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

Diplomová práce

Povrchové modely volumetrických dat

Autor : bc. Petr Bartoš
Obor : Informatika a výpočetní technika
Zaměření : Počítačová grafika
Rok vydání : 1999

Obsah

Obsah

1 Úvod

Zobrazování volumetrických dat lze rozdělit do tří základních kategorií : zobrazování tenkých vrstev a řezů, zobrazování izoploch a povrchových modelů, zobrazování komplexních modelů. Pro každou kategorii byly vyvinuty speciální algoritmy. V současné době neexistuje univerzální algoritmus, který by byl použitelný pro zobrazení všech výše uvedených typů zobrazení a zároveň byl optimální z hlediska časové náročnosti a kvality generovaného obrazu. Tato práce se dále zabývá především zobrazením a rekonstrukcí povrchových modelů.

Volumetrická data (objemová data) mohou mít různou podobu (viz. Kap. 2). V rámci této práce se soustředíme na zobrazování 3D skalárních polí jakožto speciálního případu volumetrických dat. Data tohoto typu lze získat např. z počítačového tomografu (CT - Computed Tomography, SPECT – Single-Photon Computed Tomography), magnetické rezonance (MR – Magnetic Resonance).

V první části této práce (Kap. 3) jsou popsány často používané algoritmy včetně identifikace jejich výhod a nevýhod. V následující části (Kap. 4) je navrženo jádro systému pro zobrazování a extrakci povrchových modelů.

Cílem je porovnání jednotlivých algoritmů z hlediska časové a paměťové náročnosti a algoritmické složitosti. Vzhledem k rozsahu problematiky zobrazování volumetrických dat, nebudou uvedeny všechny existující algoritmy a jejich modifikace, ale pouze vybrané metody.

2 Definice pojmů

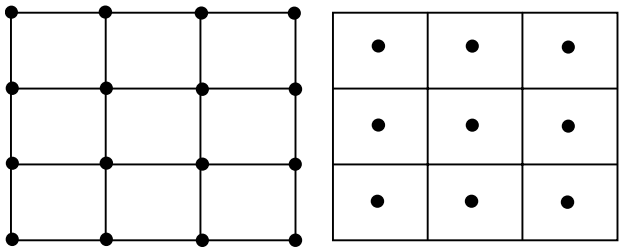
Vzhledem k tomu, že se v literatuře často zaměňují pojmy *voxel* a *cell* bude dále uvedena definice několika základních pojmů týkajících se volumetrických dat.

Volumetrická data (objemová data) jsou data ve tvaru (x_i, y_j, z_k, f_0, ... ,f_n) pro $0 \le i \le X$, $0 \le j \le Y$ a $0 \le k \le Z$, kde x_i, y_j, z_k jsou souřadnice bodu ve 3D a hodnoty f_0, ... ,f_n mohou obsahovat skalární či vektorovou hodnotu popisující nějakou vlastnost daného bodu (např. hustota materiálu, rychlost proudění ...). Jedná se tedy o množinu vzorků, kde ke každému vzorku existuje nějaká informace : skalární hodnota, vektor, pole hodnot, matice hodnot.

Vzorky mohou být uspořádány do mřížky. Mřížka dat může mít různé uspořádání [Zara98]. V této práci se dále budeme zabývat pouze takovými daty, kde jednotlivé vzorky jsou uspořádány v *pravidelné mřížce*, tj. celý prostor dat je rozdělen na stejně velké buňky tvaru kvádru. Mřížku, jejíž buňky tvoří identické krychle, budeme dále chápat jako speciální typ pravidelné mřížky (v [Zara98] nazvaná kartézská mřížka). Mřížku tvoří jednotlivé elementy označované jako *voxel* nebo *cell*.

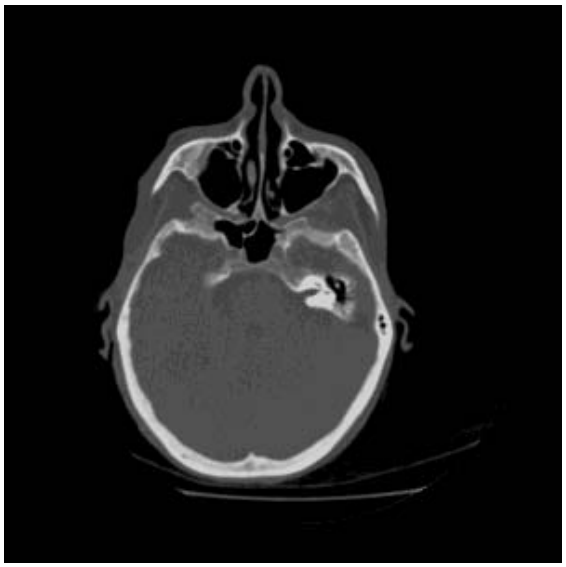
Voxel je základní objemový element, 3D obdoba pixelu. Vyplněný kvádr, který má v celém svém objemu konstantní hodnotu dané veličiny, a to takovou, jaká odpovídá části prostoru, jehož střed je určen souřadnicemi mřížky. Voxel je tedy určen svými souřadnicemi a hodnotou dané veličiny.

Cell *(buňka)* je kvádr, který tvoří osmice vzorků. Hodnoty uvnitř buňky se vypočítají pomocí lineární interpolace (velmi často) nebo interpolací vyššího řádu (méně často).

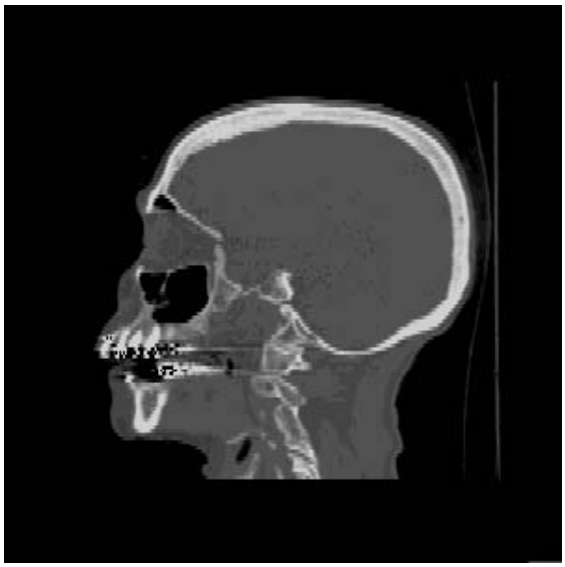


Obr. 2-1 Dva typy objemových elementů Cell (vlevo) a Voxel (vpravo)

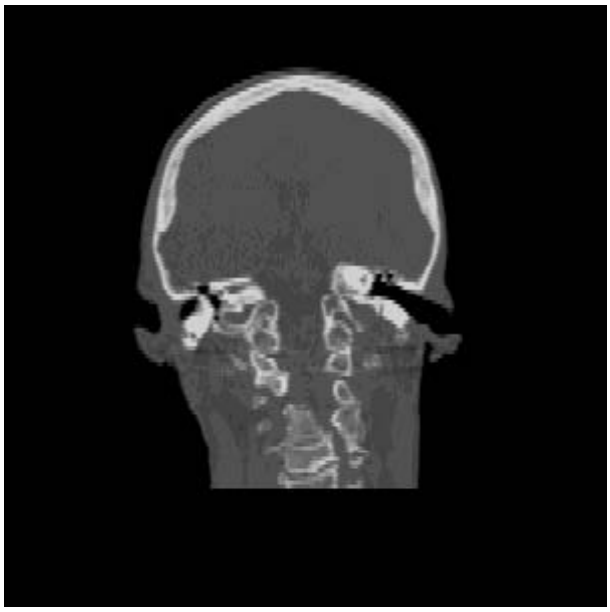
Slice (řez) je taková množina vzorků z pravidelné mřížky, kde všechny vzorky řezu mají shodnou jednu souřadnici. Lze tedy mluvit o řezu rovinou *xy*, pokud vzorky mají shodnou souřadnici *z* a analogicky lze definovat řezy rovinou *xz* a *yz*.



Obr. 2-2 Řez rovinou xy - CT snímek hlavy



Obr. 2-3 Řez rovinou yz - CT snímek hlavy



Obr. 2-4 Řez rovinou xz - CT snímek hlavy

3 Zobrazování volumetrických dat

Pro zobrazení 3D skalárních polí existuje několik algoritmů. V zásadě je lze rozdělit podle stylu práce na algoritmy hledající povrch (Marching Cubes, Marching Tetrahedra) a algoritmy přímého zobrazení (založeny na metodě sledování paprsku).

Algoritmy hledající povrch jsou konstruovány přímo pro zobrazení povrchového modelu. Nejdříve aproximují hledanou izoplochu sítí polygonů a následovně pomocí klasických technik počítačové grafiky danou síť zobrazují. Hlavní nevýhodou tohoto přístupu je, že generovaná síť polygonů může obsahovat řádově 10^6 polygonů a navíc velikost polygonů bývá často srovnatelná s velikostí pixelu. Výhodou je naopak možnost využití již existujících grafických akcelerátorů pro zobrazení sítě polygonů.

Algoritmy přímého zobrazení naopak eliminují kroky generování sítě a rasterizace polygonů.

3.1 Výpočet normály povrchu

Pro zobrazení povrchu při použití libovolného algoritmu je v případě požadavku na vysokou kvalitu výsledného obrazu nutno odhadnout normálu povrchu, neboť na normále povrchu závisí stínování. Normála povrchu se odhaduje z gradientu. Pro výpočet gradientu existuje několik metod. Základní metodou je Central Difference Method. Popis lze najít např. v [Bent96]. V [Zara98] je uveden výpočet normály z paměti hloubky nebo binárního objemu. Tyto metody jsou výpočetně méně náročné, ale ve výsledném obraze vytvářejí výrazné efekty v podobě vrstevnic.

Často používaná metoda pro výpočet gradientu je metoda, která vypočte gradient ze šesti sousedních vzorků podle vztahu [Bent96] :

$$g(x) = \frac{1}{2}(f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k))$$

$$g(y) = \frac{1}{2}(f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k))$$

$$g(z) = \frac{1}{2}(f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})),$$

kde g(x,y,z) je gradient a $f(x_i,y_j,z_k)$ je funkce jejíž funkční hodnoty byly navzorkovány.

Gradient lze také vypočítat z 26 sousedních hodnot, ale tento způsob se nepoužívá z důvodu mnohem delšího výpočtu, i když výsledek je samozřejmě přesnější.

