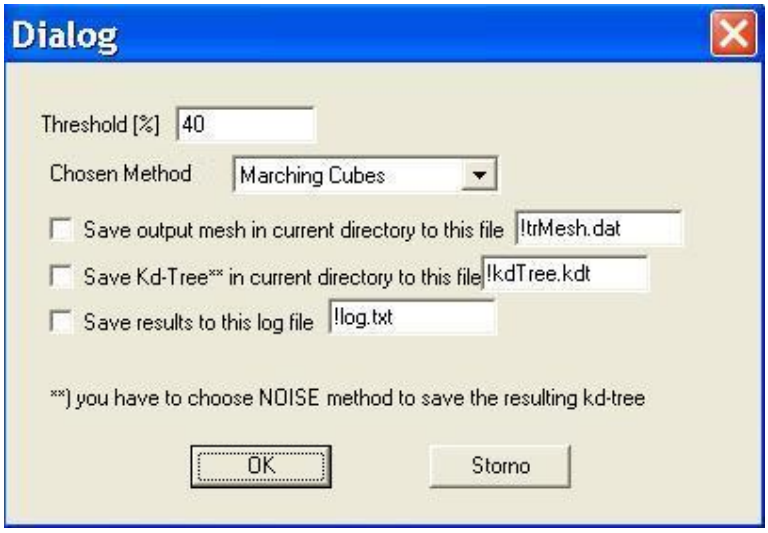


Figure A.3: Setup Dialog for IsoSurfaceExtraction module

- **Threshold** – here you can select a threshold for isosurface you want to extract. The value is in percents between minimal and maximal value that can be found in volumetric data.

- **Chosen Method** – here you can choose method for isosurface extraction. Your choices are: Marching Cubes, Marching Tetrahedra 5, Marching Tetrahedra 6, Centered Cubic Lattice and *NOISE*.
- **Save output mesh** – tick this checkbox to save resulting triangle mesh to the current directory, you can choose the file name as well.
- **Save Kd-tree** – to save the resulting kd-tree in current directory you have to select *NOISE* isosurface extraction method at first. If you wish module to save this data structure, tick this checkbox. Expected file name extension is `*.kdt`.
- **Save results** – some additional information about extracted triangle mesh (number of generated triangles, area of whole mesh, etc.) can be saved into the log file, if you tick this checkbox.

Now click the OK button to accept these settings or click the Cancel button (or a dialog close system cross) to refuse changes. The object is generated with values 255 and the rest of volumetric data has the value 0. Remember that all files are stored in current directory (usually in MVE folder or in the same directory as the chosen `*.vol` data file is). Input has to be Volume MVE data structure. Output is then Triangles MVE data structure.

Module SimpleRenderer – Introduction

This module is able to render the Triangle MVE data structure. To use extra features of this renderer click right mouse button to unhide the context menu for renderer dialog or you can use these shortcuts:

- gray +, gray – (numeric keypad) for zooming the object
- left mouse button double click to switch autorotation feature on/off

The renderer module provides these functions:

- **Surface Model** – displays rendered surface model of the input triangle mesh
- **Wired Model** – displays wired surface model of the input triangle mesh (lines)
- **Zoom In** – zooms the displayed object in
- **Zoom Out** – zooms the displayed object out
- **Points Model** – displays points model of the input triangle mesh (vertices)
- **Show/Hide Axis** – shows or hides the x, y and z axis

- **Start/Stop Autorotate** – starts or stops the auto rotation of the object
- **Default** – restores the implicit renderer settings
- **Inverse Normal Vectors** – changes the direction of normal vectors for triangle mesh which are used to render the object
- **Show/Hide Sphere** – for use of this menu item it is expected that you use this module with IsoSurfaceExtraction module and VolDataGenerator module must be set to generate a sphere.

Renderer Outputs

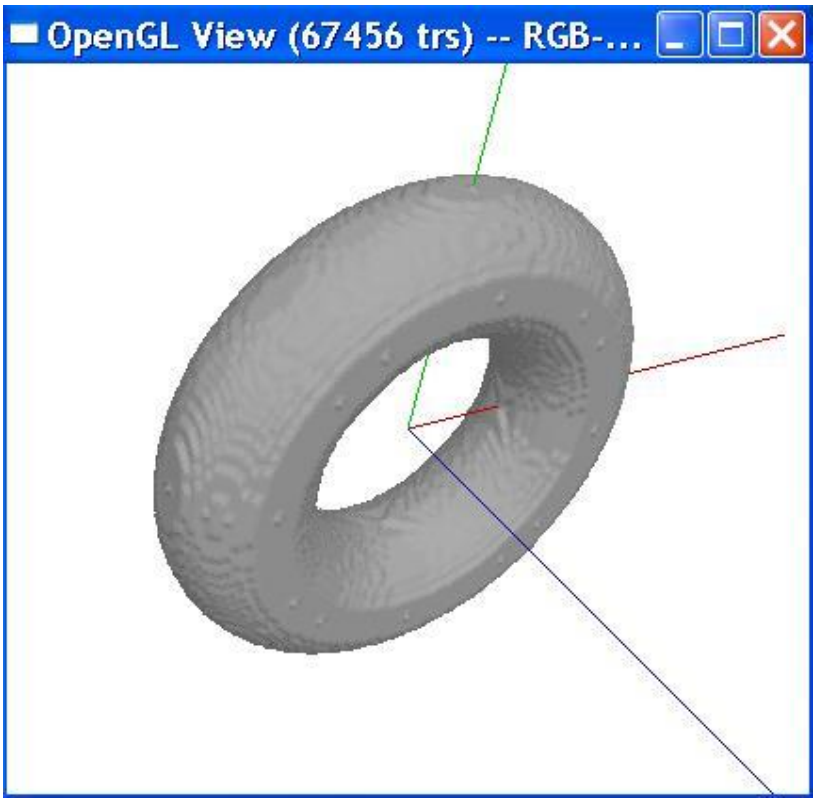


Figure A.4: Example of SimpleRenderer module output (the axis x, y and z are drawn in these colors: x = red, y = green and z = blue)