Pro malé objekty může být nevhodný výpočet gradientu i ze 6 sousedních hodnot [Bent96], proto se používá adaptivní výpočet gradientu z 3-6 sousedních vzorků. Princip výpočtu adaptivního gradientu pro x-ovou složku je následující : Jestliže hodnota vzorku (i,j,k) je větší (menší) než hodnota sousedních vzorků (i+1,j,k) a (i-1,j,k) použije se hodnota (i,j,k) a menší (větší) z hodnot sousedů, v opačném případě se provede výpočet z obou sousedních hodnot vzorků. Analogicky se vypočte y-ová a z-ová složka gradientu.

Centrální diferenciální metodou lze tedy spočítat hodnotu gradientu v bodě o souřadnicích (i,j,k), tj. v místech vzorků. Hodnoty gradientu mezi vzorky se dále počítají lineární interpolací.

3.2 Marching Cubes

Algoritmus Marching Cubes je základním algoritmem pro rekonstrukci povrchu. Poprvé byl publikován v [LORE87]. Lze jej použít pro data uspořádaná do pravidelné mřížky.

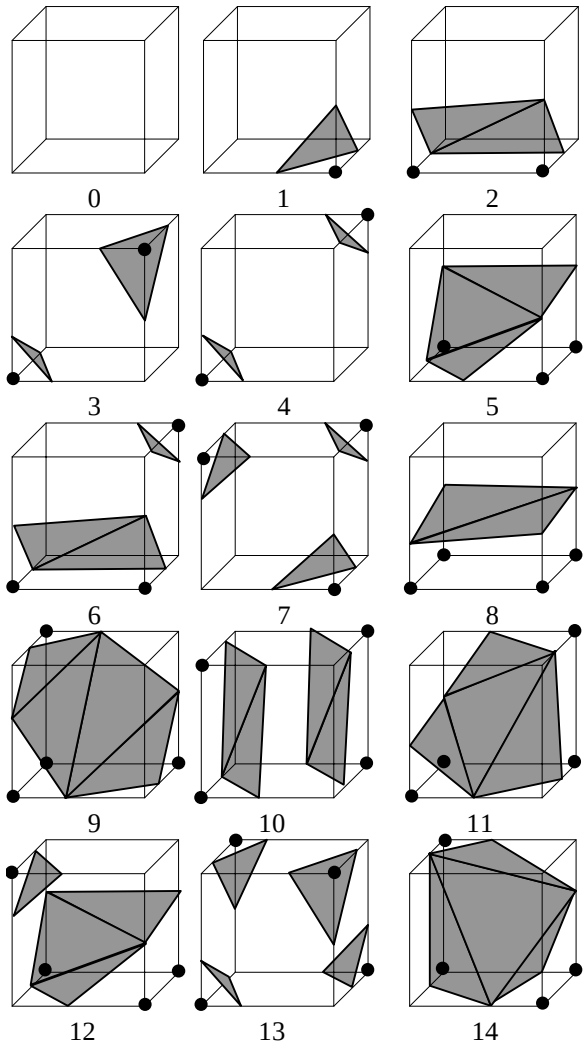
Algoritmus pracuje ve dvou fázích. V první fázi se generuje povrch (izoplocha) na základě hodnoty dané uživatelem a vypočtou se normály (centrální diferenciální metodou) ve vrcholech trojúhelníků. V druhé fázi se vygenerovaná síť zobrazuje.

Algoritmus postupně vytváří buňky tvaru krychle ze dvou čtveřic vzorků ze sousedních řezů. Pro každou buňku se zjistí, jak ji hledaná izoplocha protíná a přejde se ke zpracování další buňky.

Vrcholy každé buňky se ohodnocují podle toho, zda hodnota ve vrcholu je větší nebo rovna zadané hodnotě (vrchol je uvnitř plochy nebo na ploše) nebo je menší než daná hodnota (vrchol je vně plochy). Hledaná plocha pak protne pouze ty hrany krychle, jejichž ohodnocení koncových bodů je různé.

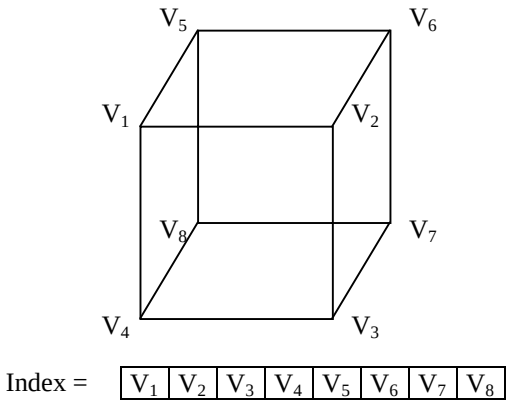
Protože krychle má osm vrcholů a každý vrchol může být ohodnocen jednou ze dvou hodnot, existuje tedy $2^8 = 256$ možných triangularizací krychle. Využitím symetrie

krychle a inverzí ohodnocení vrcholů lze získat 15 základních konfigurací (viz. Obr. 3-1).



Obr. 3-1 Triangularizace 15-ti základních konfigurací

Z předchozího odstavce plyne, že každou buňku lze ohodnotit celým nezáporným číslem z intervalu <0,255> . Toto číslo pak tvoří index do tabulky, v níž jsou uloženy hrany mající průsečík s hledanou izoplochou pro všechny možné konfigurace.



Obr. 3-2 Číslování vrcholů krychle a index krychle

Sled operací pro jednu buňku :

a) Vytvoření krychle

b) Ohodnocení vrcholů (uvnitř nebo vně) a sestavení indexu

c) Nalezení hran, které plocha protíná

d) Výpočet souřadnic trojúhelníků lineární interpolací

e) Výpočet normál ve vrcholech trojúhelníků centrální diferenční metodou a lineární interpolací

Pokud se využije skutečnosti, že každá hrana krychle je společná ještě dalším třem krychlím (vyjma speciálních případů, kdy daná hrana tvoří zároveň kraj mřížky), lze docílit pouze jednoho výpočtu vrcholu a normály pro jednu hranu.

3.2.1 Nedostatky algoritmu

Ačkoliv se na první pohled může algoritmus Marching Cubes zdát ideálním algoritmem pro rekonstrukci povrchů, byly v následujících letech po jeho publikování identifikovány některé nedostatky :

- Algoritmus může generovat povrch s děrami [Zara98].
- Algoritmus může generovat redundantní trojúhelníky [Li97], které mohou degenerovat na úsečky a body.

- Algoritmus generuje příliš velký počet trojúhelníků, které se velikostí blíží rozměru pixelu.
- Algoritmus musí projít všechny buňky.

3.2.2 Problém děr

Díry mohou vznikat při rekonstrukci povrchu z toho důvodu, že se při triangularizaci povrch určuje pouze na základě osmi vrcholů krychle a ne globálně s ohledem na její okolí [Zara98],[Zhou94]. V [Zhou94] lze nalézt modifikaci algoritmu Marching Cubes, která řeší problém děr přidáním dalších tří základních konfigurací. Jiným možným řešením problému, je prosté zalepení vzniklé díry dvěma trojúhelníky nebo čtyřúhelníkem [Zara98].

3.2.3 Redundantní trojúhelníky

Redundantní trojúhelníky mohou vznikat v případě, že hledaný povrch je určen hodnotou shodnou s hodnotou některého vrcholu krychle. Např. pro případ 1 z Obr.3.1 může trojúhelník degenerovat na bod pro případ 2 z Obr. 3.1. pak na úsečku [Li97]. Body ani úsečky nemohou mít žádný efekt na výslednou plochu.

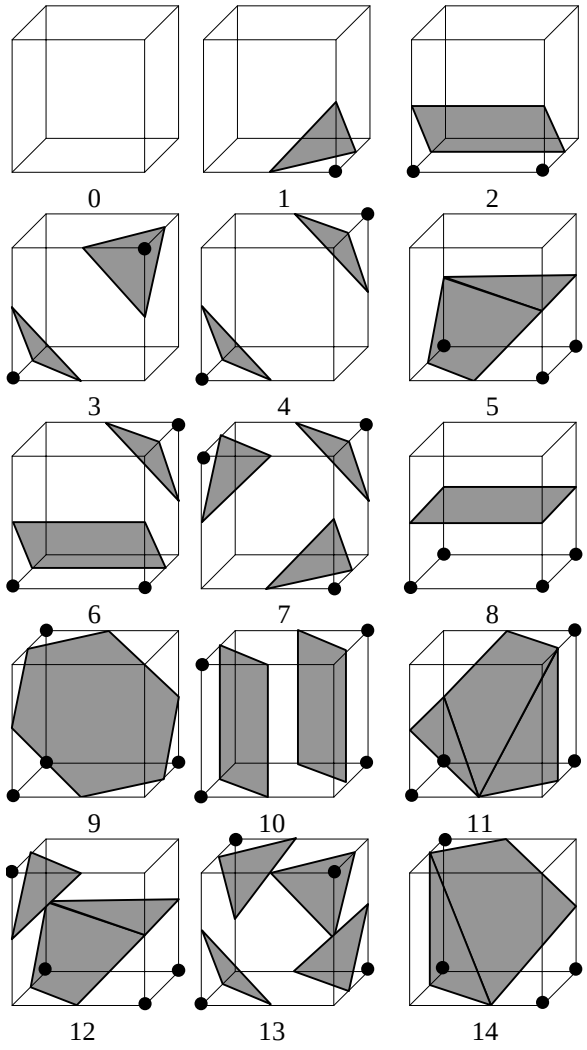
Řešení uvedené v [Li97] spočívá v identifikaci problematických vrcholů u všech základních konfigurací. Jednotlivé vrcholy se ohodnocují jako v původním algoritmu a navíc se ke každému vrcholu přiřadí další ohodnocení v případě, že daný vrchol leží na hledaném povrchu. Tato nová hodnota udává na kolika hranách vedoucích z daného vrcholu jsou průsečíky s izoplochou. Autoři uvádí, že problematické jsou pouze vrcholy s ohodnocením 2 a 3. Takovému vrcholu krychle pak přiřadí ohodnocení 0, když je vrchol na povrchu, nebo 1 v opačném případě.

Pokud je detekován problematický vrchol, provede se přeindexování do tabulky případů a dále se pokračuje podle původního algoritmu.

3.2.4 Snížení počtu generovaných trojúhelníků

Pro velké soubory vstupních dat např. 1024×1024×100 může algoritmus Marching Cubes generovat až 2 miliony trojúhelníků [Zara98]. Snahou tedy bylo snížit počet trojúhelníků při zachování relativně malé chyby.