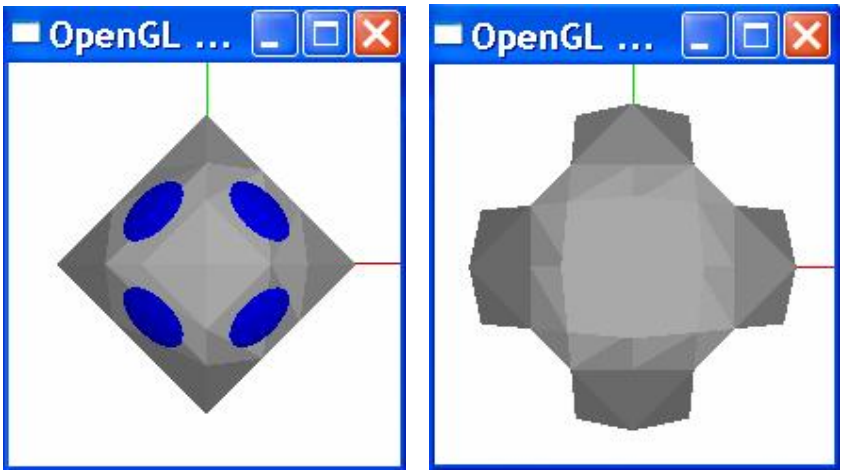


Figure A.5: Example of SimpleRenderer module output: black=gluSphere(), gray=triangle mesh approximation (left — MC sphere with radius=1, right—MT5_1 sphere with radius=1)

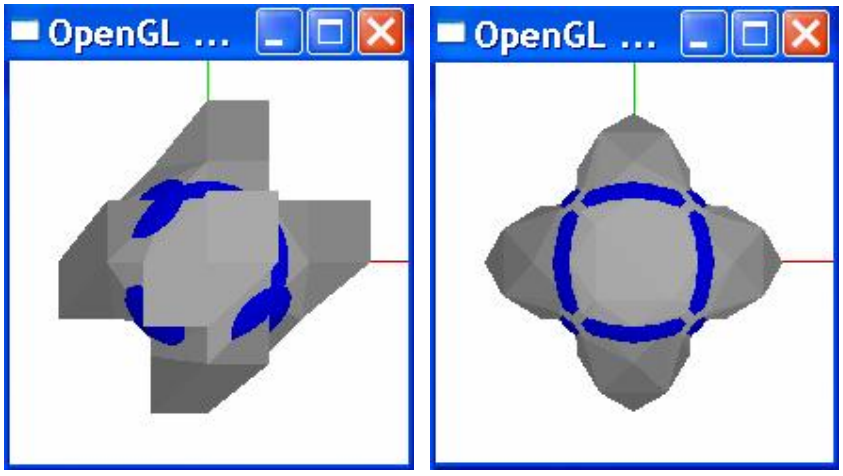


Figure A.6: Example of SimpleRenderer module output: black=gluSphere(), gray=triangle mesh approximation (left — MT6 sphere with radius=1, right—CCL sphere with radius=1)

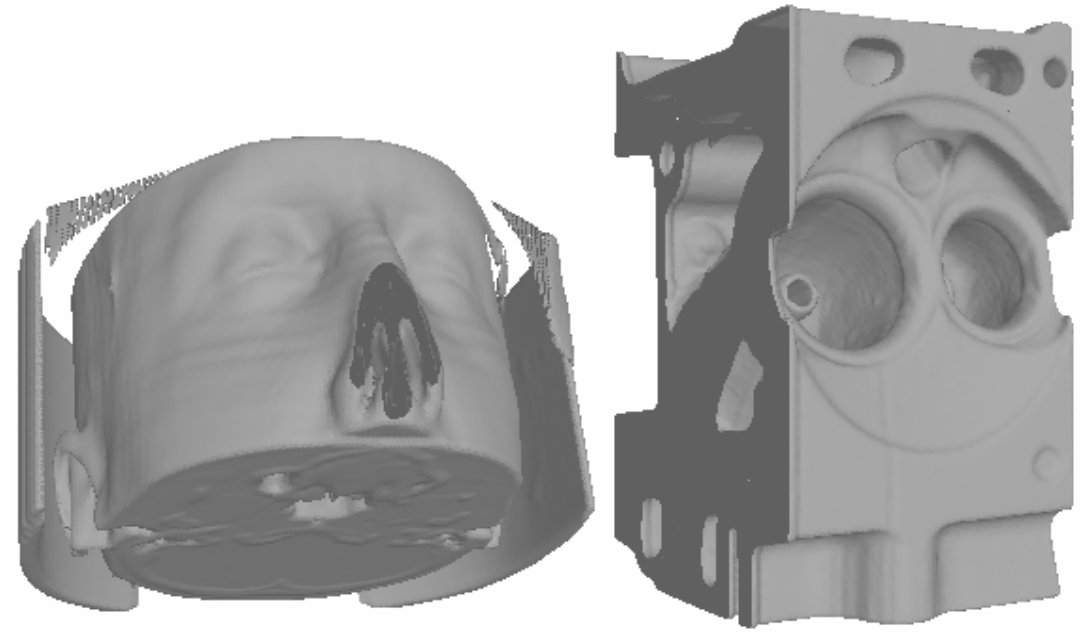


Figure A.7: Example of SimpleRenderer module output (left — MC ctmayo.vol, right—MC engine.vol)

Figure A.8: Example of SimpleRenderer module output (MC cthead.vol)

VTK Outputs Examples

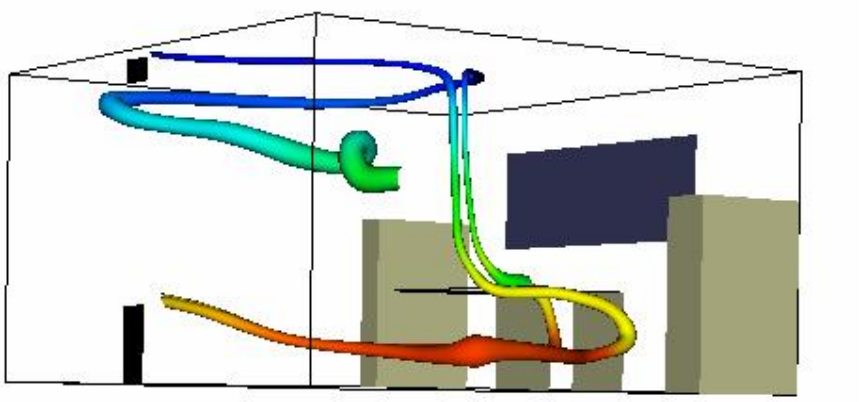


Figure A.9: Example of vtkStreamLine visualization (office.vtk)

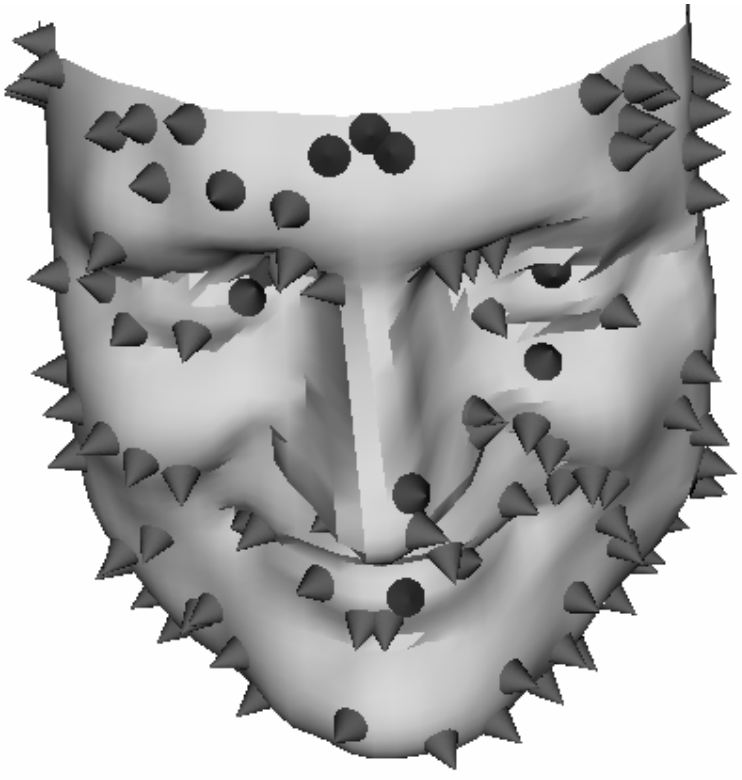


Figure A.10: Example of vtkGlyph3D visualization (fran_cut.vtk)

Příloha B
(Appendix B)

Installation Guide

All described modules are stored in `\Public\isosurf_extraction.dll` library on enclosed CD-ROM. Just copy this file into MVE installation directory `MVE\MODULES` where other modules are installed. For more information look at MVE complete documentation on: `http://herakles.zcu.cz`, in section Research. The SimpleRenderer module requires the OpenGL® drivers to be installed on the destination computer.

Programmer's Guide

All modules were programmed in Microsoft Visual C++ 6.0 (MSVC++) with use of MFC (Microsoft Foundation Classes) library. Whole project consists of the following files. Source codes can be found on enclosed CD-ROM in `\Private\SourceCode` directory.

Standard MSVC++ Files

Standard MSVC++ files which are generated automatically when the project for DLL library is established under MSVC++.

- `isosurf_extraction.dsp` – This file (the project file) contains information at the project level and is used to build a single project or subproject. Other users can share the project (`*.dsp`) file, but they should export the makefiles locally.
- `isosurf_extraction.cpp` – This is the main DLL source file that contains the definition of `DllMain()` function and the definitions of exported functions as well
- `isosurf_extraction.rc` – This is a listing of all of the MS Windows resources that the program uses. It includes the menus, icons, bitmaps, buttons, etc. This file can be directly edited in MSVC++.
- `isosurf_extraction.clw` – This file contains information used by Class Wizard to edit existing classes or add new classes. Class Wizard also uses this file

to store information needed to create and edit message maps and dialog data maps and to create prototype member functions.

- `res\isosurf_extraction.rc2` – This file contains resources that are not edited by MSVC++. You should place all resources not editable by the resource editor in this file.
- `isosurf_extraction.def` – This file contains information about the DLL that must be provided to run with MS Windows. It defines parameters such as the name and description of the DLL. It also exports functions from the DLL.
- `StdAfx.h`, `StdAfx.cpp` – These files are used to build a precompiled header (PCH) file named `isosurf_extraction.pch` and a precompiled types file named `StdAfx.obj`
- `Resource.h` – This is the standard header file, which defines new resource IDs. MSVC++ reads and updates this file