Prvním z možných způsobů je místo trojúhelníků generovat obecně síť polygonů (trojúhelníků až šestiúhelníků) [Li97]. Počet těchto polygonů je menší a zároveň je větší jejich plocha. Autoři této modifikace algoritmu Marching Cubes nepoužívají pro výpočet vrcholu trojúhelníku (polygonu) lineární interpolaci, ale umístí vrchol polygonu do středu hrany krychle. Z této skutečnosti pak plyne, že některé trojúhelníky ze základních topologických konfigurací lze spojit do čtyřúhelníků a šestiúhelníků (Obr. 3-3), protože tyto trojúhelníky leží v jedné rovině. Pomocí této techniky pak dojde k redukci počtu polygonů sítě o 30 % až 60 % [Li97].



Obr. 3-3 15 základních konfigurací při použití středního bodu místo lineární interpolace

Další možnost, jak snížit počet trojúhelníků v generované síti, je decimace trojúhelníkové sítě. V [Shek96] byla publikována modifikace algoritmu Marching Cubes

umožňující decimaci sítě již v průběhu generování izoplochy. Algoritmus pak generuje síť trojúhelníků na základě uživatelem zadané hodnoty jako v případě původního algoritmu, a uživatelem definované chyby. Hledaný povrch se aproximuje pomocí větších trojúhelníků v těch částech objemu, kde se hodnoty vzorků mění s menší frekvencí. Pro hodnotu chyby 0.1 dochází ke snížení počtu trojúhelníků na 68 % až 88 % oproti původnímu algoritmu při vizuálně nezjistitelných změnách generovaného povrchu. Detailní popis lze najít v [Shek96].

3.2.5 Další modifikace algoritmu

Mezi další často se vyskytující modifikace patří použití různých stromových struktur např. [Chua95], které slouží k rychlému přeskočení té části objemu, která neobsahuje buňky protnuté konstruovaným povrchem. Vzhledem k tomu, že vstupní data obsahují 10 % až 25 % buněk protínajících povrch [Li97] a zbylých 75 % až 90 % buněk se pouze testuje, může použití stromové struktury významně zvýšit rychlost výpočtu. Na druhou stranu však tyto datové struktury mohou narůst do velikosti srovnatelné s velikostí vstupního souboru.

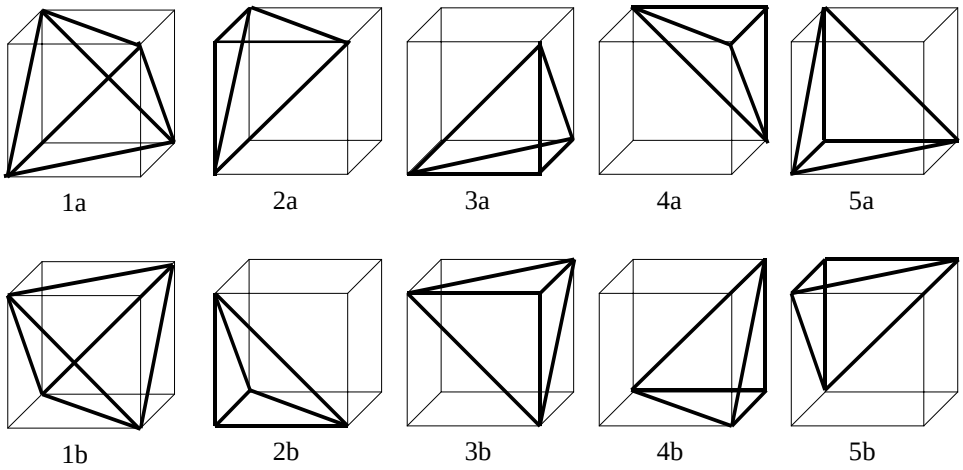
V [Zhou95] se lze dočíst o konstrukci trojúhelníkové sítě ve speciálním tvaru (tzv. triangle strip), který redukuje nadbytečné ukládání informací a urychluje zobrazení pomocí OpenGL.

3.3 Marching Tetrahedra

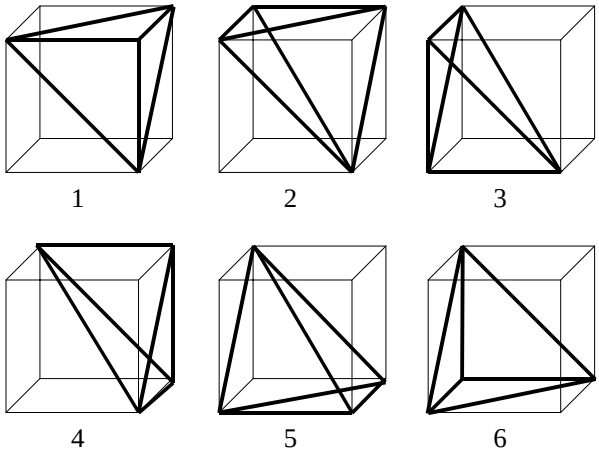
Marching Tetrahedra je algoritmem pracujícím obdobně jako Marching Cubes. Postupně zpracovává objemové elementy jež tvoří tetrahedrony. Stejně jako v Marching Cubes se ohodnotí všechny vrcholy čtyřstěnu podle kritéria, zda hodnota ve vrcholu je uvnitř nebo vně povrchu. Po ohodnocení se provede výpočet vrcholů trojúhelníků na těch hranách, které mají různé ohodnocení počátečního a koncového bodu. Poté následuje triangularizace.

Výhodou algoritmu je, že jej lze použít i pro vzorky v neuspořádané mřížce. V tomto případě je nejdříve nutné zkonstruovat z dané množiny bodů síť tetrahedronů např. pomocí Delaunayovy triangularice ve 3D. Pro pravidelnou mřížku je konstrukce tetrahedronů jednodušší. Jednu buňku tvaru krychle lze rozdělit na 5 (Obr. 3-4), 6 (Obr. 3-5), 12 (Obr. 3-6), 24 nebo 48 tetrahedronů. Krychli lze rozdělit na pět tetra-

hedronů dvěma způsoby. Tato dělení je nutno z důvodu zachování návaznosti pravidelně střídat. Zlepšení návaznosti lze dosáhnout dělením na vyšší počet tetrahedronů, ale opět se začne zvyšovat počet generovaných trojúhelníků.

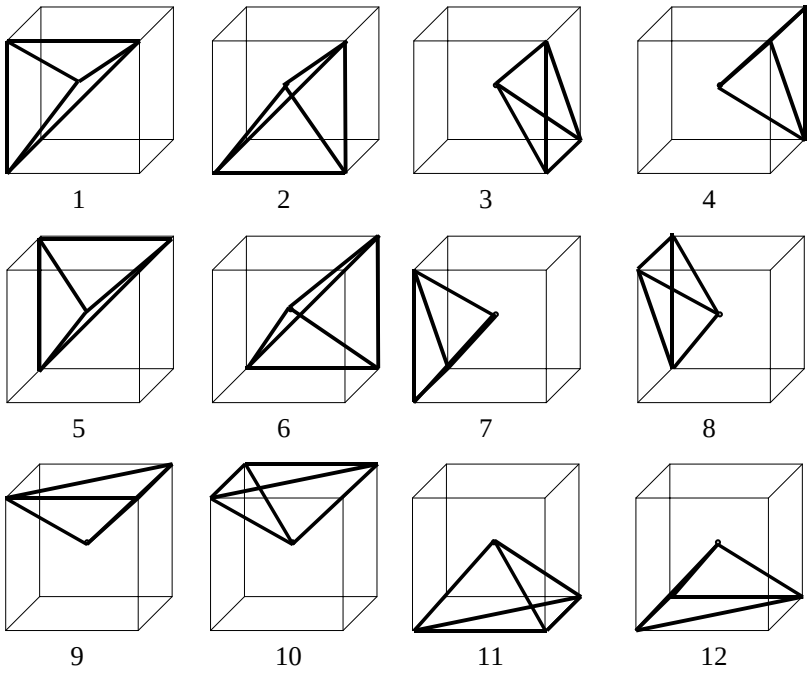


Obr. 3-4 Dva způsoby dělení krychle na pět tetrahedronů



Obr. 3-5 Jeden z možných způsobů dělení krychle na šest tetrahedronů

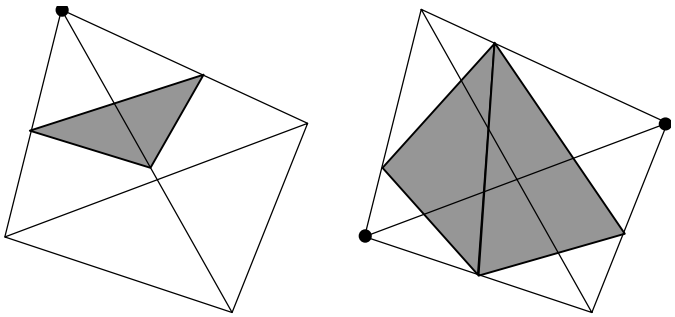
Na následujícím obrázku je zobrazeno dělení krychle na šest tetrahedronů. Bod uvnitř krychle je těžiště krychle. Hodnota v tomto bodě je spočtena z osmi vzorků tvořících krychli pomocí trilineární interpolace. Toto dělení krychle má pak tu výhodu, že všechny tetrahedrony jsou shodné, tj. mají stejný objem.



Obr. 3-6 Dělení krychle na dvanáct tetrahedronů

Další výhodou použití tetrahedronu jako objemového primitiva je, že se eliminuje problém děr známý z algoritmu Marching Cubes. Na druhou stranu se ale zvýší počet trojúhelníků a navíc pro dělení na pět tetrahedronů je nutné pravidelné střídání.

Existuje $2^4 = 16$ různých ohodnocení tetrahedronu. Díky symetrii dostaneme dva základní způsoby triangularizace na tetrahedronu (Obr. 3-7).



Obr. 3-7 Dva způsoby triangularizace na tetrahedronu

3.4 Porovnání Marching Cubes a Marching Tetrahedra

Jak již bylo uvedeno dříve základní nevýhodou obou algoritmů je generování relativně velkého počtu trojúhelníků (polygonů) o velikosti srovnatelné s velikostí pixelu. Rasterizace tohoto počtu trojúhelníků (polygonů) je pak časově dosti náročná. Z tohoto hlediska je lepší algoritmus Marching Cubes, protože generuje menší počet trojúhelníků.

Pokud je potřeba zobrazovat data z jiné než pravidelné mřížky, nezbyde než použít Marching Tetrahedra, tj. Marching Tetrahedra je použitelný pro více druhů vstupních dat.

Marching Tetrahedra pak dále generuje povrch bez děr na rozdíl od původního Marching Cubes. Marching Tetrahedra je také jednodušší na implementaci, neboť obsahuje podstatně menší počet možných triangularizací objemového elementu oproti (tetrahedronu) algoritmu Marching Cubes.