User Created and Added Files

- `marcube.c`, `marcube.h`, `marcube_tab.h` – These files implement *Marching Cubes* method for isosurface extraction. All files are programmed in ANSI C; commentary is created by // and is not ANSI. This function can be found in `marcube.c` source file, we are talking about `_expand()` function, which is able to expand or shrink a block of memory without reallocation.
- `mt5.c`, `mt5.h`, `mt5_macros.c`, `mt5_macros.h`, `mt5_tabs.h` – These files implement *Marching Tetrahedra 5* method for isosurface extraction. All files are programmed in ANSI C; commentary is created by // and is not ANSI. This function can be found in `mt5.c` source file, we are talking about `_expand()` function, which is able to expand or shrink a block of memory without reallocation.
- `mt6.c`, `mt6.h`, `mt6_macros.c`, `mt6_macros.h`, `mt6_tabs.h` – These files implement Marching Tetrahedra 6 method for isosurface extraction. All files are programmed in ANSI C; commentary is created by // and is not ANSI. This function can be found in `mt6.c` source file, we are talking about `_expand()` function, which is able to expand or shrink a block of memory without reallocation.
- `ccl.c`, `ccl.h`, `ccl_macros.c`, `ccl_macros.h`, `ccl_tab.h` – These files implement *Centered Cubic Lattice* method for isosurface extraction. All files are

programmed in ANSI C; commentary is created by // and is not ANSI. This function can be found in ccl.c source file, we are talking about `_expand()` function, which is able to expand or shrink a block of memory without reallocation.

- `noise.c`, `noise.h`, `noise_macros.c`, `noise_macros.h`, `noise_tab.h` – These files implement NOISE method for iso-surface extraction. All files are programmed in ANSI C; commentary is created by // and is not ANSI. This function can be found in `noise.c` source file, we are talking about `_expand()` function, which is able to expand or shrink a block of memory without reallocation.
- `GenSetupDlg.cpp`, `GenSetupDlg.h` – These two dialog source files define functionality of VolDataGenerator module setup dialog
- `OPGLView.cpp`, `OPGLView.h` – These two files define the functionality of SimpleRenderer module
- `SetupDlg.cpp`, `SetupDlg.h` – These two dialog source files define the functionality of IsoSurfaceExtraction module setup dialog
- `vol_gener.c`, `vol_gener.h` – These two source files define functionality of VolDataGenerator module.
- `comput.c`, `comput.h` – These files contain functions for computation of errors in approximation, area of the objects, volume of the objects, etc.
- `ma_globals.h` – this header file contains all global data types and variables

MVE Header Files

- `mve_include.h` – contains basic data structure definitions for module creation and setup
- `triangle_types.h` – contains definition of triangle mesh data structure
- `types.h` – contains basic data type definitions (pointers, signed vs. unsigned data types, floating numbers data types, etc.)
- `volume_types.h` – contains definition of data structure for volumetric data

Příloha C

(Appendix C)

Main Application

This executable file which was created as a part of this diploma thesis can be found on enclosed CD-ROM in `\Public\Bin` directory.

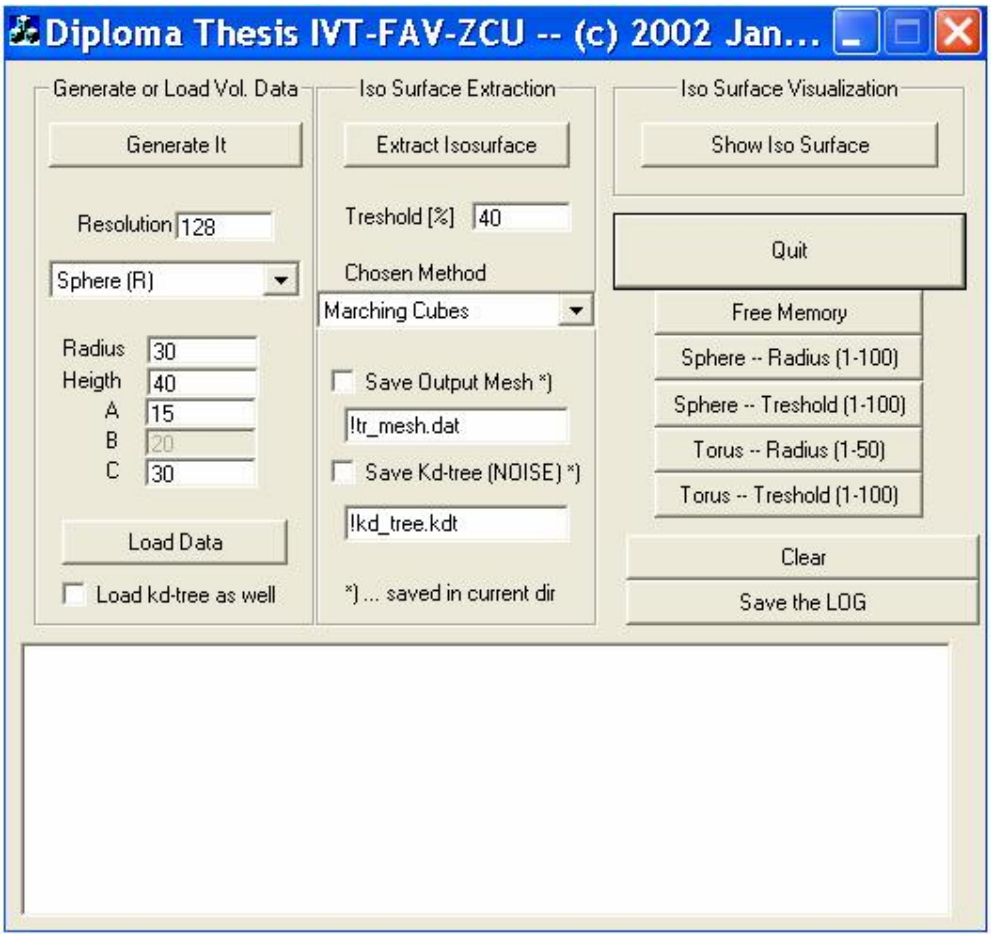


Figure C.1: Main Dialog Window

There are four main blocks as described bellow:

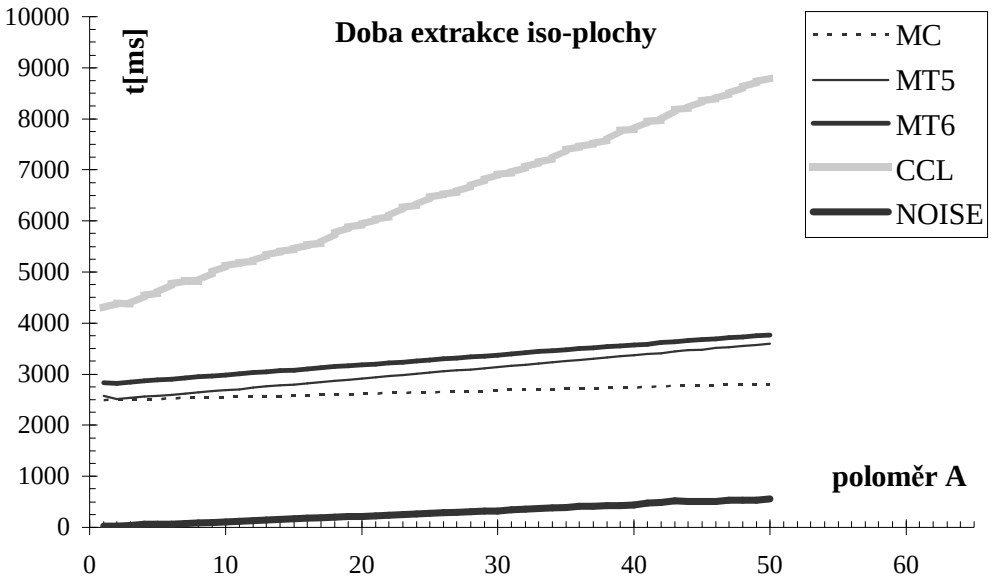
- **Generate or Load Vol. Data** – In this block you can select an object you want to generate or you can load vol. data from a file. Resolution – structured data resolution (the same in all three directions *x*, *y* and *z*); Popup (Sphere(R)) – here

you can choose an object you want to generate; Radius, Height, A, B and C edit boxes – appropriate parameters for previously chosen object (stated in the brackets in popup); Load Data – loads input *.vol data file; Load Kd-tree as well tick box – tick this box for load *.kdt file as well (file has to be placed in the same directory as chosen *.vol file). Press Generate It button to generate an object.

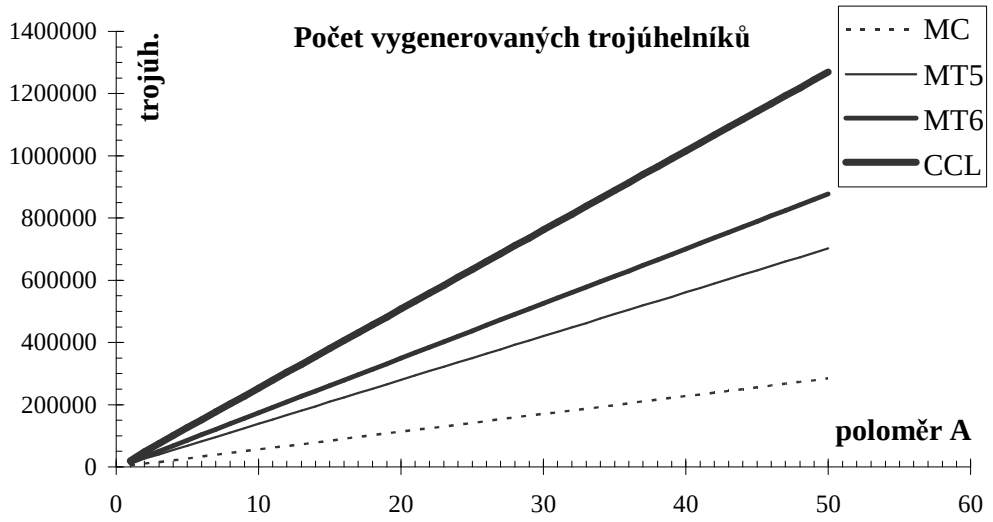
- **Iso Surface Extraction** – Threshold – select required threshold in percents between minimal and maximal data value for iso-surface extraction; Chosen Method popup – choose method for iso-surface extraction; Save output mesh tick box – saves output mesh in current directory to selected file name; Save kd-tree tick box – saves output kd-tree (NOISE method only) in current directory to selected file name. Press Iso Surface Extraction button to extract iso-surface.
- **Iso Surface Visualization** – Press this button for view of resulting triangle mesh.
- **Rest Buttons** – Free Memory – frees all allocated memory blocks; Sphere Radius (1-100) – generates the spheres and measures the methods parameters for radius changes; Sphere Threshold (1-100) – generates the spheres and measures the methods parameters for threshold changes; Torus Radius (1-50) – generates the toruses and measures the methods parameters for secondary radius changes; Torus Threshold (1-100) – generates the toruses and measures the methods parameters for threshold changes; Clear – clears the log box; Save the Log – saves the log in current dir to file !log.txt.