Oba dva algoritmy jsou vhodné pro zobrazení povrchových modelů z hlediska kvality výsledného zobrazení. V obecné rovině nelze říci, který z algoritmů je lepší. V případě, že se omezíme na data uspořádaná v pravidelné mřížce, je výhodnější použít algoritmus Marching Cubes, protože algoritmus Marching Cubes aproximuje hledanou izoplochu pomocí menšího počtu trojúhelníků (polygonů) než algoritmus Marching Tetrahedra. Nižší počet trojúhelníků znamená rychlejší zobrazení a o rychlost zobrazení jde především. Další výhodou algoritmu Marching Cubes je existence velkého množství modifikací, ze kterých lze téměř vždy vybrat výhodné řešení dané úlohy. Některé modifikace sice řeší problém vzniku děr a velkého množství generovaných trojúhelníků, ale často za cenu významně stížené implementace oproti původnímu algoritmu.

3.5 Dividing Cubes

Algoritmus Dividing Cubes byl publikován v roce 1988 v [Cline88]. K jeho vzniku vedla snaha o eliminaci rasterizace velkého počtu trojúhelníků, které jsou výsledkem dříve uvedených algoritmů.

Základním principem algoritmu je reprezentace hledaného povrchu pomocí množiny bodů ležících na hledané ploše a nikoliv pomocí trojúhelníků nebo polygonů. K tomuto závěru autory dovedla úvaha o tom, že s rostoucím rozlišením vstupních dat

roste počet polygonů generovaných algoritmem Marching Cubes a tyto polygony po projekci na průmětnu displeje mají velikost několika málo pixelů. Takže vygeneruje-li algoritmus dostatečně velký počet bodů ležících na dané izoploše, je možné docílit toho, aby každý generovaný bod byl mapován právě na jeden pixel displeje.

Algoritmus Dividing Cubes rozdělí všechny buňky objemu tvaru krychle, které protíná daná izoplocha na menší krychle a následně promítne vypočtenou barvu a intenzitu každé malé krychle na průmětnu. Počet krychlí, na který je původní buňka rozdělena se volí tak, aby velikost krychliček byla totožná s velikostí pixelu na průmětně.

Pro každou buňku, kterou protíná izoplocha je nutné provést následující operace :

a) Test, zda danou buňku protíná izoplocha
b) Výpočet normály ve všech vrcholech buňky
c) Rozdělení buňky na $i \times j \times k$ krychliček
d) Identifikace těch krychliček, které protíná daná izoplocha
e) Výpočet normály pro všechny krychličky uvedené v bodě d) pomocí trilineární interpolace
f) Projekce všech krychliček z bodu d) do průmětny a výpočet světelné intenzity

Eliminaci neviditelných krychliček lze řešit pomocí z-bufferu.

Výhodou algoritmu Dividing Cubes je eliminace rasterizace polygonů, ale na druhou stranu se pro každou buňku provádí více aritmetických operací oproti algoritmu Marching Cubes. Velkou nevýhodou pak je skutečnost, že při změně velikosti průmětny (pouze při zvětšení rozlišení) je nutné znovu generovat množinu bodů, protože pro větší průmětnu (menší pixel) je nutné jemnější dělení původních buněk, aby nedocházelo ke vzniku děr v zobrazovaném povrchu.

4 Realizace

4.1 Výběr algoritmů pro implementaci

Pro implementaci byly vybrány následující algoritmy :

- Marching Cubes
- Marching Tetrahedra

Algoritmus Marching Cubes byl vybrán, protože je považován za základní algoritmus rekonstrukce povrchových modelů z objemových dat. Dále bude implementována modifikace s použitím středu úsečky místo lineární interpolace pro výpočet průsečíku hledané izoplochy a hrany krychle (viz. Kap. 3.2.4). Algoritmus Marching Tetrahedra je vybrán pro porovnání. Bude implementován pro tři různá dělení krychle, tj. na 5,6 a 12 tertrahedronů (viz. Kap. 3.3).

4.2 Specifikace požadavků

Cílem je realizace knihovny funkcí pro vizualizaci objemových dat a ukázkové aplikace, která bude demonstrovat použití dané knihovny. Knihovna musí mít takovou formu, aby ji bylo možné snadno integrovat do jiného vizualizačního softwaru, tj. jasně definované aplikační rozhraní nezávislé na žádné objektové knihovně jako jsou VCL firmy Borland nebo MFC firmy Microsoft..

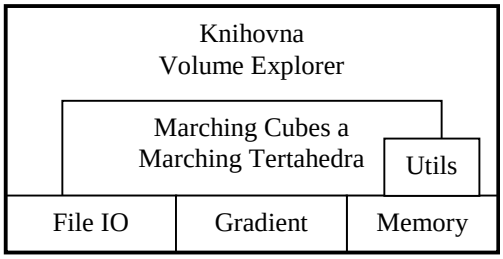
Knihovna musí nabízet funkce a procedury pro generování izoploch z objemových dat pomocí algoritmů Marching Cubes (MC) a Marching Tetrahedra pro dělení krychle na 5, 6 a 12 tetrahedronů (MT5, MT6, MT12). Všechny algoritmy musí být schopny používat různé funkce pro výpočet normály povrchu. Dalším požadavkem na knihovnu funkcí je reentrantnost všech implementovaných funkcí a procedur.

Aplikace by pak měla být schopna zpracovávat data v tzv. „syrové“ podobě s různou hlavičkou souboru a různou implementací hodnoty vzorky, tj. vzorku mohou být v soubory uloženy jako celé číslo na 8 nebo 16-ti bitech a to jak se znaménkem tak bez znaménka. Dalším požadavkem na demonstrační aplikaci je možnost měření časů výpočtů, zobrazení počtu vygenerovaných vrcholů a polygonů a možnost snadné rotace zobrazené izoplochy.

4.3 Design

4.3.1 Knihovna

Jak bylo uvedeno v kap. 4.2, musí knihovna implementovat algoritmy Marching Cubes a Marching Tetrahedra a různé způsoby výpočtu gradientu (normály povrchu). Vzhledem k netriviálnosti a rozsahu dané úlohy bude knihovna rozdělena do modulů jak ukazuje obr. 4.1.



Obr. 4-1 Architektura knihovny

Popis funkce jednotlivých modulů :

- *File IO* – operace pro čtení souboru v tzv. syrovém formátu, s různým počtem bytů na vzorek a s různou hlavičkou.
- *Gradient* – výpočet gradientu metodami : centrální diferenciální metoda, adaptivní výpočet gradientu, výpočet z 18-ti sousedů a výpočet z 26-ti sousedů
- *Memory* – alokování a uvolňování paměti pro potřeby knihovny a aplikace
- *Utils* – pomocné rutiny, např. konstrukce histogramu, vytvoření display listu z rekonstruované izoplochy, konverze bloku dat na vnitřní reprezentaci, zjištění minimální a maximální hodnoty z načtených dat, vykreslení modelu pomocí OpenGL, atd.
- *Marching Cubes* – generování izoplochy algoritmem Marching Cubes
- *Marching Tetrahedra* – generování izoplochy algoritmem Marching Tetrahedra

4.3.2 Aplikace

Aplikace pro demonstraci funkčnosti knihovny Volume Explorer nebude realizována jako plnohodnotný software pro vizualizaci dat, ale pouze jako ukázková aplikace nabízející základními funkce jako načtení souboru, určení hodnoty hledané izoplochy,

vygenerování izoplochy s použitím knihovny Volume Explorer, vykreslení izoplochy s použitím OpenGL a zobrazení histogramu, doby výpočtu a informace o vygenerované izoploše.

Aplikace bude nabízet grafické uživatelské rozhraní odpovídající současným zvyklostem uživatelů operačních systémů MS Windows 95/98 a MS Windows NT 4.0. Aplikace bude realizována jako MDI aplikace, což umožní vložit všechna okna programu do hlavního okna aplikace a zabrání rozmístění oken po celé obrazovce. Jinými slovy, díky tomuto návrhu uživatelského rozhraní působí aplikace jako jeden celek a nenutí uživatele hledat další okno patřící aplikaci mezi okny jiných spuštěných programů.

Uživatelské rozhraní se bude skládat z následujících typů formulářů (oken) :

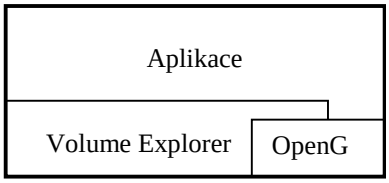
- Formulář pro zobrazení 3D modelů
- Informační formuláře – doba výpočtu, počet vrcholů a polygonů izoplochy, informace o souboru, histogram
- Nemodální formuláře s editačními prvky – změna barvy modelu, změna konstanty udávající izoplochu
- Modální formuláře s editačními prvky – nastavení parametrů programu a algoritmů

4.3.3 Vztah mezi aplikací a knihovnou funkcí

Knihovna Volume Explorer bude nabízet jasně definované rozhraní a bude podporovat zobrazení izoploch pomocí OpenGL.

OpenGL bylo pro zobrazení vygenerované izoplochy vybráno z toho důvodu, že dnes existuje její implementace jak pod Windows NT tak pod Windows 95/98 a dalším důvodem je poměrně snadná dostupnost grafických akcelerátorů, které podporují OpenGL. Nezanedbatelný fakt je také to, že se aplikační programátor nemusí zabývat implementací vlastní grafické knihovny obsahující základní algoritmy počítačové grafiky. Stručně řečeno OpenGL je obecně přijatý standard, nezávislý na platformě a konkrétním hardwaru, který zajistí korektní zobrazení na libovolném počítači, ať s grafickým akcelerátorem nebo bez něj. Rozdíl je pouze v rychlosti zobrazení.

Knihovna bude podobně jako OpenGL nabízet jisté služby aplikačním programátorům bez nároku, aby programátor byl blíže seznámen s vlastní implementací jednotlivých funkcí. Vztah mezi aplikací a knihovnami Volume Explorer a OpenGL ukazuje obr.4.2.



Obr. 4-2 Vztah mezi aplikací a knihovnami Volume Explorer a OpenGL

4.4 Datové struktury

Pro zpracování volumetrických dat a generování izoploch jsou potřebné dvě základní datové struktury : datová struktura pro vstupní data a datová struktura pro uložení výsledné izoplochy.