Příloha D

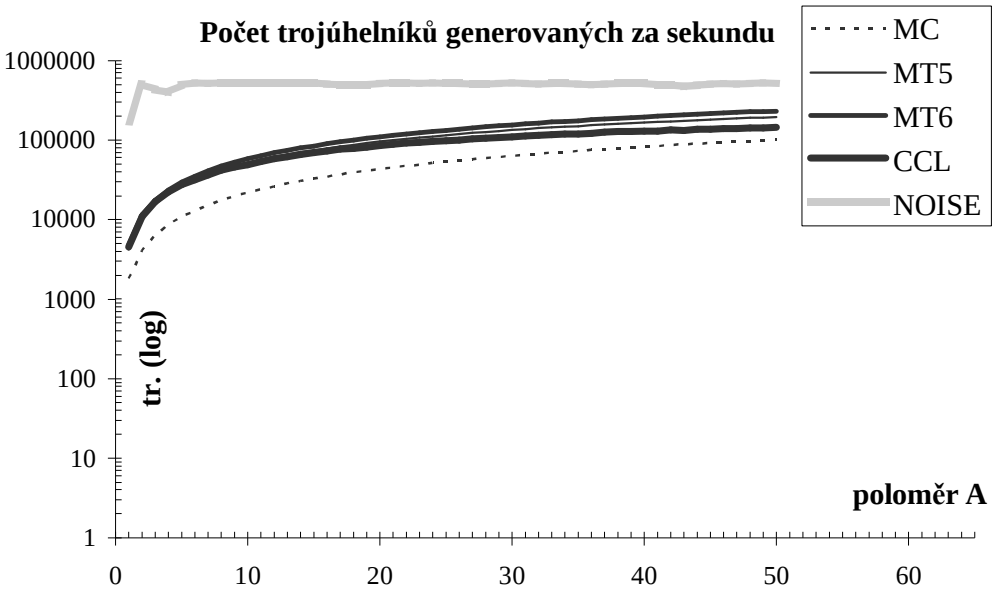
Grafy – Torus



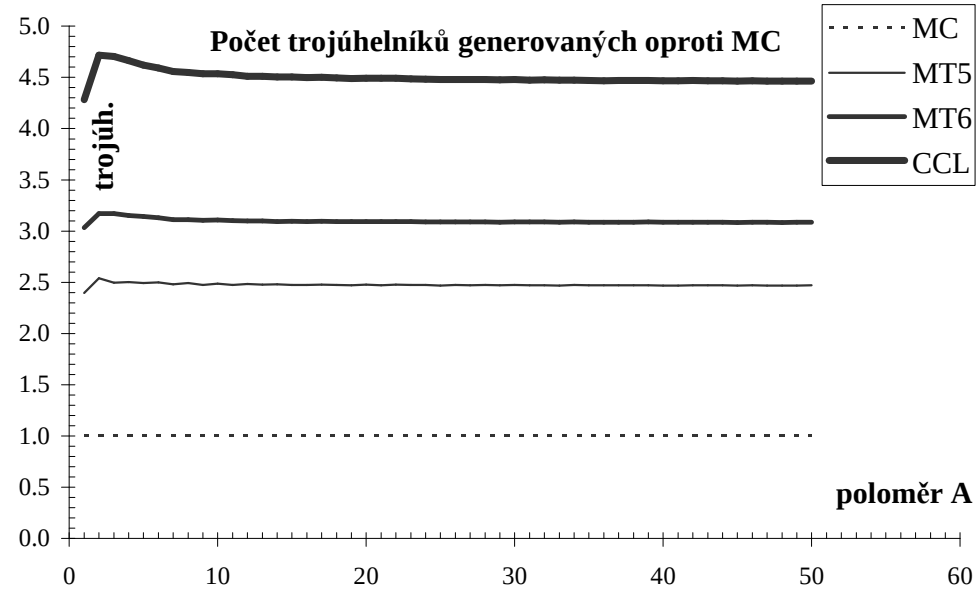
Graf. D.1: Doba extrakce iso-plochy



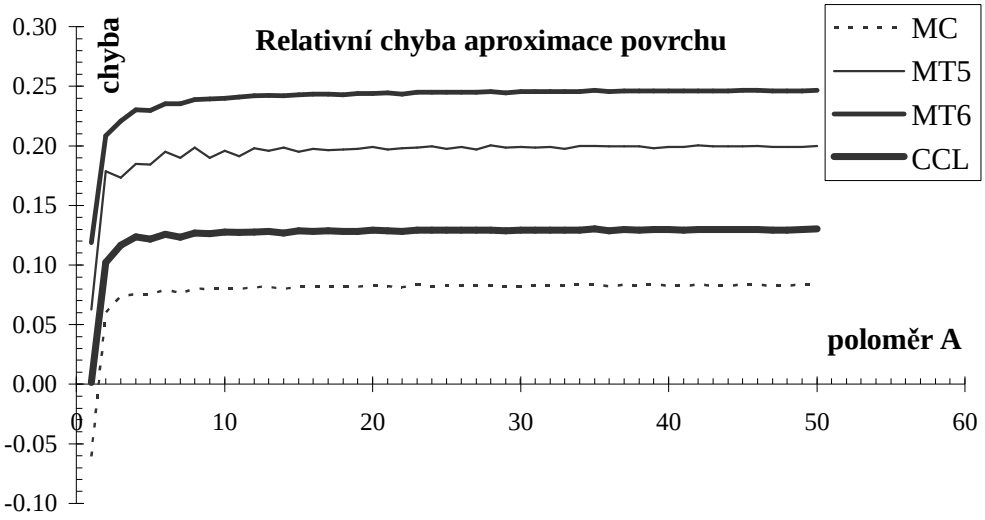
Graf. D.2: Počet vygenerovaných trojúhelníků



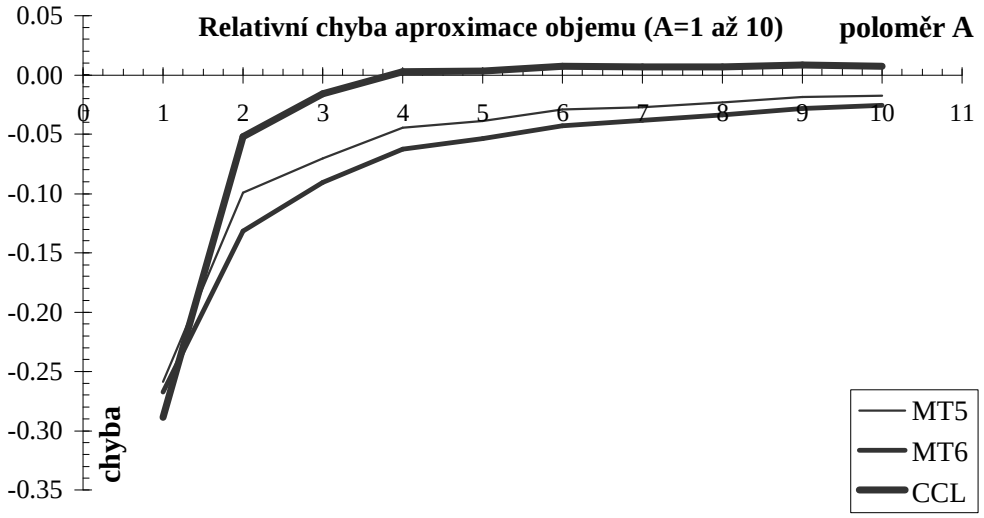
Graf. D.3: Počet trojúhelníků generovaných za sekundu



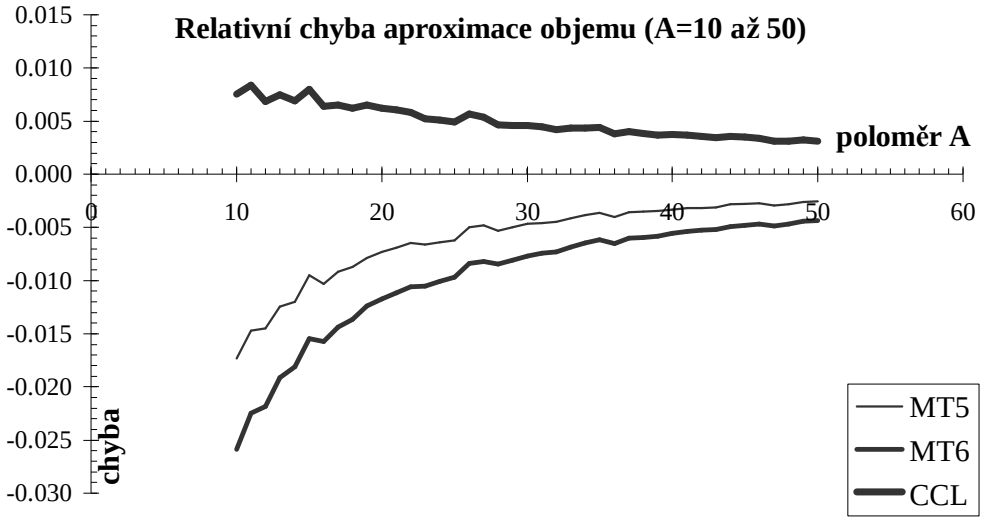
Graf. D.4: Počet trojúhelníků generovaných oproti MC



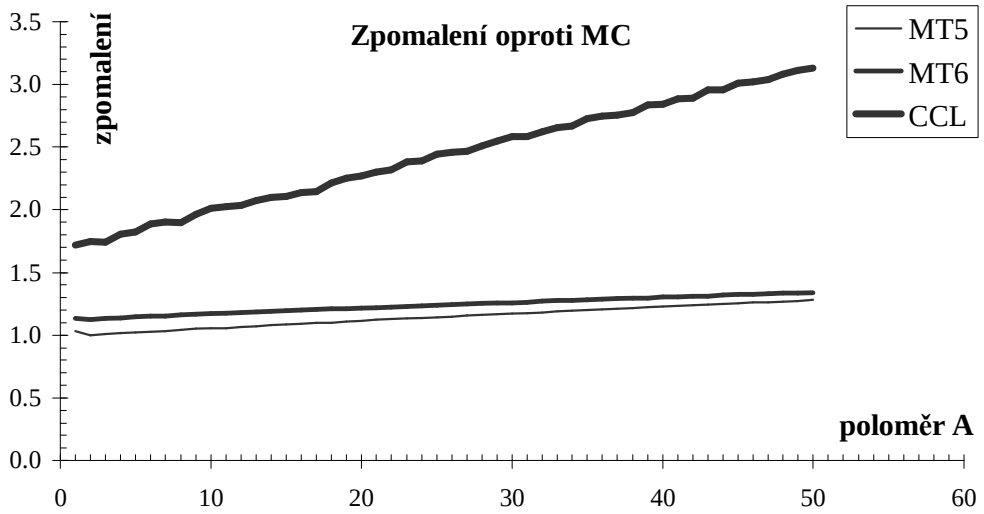
Graf. D.5: Relativní chyba aproximace povrchu