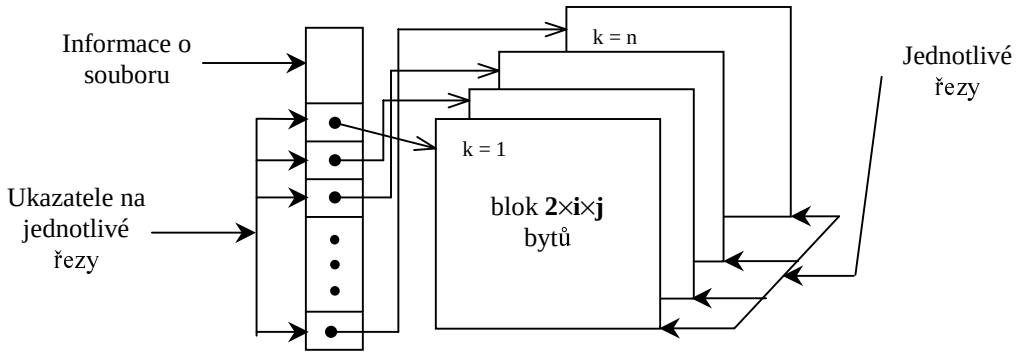
4.4.1 Reprezentace 3D skalárního pole

Jak již bylo uvedeno dříve mohou mít vstupní data různou podobu. Jednotlivé vzorky mohou být uloženy ve vstupním souboru jako celé číslo se znaménkem nebo bez znaménka a navíc na různém počtu bytů.

Bylo by vhodné navrhnout nějakou vnitřní reprezentaci hodnoty vzorku, která by umožnila uložení všech výše uvedených typů hodnot. Každý vzorek lze vnitřně reprezentovat jako 16-bitové číslo bez znaménka. Šestnáct bitů na vzorek je dostatečných, protože současné skenery ukládají hodnoty převážně v rozsahu 12-ti bitů. Znaménkové číslo lze převést na neznaménkové např. následujícím způsobem. Zjistí se minimum ze všech vzorků v souboru a následně se pro každý vzorek vypočte rozdíl mezi původní hodnotou a dříve zjištěným minimem, tj. provede se posunutí na číselné ose směrem doprava. Výsledkem je, že všechny hodnoty jsou větší nebo rovny nule. Je samozřejmě nutné uchovat hodnotu původního minima pro zpětný výpočet původních hodnot vzorků.

Každý soubor obsahuje k řezů s daným rozlišením $i \times j$. Každý řez lze reprezentovat jako souvislý blok dat o velikosti $2 \times i \times j$ bytů. Celý objem je pak možné uložit jako seznam řezů. Tato datová struktura pak musí být doplněna o další informace, tj. počet

řezů a rozlišení řezů v obou osách, dále pak o hodnotu offsetu uvedenou v předchozím odstavci. Představa datové struktury je zobrazena na obr. 4.3.



Obr. 4-3 Datová struktura pro uložení vstupních dat

4.4.2 Reprezentace izoplochy

Hledaná izoplocha je aproximována množinou rovinných ploch. Původní algoritmus Marching Cubes generuje izoplochu jako množinu trojúhelníků. Použijeme-li však pro výpočet průsečíku hledané izoplochy a hrany krychle střed úsečky místo lineární interpolace, můžeme aproximovat hledanou izoplochu pomocí trojúhelníků, čtyřúhelníků a šestiúhelníků.

Z předchozího odstavce plyne, že datová struktura pro uložení izoplochy musí umožnit ukládání tří různých typů polygonů. Množinu těchto typů polygonů lze reprezentovat dvěma způsoby :

- Seznamem trojúhelníků, čtyřúhelníků a šestiúhelníků, kde jednotlivé záznamy v seznamech obsahují kompletní informaci o vrcholech.
- Seznamem vrcholů, trojúhelníků, čtyřúhelníků a šestiúhelníků, kde záznamy seznamů polygonů obsahují pouze index do seznamu vrcholů.

Druhý způsob uložení informace je mnohem efektivnější z hlediska paměťové náročnosti, protože eliminuje nutnost vícenásobného uložení stejných vrcholů, a proto byl tento způsob vybrán i pro implementaci.

Izoplocha je tedy tvořena čtyřmi seznamy :

- Seznam vrcholů – každý záznam obsahuje souřadnice bodu ve 3D a normálu povrch v daném bodě.
- Seznam trojúhelníků – každý záznam obsahuje tři indexy do seznamu vrcholů.
- Seznam čtyřúhelníků – každý záznam obsahuje čtyři indexy do seznamu vrcholů.
- Seznam šestiúhelníků – každý záznam obsahuje šest indexů do seznamu vrcholů.

4.5 Implementace

Pro implementaci vybraných algoritmů a ukázkové aplikace byl vybrán programovací jazyk Object Pascal, protože se jedná o univerzální programovací jazyk s dobrým překladačem a optimalizátorem strojového kódu, dále vývojářský nástroj Borland Delphi 3.0, protože podporuje jednak realizaci výpočetních modulů, ale také velmi rychlý návrh uživatelského rozhraní pomocí vizuálních nástrojů.

Knihovna implementující algoritmy, byla realizována jako dynamicky linkovaná knihovna a to z toho důvodu, aby mohli danou knihovnu použít i programátoři pracující s jiným programovacím jazykem a jiným vývojářským nástrojem. Velký důraz byl při psaní zdrojového kódu kladen na optimalizaci na rychlost.

4.5.1 Rozhraní knihovny Volume Explorer

Rozhraní knihovny tvoří 21 funkcí a procedur, které lze logicky rozdělit do čtyř skupin :

- Funkce pro čtení dat ze souboru – 2 funkce
- Funkce pro zprávu paměti – 10 funkcí
- Pomocné funkce a procedury – 7 funkcí
- Funkce pro generování izoplochy algoritmy Marching Cubes a Marching Tetrahedra – 2 funkce

Konvence předávání parametrů byla zvolena standardní konvence operačního systému MS Windows, tj. parametry jsou do zásobníku ukládány zprava doleva a od-

stranění parametrů ze zásobníku musí provést volající. Tato konvence je dostupná pro všechny překladače na platformě Windows, protože jsou tímto způsobem předávány i parametry při volání funkcí Windows API. Zdrojový kód rozhraní knihovny v jazyce Object Pascal lze nalézt na přiloženém CD v adresáři *RunImage\Source DLL\Interface.*

4.5.1.1 Funkce pro čtení dat ze souboru

Pro čtení dat nabízí knihovna dvě funkce :

- ```
  function veLoadInfo (var I : vePFileInfo;const FN : veString);
  ```
- ```
  function veLoadFile (const I : vePFileInfo;
                       const FN : veString;
                       const ClbP : veCallBack) : vePVolume;
  ```

Pomocí funkce *veLoadInfo* lze načíst informace o způsobu uložení dat z textového souboru (Volume Info File – VIF soubor). Formát tohoto souboru lze najít v příloze „Příloha C – Formát souboru VIF“. Tento způsob popisu formátu vstupních dat byl vybrán z toho důvodu, že data na kterých byly algoritmy testovány pochází z různých zdrojů, v každém souboru je jiný formát tzv. hlavičky a v některých souborech není hlavička vůbec uložena. Ačkoliv knihovna nabízí tento způsob popisu struktury vstupního souboru, je možné nastavení správných hodnot položek struktury *veFileInfo* jakýmkoliv jiným způsobem např. přiřazením hodnot ve zdrojovém kódu.

Funkce *veLoadFile* načte datový soubor a vrací ukazatel na strukturu *veVolume.*

4.5.1.2 Funkce pro zprávu paměti

Pro ulehčení práce případným programátorům, kteří by chtěli používat knihovnu Volume Explorer jsou k dispozici tyto funkce pro vytvoření a rušení datových struktur :

- ```
  function  veCreateFileInfo : vePFileInfo;
  ```
- ```
  procedure veDestroyFileInfo(var FI : vePFileInfo);
  ```
- ```
  function  veCreateVolume : vePVolume;
  ```
- ```
  procedure veDestroyVolume(var V : vePVolume);
  ```
- ```
  function  veCreateIsoSurface : vePIsoSurface;
  ```
- ```
  procedure veDestroyIsoSurface(var I : vePIsoSurface);
  ```
- ```
  function  veCreateHistogram(const V : vePVolume) :
            vePHistogram;
  ```
- ```
  procedure veDestroyHistogram(var H : vePHistogram);
  ```

- `function veCreateMinMaxStruct(const VI : veFileInfo; const C : veByte) : vePMinMaxStruct;`
- `procedure veDestroyMinMaxStruct(var X : vePMinMaxStruct);`

Výše uvedené funkce a procedury slouží k alokování a uvolňování paměti pro datové struktury *veFileInfo* (informace o souboru), *veVolume* (reprezentace 3D skalárního pole), *veIsoSurface* (datová struktura pro uložení vygenerované izoplochy), *veHistogram* (histogram) a *veMinMaxStruct* (akcelerační struktura).

4.5.1.3 Pomocné funkce a procedury

- `procedure veCreateBox (const M : veInteger; const V : vePVolume);`
- `procedure veCreateModel (const M : veInteger; const V : vePVolume; var IsoSurf : vePIsoSurface; const FreeIsoSurf : veBoolean);`
- `procedure veCorrectOrientation(const V : vePVolume; var IsoSurf : vePIsoSurface);`
- `procedure veDrawModel(const V : vePVolume; var IsoSurf : vePIsoSurface);`
- `procedure veFillMinMaxStruct(const V : vePVolume; const X : vePMinMaxStruct);`
- `function veGetHistogram(const V : vePVolume; const ClbP : veCallBack) : vePHistogram;`
- `procedure veIsoSurfaceInfo(const Iso : vePIsoSurface; var Vc,Tc : veCardinal);`

Procedury *veCreateBox*, *veCreateModel* a *veDrawModel* volají funkce OpenGL. První dvě z těchto funkcí vytvoří „display list“ pro vykreslení ohraničujícího kvádru, respektive izoplochy vygenerované z dat. Třetí procedura pouze vykreslí izoplochu, nevytvoří „display list“. Je možné, aby aplikace použila libovolného přístupu k vykreslení izoplochy.

Použití „display listů“ je výhodné z toho důvodu, že po popsání modelu, popř. scény, jsou data uložena uvnitř OpenGL v podobě, která je nejbližší pro konkrétní hardwarový akcelerátor. Stručně řečeno by používání „display listu“ mělo zajistit rychlejší vykreslení izoplochy. Bohužel však po vytvoření „display listu“ uloží OpenGL izoplochu v nejméně efektivní podobě z hlediska paměťové náročnosti, tj. každý polygon z dané izoplochy je uložen jako záznam obsahující kompletní informaci o každém vrcholu včetně normály. Uvědomíme-li si skutečnost, že každý vrchol polygonu (troj-

úhelníku) je také vrcholem dalších tří jiných polygonů (trojúhelníků), dojdeme k závěru, že OpenGL potřebuje pro uložení izoplochy po vytvoření „display listu“ přibližně 3 × více paměti než kolik je potřeba pro její uložení v datové struktuře *veIsoSurface* (viz. kap. 4.4.2).

Tento problém by bylo možno odstranit pomocí rozšíření OpenGL, které dovoluje reprezentovat plochu jako množinu polygonů pomocí dvou seznamů, tj. seznamu vrcholů a seznamu polygonů (resp. Seznamu trojúhelníků, čtyřúhelníků a polygonů). Jenže firma Microsoft zatím nebyla schopna toto rozšíření začlenit do jejich implementace OpenGL a tak musíme zůstat pouze u teoretických úvah o tom, jak by bylo rychlé zobrazení, kdyby … .

Procedura v*eCorrectOrientation* provede převod souřadného systému, v kterém byla vygenerována izoplocha, do souřadného systému OpenGL.

Procedura *veFillMinMaxStruct* naplní dříve vytvořenou akcelerační strukturu správnými údaji o minimální a maximální hodnotě vzorku (podrobnosti kap.4.5.2).

Pomocí funkce *veGetHistogram* lze získat histogram hodnot obsažených v souboru.

Procedura *veIsoSurfaceInfo* vypočte počet vrcholů a počet polygonů, které tvoří izoplochu.

4.5.1.4 Funkce pro generování izoplochy

Nejdůležitějšími funkcemi knihovny jsou pak funkce pro generování izoplochy :

- ```
  function veMarchingCubes(const Volume : vePVolume;
                           const Params : veMCParams;
                           const MinMaxStruct : vePMinMaxStruct;
                           Iso : veInteger) : vePIsoSurface;
  ```
- ```
  function veMarchingTetrahedra(const Volume : vePVolume;
                          const Params : veMTParams;
                          const MinMaxStruct : vePMinMaxStruct;
                          Iso : veInteger) : vePIsoSurface;
  ```

Jak už názvy napovídají, funkce *veMarchingCubes* generuje izoplochu pomocí algoritmu Marching Cubes a druhá funkce pomocí algoritmu Marching Tetrahedra. Po-

mocí struktur *veMCParams* a *veMTParams* lze nastavit způsob výpočtu gradientu a způsob výpočtu průsečíků izoplochy s hranou objemových elementů.

Následující fragment zdrojového kódu ukazuje, jak z pomocí knihovny Volume Explorer načíst soubor, vygenerovat izoplochu a zobrazit izoplochu pomocí OpenGL. Předpokládejme, že chceme zpracovat soubor v adresáři *C:\Data*, který se jmenuje *hlava.dat*. Soubor s informacemi o formátu dat je ve stejném adresáři a jmenuje se *hlava.vif*. Dále požadujeme generování izoplochy dané hodnotou 128, tuto izoplochu chceme generovat pomocí algoritmu Marching Cubes bez použití akcelerační struktury, gradient budeme počítat z 26-ti sousedních vzorků a pro výpočet průsečíků hran objemových elementů s izoplochou použijeme lineární interpolaci.

```
var FileInfo : vePFileInfo;
    Volume   : vePVolume;
    IsoSurf  : vePIsoSurface;
    MCParams : veMCParams;

// načtení souboru
FileInfo := veCreateFileInfo;
if veLoadInfo(FileInfo,'C:\Data\hlava.vif') = ve_TRUE then
  begin
    Volume := veLoadFile(FileInfo,'C:\Data\hlava.dat' ,ve_NIL);
    if Volume <> ve_NIL then
      begin
        // generování izoplochy
        with MCParams do
          begin
            Acceleration  := veNONE;      // bez akcelerace
            Gradient      := ve26_POINTS; // gradient z 26-ti
                                          // sousedů
            Interpolation := veLINEAR     // lineární interpolace
          end;
        IsoSurf := veMarchingCubes(Volume,MCParams,ve_NIL,128);
        // otevření okna pro vykreslení
        . . .
        // vytvoření render kontextu a nastavení OpenGL pro
        // kreslení, aktivování render kontextu
        . . .
        // převod souřadných systémů
        veCorrectOrientation(Volume,IsoSurf);
        // vlastní vykreslení izoplochy
        veDrawModel(Volume,IsoSurface);
        veDestroyIsoSurface(IsoSurf);
        veDestroyVolume(Volume)
      end;
    veDestroyFileInfo(FileInfo)
  end;
```

4.5.2 Způsoby výpočtu gradientu

Byly implementovány čtyři způsoby odhadu normály povrchu :

- centrální diferenciální metoda (viz. Kap. 3.1)
- adaptivní výpočet gradientu (viz. Kap. 3.1)
- výpočet gradientu z 18-ti sousedních vzorků
- výpočet gradientu z 26-ti sousedních vzorků

4.5.3 Implementace algoritmu Marching Cubes

Algoritmus Marching Cubes byl implementován v původní podobě s několika modifikacemi přispívajícími k jeho zrychlení.

První možností urychlení je eliminovat potřebu vícenásobného výpočtu průsečíku izoplochy a hrany objemového elementu (krychle). Při určitém pořadí zpracování krychlí je nutné v následující krychli vypočítat průsečíky pouze na třech hranách z celkového počtu dvanácti hran, pokud vůbec průsečík na těchto hranách existuje. První vrstva buněk, první řádek buněk v každé vrstvě a první buňka v řádku pak musí být zpracovány jako speciální případ, kdy je nutno počítat průsečíky na všech dvanácti hranách.

Takže pokud budeme jednotlivé krychle postupně zpracovávat zleva doprava, shora dolů a zepředu dozadu můžeme počítat průsečíky pouze na třech hranách krychle a zbylé průsečíky získat z nějaké datové struktury, do které byly dříve vypočtené hodnoty uloženy. Vzhledem k tomu, že jednu vrstvu krychlí tvoří dva po sobě jdoucí řezy, potřebujeme pro tyto dva řezy alokovat potřebné množství paměti pro uložení již vypočítaných průsečíků. Požadovanou datovou strukturu lze tedy implementovat pomocí dvourozměrného pole, jehož počet řádků a sloupců odpovídá počtu vzorků v osách x a y.

Jak již bylo uvedeno v kap. 3.2.1.1 musí algoritmus projít všechny buňky objemu. Uvážíme-li, že počet buněk které hledaná izoplocha protíná, tj. buněk v nichž budeme počítat průsečíky s izoplochou a generovat polygony, tvoří pouze menší část celého objemu, můžeme algoritmus urychlit tím, že celý objem rozdělíme na stejně velké části a pro všechny tyto části vypočteme minimální a maximální hodnotu vzorku.

Tyto minima a maxima včetně informace o příslušné části objemu uložíme do paměti. Poté můžeme snadno detekovat ty části objemu, kde existují průsečíky s jednotlivými objemovými elementy a zároveň rychle přeskočit ty části objemu, kde tyto průsečíky určitě nejsou. Nazvěme tento způsob urychlení akcelerací pomocí „min-max struktury".

Je vidět, že urychlení algoritmu výše uvedeným způsobem závisí na velikosti dílčího objemu.

4.5.4 Implementace algoritmu Marching Tetrahedra

Algoritmus Marching Tetrahedra byl implementován pro tři různá dělení krychle na tetrahedrony, tj. na pět, šest a dvanáct tetrahedronů (viz. kap. 3.2.2).

I pro tento algoritmus můžeme použít místo lineární interpolace při výpočtu průsečíku s hranou objemového elementu (tetrahedronu) střed úsečky. Tím docílíme snížení počtu matematických operací a zároveň i počtu generovaných polygonů. Ve všech tetrahedronech, v nichž byla izoplocha aproximována pomocí dvou trojúhelníků, můžeme tyto trojúhelníky reprezentovat jako čtyřúhelník, protože všechny čtyři vrcholy leží v jedné rovině za předpokladu, že vstupní data tvoří pravidelnou mřížku. Důkaz tohoto tvrzení je triviální. Je nutné dokázat, že pro všechny tetrahedrony, vzniklé dělením krychle, na jejichž hranách existují čtyři průsečíky, leží tyto průsečíky v jedné rovině. Důkaz byl proveden pro všechna výše uvedená dělení krychle na tetrahedrony a bylo zjištěno, že výše uvedené tvrzení platí.

Algoritmus Marching Tetrahedra lze dále zrychlit pomocí „min-max struktury" jako v případě algoritmu Marching Cubes.

5 Výsledky

Po implementaci algoritmů byly tyto podrobeny testování. Byly sledovány veličiny : čas generování izoplochy všemi algoritmy, počet polygonů vygenerované sítě pro různé algoritmy, vliv zvolené metody výpočtu gradientu na dobu generování povrchu, vliv způsobu výpočtu průsečíků hran objemových primitiv s izoplochou na dobu generování povrchu a urychlení algoritmů použitím „min-max struktury“ v závislosti na velikosti dílčího objemu.

Všechny časy byly měřeny pomocí vysokofrekvenčního čítače obsaženého v systému. Tento čítač umožňuje velmi přesné měření, jeho frekvence je odvozena od taktovací frekvence procesoru. Windows API nabízí dvě funkce pro práci s tímto čítačem :

- QueryPerformanceFrequency
- QueryPerformanceCounter

Pomocí funkce *QueryPerformanceFrequency* lze zjistit frekvenci čítače v Hz a voláním *QueryPerformanceCounter* aktuální hodnotu čítače. Dobu trvání nějaké operace pak lze měřit tak, že těsně před začátkem a ihned po skončení této operace zjistíme hodnotu čítače. Tyto dvě hodnoty odečteme, vydělíme dříve zjištěnou frekvencí a dostaneme údaj o čase v sekundách.

Algoritmy byly testovány na pracovní stanici Intergraph GL2 s procesorem Intel Pentium II 400 MHz, grafickou kartou RealiZm II a 512 MB operační paměti.