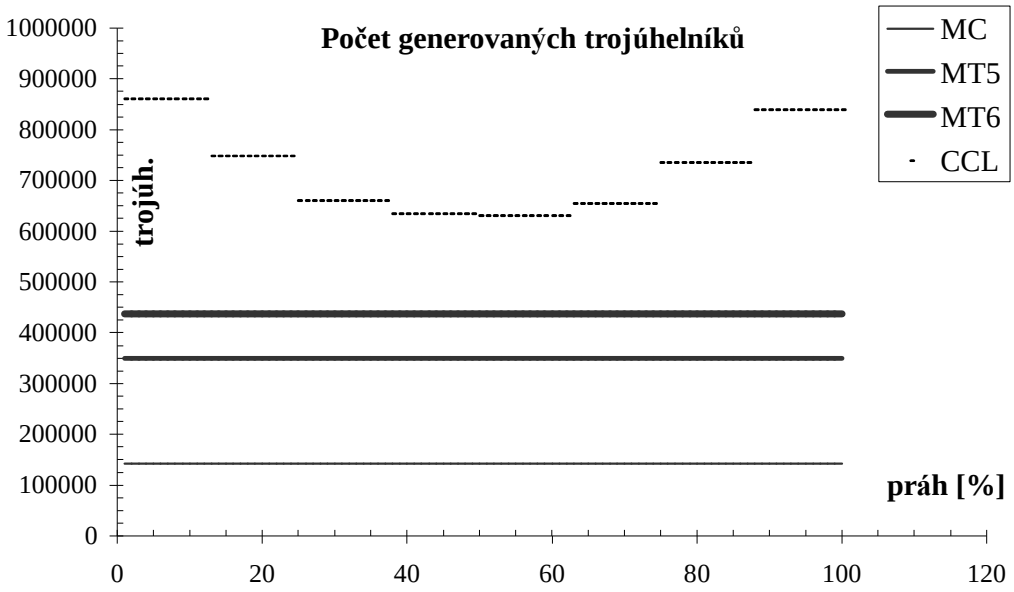
Graf. D.7: Relativní chyba aproximace objemu (r=1 až 10)



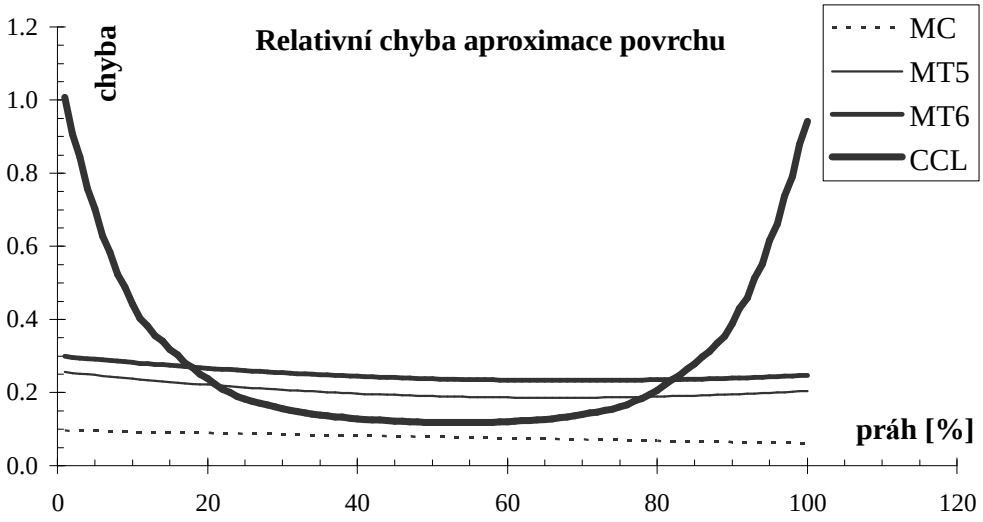
Graf. D.6: Relativní chyba aproximace objemu (r=10 až 50)



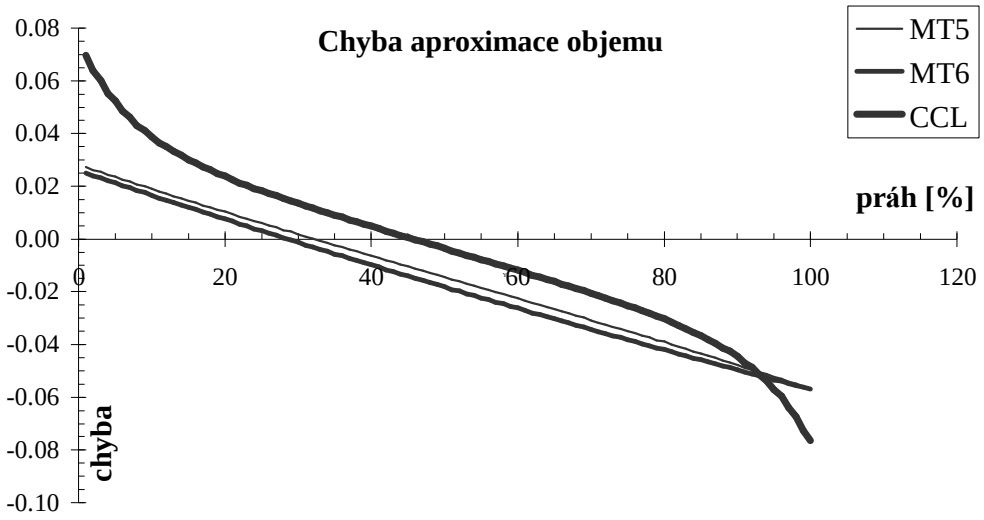
Graf. D.8: Zpomalení oproti MC



Graf. D.9: Počet generovaných trojúhelníků



Graf. D.10: Relativní chyba aproximace povrchu



Graf. D.11: Relativní chyba aproximace objemu

Všechny výše uvedené chyby jsou počítány obdobným způsobem jako tomu bylo u koule v kapitole 11.4 (rozdíl spočívá v tom, že u koule je chyba vztažena na její střed, kdežto u torusu na jeho prstencovitý střed ve tvaru kružnice o poloměru C).

Příloha E

Evidenční list

Souhlasím s tím, aby moje diplomová práce byla půjčována k prezenčnímu studiu v Univerzitní knihovně ZČU v Plzni.

V Plzni dne *20.5.2002* . . . . . .

Jan Patera

Uživatel stvrzuje svým čitelným podpisem, že tuto diplomovou práci použil ke studijním účelům a prohlašuje, že ji uvede mezi použitými prameny.

Jméno	Fakulta/Katedra	Datum	Podpis