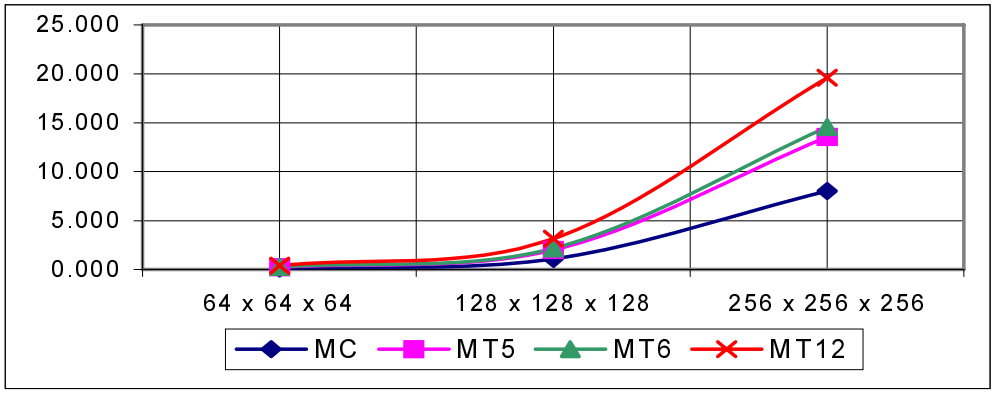
5.1 Závislost doby generování izoplochy na objemu dat

Závislost doby generování izoplochy na objemu dat byla sledována pro všechny implementované algoritmy : Marching Cubes (MC), Marching Tertrahedra s dělením krychle na 5,6 a 12 tetrahedronů (MT5,MT6 a MT12). Pro test byl vybrán vstupní soubor *bentum.vol* s rozlišením 256×256×256 vzorků. Tato data pak dále byla převzorkována na rozlišení 128×128×128 a 64×64×64 vzorků. Doba generování izoplochy byla sledována pro akceleraci pomocí „min-max struktury“ s velikostí dílčího objemu 4. V zájmu zachování objektivity byly všechny algoritmy spuštěny se stejným způsobem

výpočtu gradientu (výpočet z 26-ti sousedních hodnot), shodným způsobem výpočtu průsečíku izoplocha – hrana (lineární interpolace) a stejnou konstantou udávající izoplochu. Tabulka 1 ukazuje naměřené hodnoty v sekundách.

	64×64×64	128×128×128	256×256×256
MC	0,110	1,083	8,007
MT5	0,225	1,986	13,549
MT6	0,253	2,185	14,570
MT12	0,390	3,141	19,611

Tabulka 1 – Závislost doby generování izoplochy na objemu vstupních dat



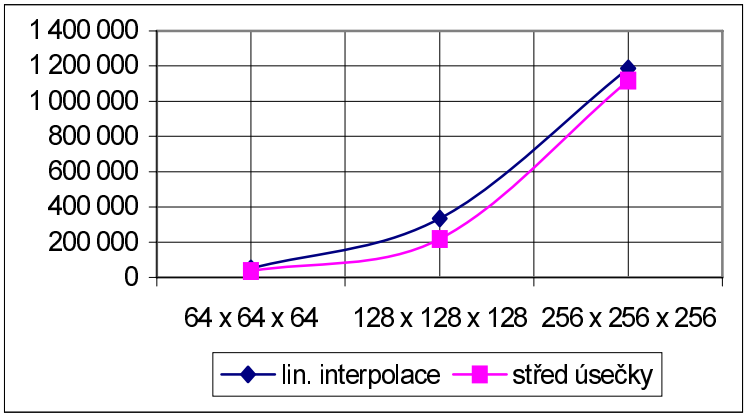
Graf 1 – Závislost doby generování izoplochy na objemu vstupních dat

5.2 Počet polygonů izoplochy

Počet vygenerovaných polygonů aproximujících izoplochu je základním parametrem, na kterém přímo závisí rychlost zobrazení dané izoplochy. V Tabulce 2 jsou uvedeny počty polygonů vygenerované algoritmem MC v závislosti na objemu dat pro soubor *bentum.vol*. Počet polygonů byl sledován pro oba implementované způsoby výpočtu průsečíku izoplochy s hranou a samozřejmě pro stejnou izoplochu.

	64×64×64	128×128×128	256×256×256
lin. interpolace	50 097	333 388	1 185 698
střed úsečky	34 417	217 708	1 117 865

Tabulka 2 Závislost počtu generovaných polygonů na objemu dat pro algoritmus MC



Graf 2 Závislost počtu generovaných polygonů na objemu dat pro algoritmus MC

Porovnání počtu generovaných polygonů různými algoritmy při konstrukci stejné izoplochy ze souboru *bentum.vo*l s rozlišením 256×256×256 ukazuje Tabulka 3. Třetí sloupec ukazuje poměr mezi počtem polygonů generovaných při použití středu úsečky k počtu polygonů generovaných při použití lineární interpolace pro výpočet průsečíků izoplochy s hranami.

	lin. interpolace	**střed úsečky**	**K_{li} / $K_{sú}$**
MC	1 185 698	1 117 865	94,28 %
MT 5	4 446 122	3 353 161	75,42 %
MT 6	5 552 376	4 426 876	79,73 %
MT 12	8 877 336	6 682 154	75,27 %

Tabulka 3 Počty polygonů generovaných různými algoritmy

Z Tabulky 3 je jasně vidět, že nejméně polygonů generuje algoritmus Marching Cubes.

5.3 Vliv metody výpočtu gradientu na dobu výpočtu

Při testování algoritmů se ukázalo, že ačkoliv na výpočet gradientu z 26-ti sousedů je zapotřebí více jak 2 krát tolik operací v pohyblivé řádové čárce než pro výpočet gradientu pomocí centrální diferenciální metody, nemá výběr metody příliš velký vliv na dobu potřebnou pro generování izoplochy. Tabulka 4 ukazuje doby generování izo-

plochy dané konstantou k=75 ze souboru *bentum.vol* pro všechny implementované metody výpočtu gradientu.

Metoda výpočtu gradientu	Čas generování sítě [s]
Centrální diference	7,583
Adaptivní výpočet	7,887
Výpočet z 18-ti sousedů	7,885
Výpočet z 26-ti sousedů	8,007

Tabulka 4 Vliv metody výpočtu gradientu na čas potřebný pro generování izoplochy

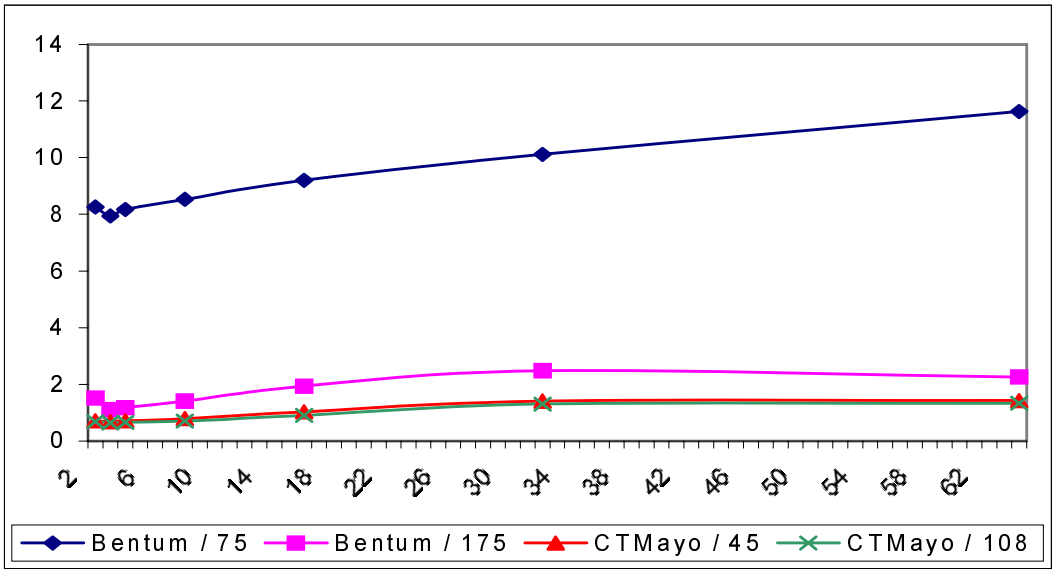
Předchozí tabulka ukazuje, že při použití výpočetně nejnáročnější metody gradientu dojde ke zpomalení o cca 5.5 % ve srovnání s centrální diferenciální metodou. Přitom však dojde k výraznému vyhlazení povrchu ve výsledném obrazu.

5.4 Urychlení pomocí „min-max struktury“

V kapitole 4.5.3 bylo uvedeno, že urychlení pomocí „min-max struktury“ je závislé na volbě velikosti dílčího objemu. V Tabulce 4 jsou uvedeny doby generování výpočtu pro různou velikost dílčího objemu, pro dva různé soubory *bentum.vol* s rozlišením 256×256×256 a *ctmayo.vol* s rozlišením 128×128×128. Pro oba soubory byla navíc testována závislost pro dvě různé izoplochy. V předposledním sloupci tabulky jsou napsány doby generování izoplochy bez použití „min-max struktury“ a v posledním sloupci je uvedeno maximální zrychlení oproti neakcelerované verzi algoritmu.

	2	3	4	8	16	32	64	X	Akc.
Bentum / 75	8,26	7,94	8,17	8,53	9,20	10,12	11,63	12,18	1,53
Bentum / 175	1,51	1,12	1,18	1,41	1,94	2,48	2,26	8,99	8,03
CTMayo / 45	0,71	0,68	0,71	0,79	1,03	1,41	1,43	1,43	2,10
CTMayo / 108	0,68	0,64	0,66	0,70	0,90	1,30	1,34	1,267	1,98

Tabulka 5 Závislost doby generování izoplochy na velikosti dílčího objemu



Graf 3 Závislost doby generování izoplochy na velikosti dílčího objemu

Jako optimální velikost dílčího objemu pro akceleraci pomocí „min-max struktury“ lze zvolit hodnotu 3. Při této velikosti dílčího objemu dochází k největšímu zrychlení.

6 Závěr

Implementace algoritmů Marching Cubes a Marching Tetrahedra na platformě PC ukazuje, že je možné v současné době použít tento typ počítače k interaktivní práci s volumetrickými daty do rozlišení 128×128×128.

Centrální diferenciální metoda se ukázala jako špatný odhad normály povrchu ve srovnání s výpočtem z 26-ti sousedů z hlediska kvality výsledného obrazu. Při současném výpočetním výkonu osobních počítačů je možné použít složitější výpočet gradientu s minimálním zpomalením výpočtu (řádově o 5 %) v porovnání s centrální diferenciální metodou.

Jako účinná metoda snížení počtu generovaných polygonů aproximujících izoplochu se ukázalo použití středu úsečky při výpočtu průsečíků izoplochy s hranami objemových primitiv. Tato metoda dokáže snížit počet polygonů až o 50 % a to hlavně pro izoplochy, které obsahují relativně velké rovinné plochy. V průměru se pak redukce počtu generovaných polygonů pohybuje mezi 10 % – 30 %.

Algoritmus Marching Cubes je podle výsledků vhodnější pro praktické použití než algoritmus Marching Tetrahedra, protože generuje menší počet polygonů v kratším čase. Algoritmus Marching Tetrahedra je pak vhodný hlavně pro generování izoplochy pouze z části objemu , kdy existuje předpoklad, že vygenerovaný povrch bude při zobrazení mnohonásobně zvětšen.

Literatura

[Bent96] Bentum, J.M., Barthold B.A, Lichtenbelt a Malzbender, A.: *Frequency Analysis of Gradient Estimators in Volume Rendering.* IEEE Transaction on Visualization and Computer Graphics, Vol. 2, No. 3, 1996

[Chua95] Chuang, J., Lee, W.: *Efficient Generation of Isosurface in Volume Rendering.* Computer and Graphics, Vol. 19, No 6, 805-813, 1995

[Cign97] Cignoni, P., Marino, P., Montani, C., Puppo, E. a Scopigno, R.: *Speeding Up Isosurface Extraction Using Interval Trees.* IEEE Transactions on Visualization and Computer Graphics, Vol. 3, No. 2, 158-170, 1997

[Cline88] Cline, H.E., Lorensen, W.E., Ludke, S.: *Two Algorithms for Three-dimensional Reconstruction of Tomograms*. Medical Physics, Vol. 15, No. 3, 320-327,1988

[Cohe92] Cohen, M.F, Painter, J., Mehta, M. a Ma, K.: Volume Seedlings ACM Siggraph - Computer Graphics Special issue on 1992 Symposium on Interactive 3D Graphics, 139-145, 1992

[Dank92] Dankins, J., Hanrahan, P.: *Fast Algorithms for Volume Ray Tracing.* Workshop on Volume Visualization 1992, 91-98

[Levo88a] Levoy, M.: *Display of Surfaces from Volume Data*. IEEE Computer Graphics and Applications, No. 8,29-37, 1988

[Levo88b] Levoy, M.: *Efficient Ray Tracing of Volume Data*. Technical Report TR88-029 University of North Carolina at Chapel Hill, June 1988

[Levo88c] Levoy M.: *Volume Rendering by Adaptive Refinement*. The Visual Computer, No.6 , 2 – 7, 1990

[Li97] Li, J., Agathoklis, P.: *An Efficiency Enhanced Isosurface Generation Algorithm for Volume Visualization*. The Visual Computer 13:391-400, 1997

[Lore87] Lorensen, W.N., Cline, H.E.: *Marching Cubes : A High Resolution 3D Surface Construction Algorithm.* Computer Graphics 21, 163-169, 1987

[Shek96] Shekhar, R., Fayyad, E., Yagel, R., Cornhill, J.F: *Octree-Based Decimation of Marching Cubes Surfaces*. IEEE Visualization, 1996

[Shu91a] Shu, R., Liu, A. Huang, K. : *An Improved Adaptive Ray Casting Algorithm.* Proceedings IFIP TC5/WG 5.10. Conference on Copmuter Graphics, Tokyo Japan, April 1991

[Shu91b] Shu, R., Liu A.: A *Fast Ray Casting Algorithm Using Adaptive Isotriuangular Subdivision.*, IEEE Visualization'91, San Diego, USA, 232-239, 1991

[Shu92] Shu, R. Chui, Ch.: *An Adaptive Incremental Sampling Approximation to Volume Rendering.* CGI'92, Japan, 273-294, 1992

Literatura

[Tost95] Tost, D., Puig, A., Navazo, I.: *A Volume Visualization Algorithm Using a Coherent Extended Wieght Matrix.* Computer & Graphics, Vol. 19, No. 1, 37-45, 1995

[West89] Westover, L.: *Interactive Volume Rendering*. Chopel Hill Workshop on Volume Visualization, 9-16, May 1989

[West90] Westover, L.: *Footprint Evaluation for Volume Rendering*. Computer Graphics (ACM Siggraph Proceedings) 24(4), 367-376, 1990

[Wilh91] Wilhelms, J., Gelder, A.: *A Coherent Projection Approach for Direct Volume Rendering*. ACM Computer Graphics, 25(4), 285-318, 1991

[Yage92] Yagel, R., Cohen, D. a Kaufman, A.: *Discrete Ray Tracing*. IEEE Computer Graphics & Applications, Vol. 12 No. 5, 19-28, 1992

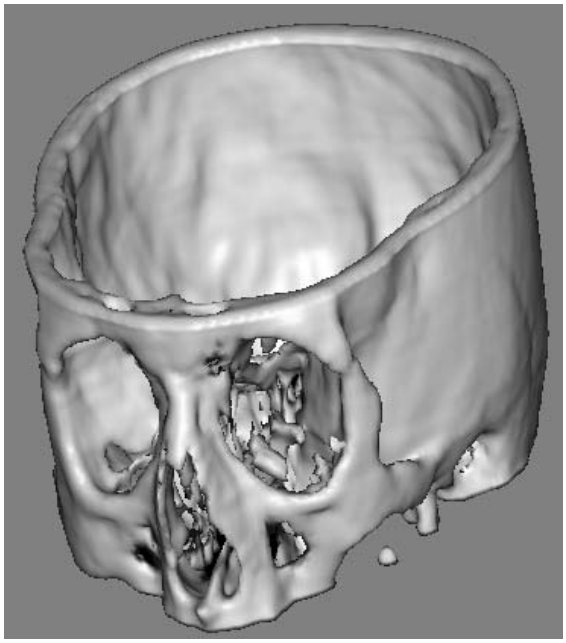
[Zara98] Žára, J., Beneš, B. a Fenkel, P.: *Moderní počítačová grafika.* Computer Press 1998

[Zhou94] Zhou, C., Shu, R., Kankahalli, M.S.: *Handling Small Features In Isosurface Generation Using Marching Cubes*. Computer and Graphics, Vol. 18, No. 6, 845-848, 1994

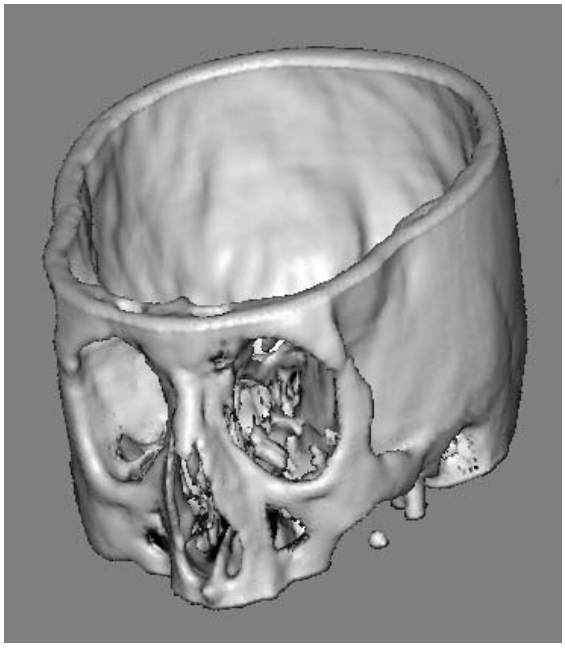
[Zhou95] Zhou, C., Shu, R. Kankanhalli, M.S: *Selectively Meshed Surface Representation.* Computer & Graphics, Vol. 19, No 6, 793-804, 1995

Příloha A – Obrazová příloha

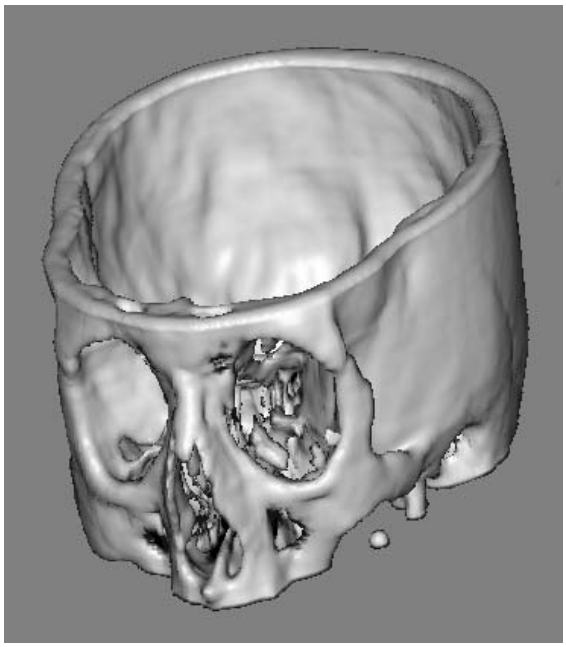
Povrch generovaný různými algoritmy



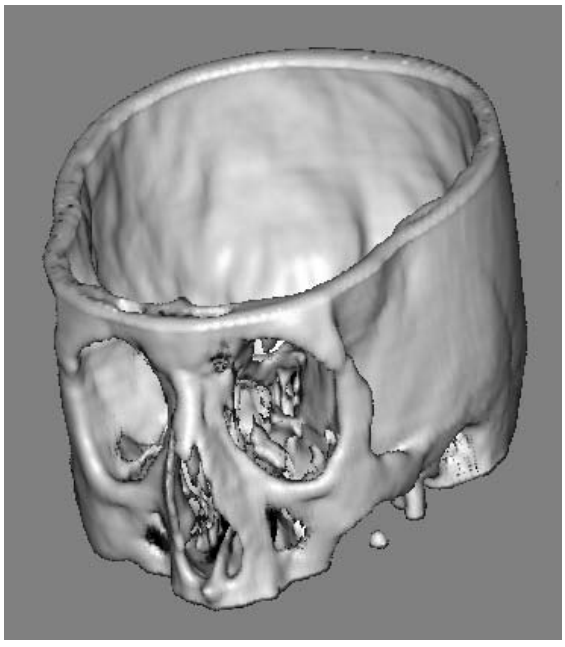
MC, čas generování izoplochy 0.65 s, vrcholů 93 395, polygonů 185 535



MT5, čas generování izoplochy 1.19 s, vrcholů 230 015, polygonů 460 068

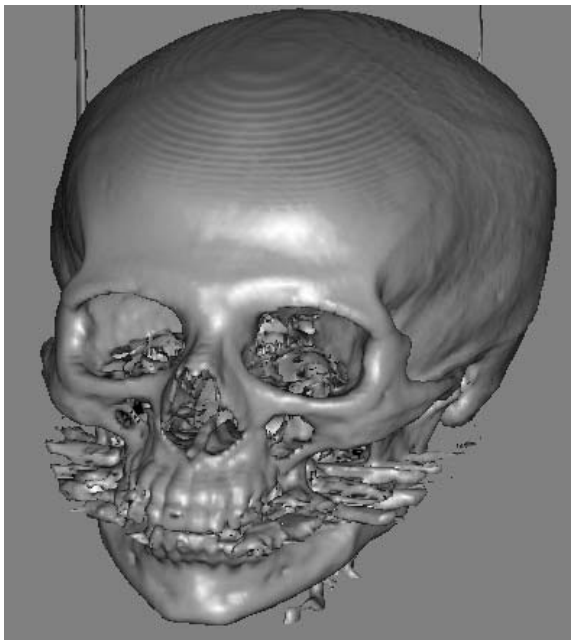


MT6, čas generování izoplochy 1.31 s, vrcholů 288 866, polygonů 577 778

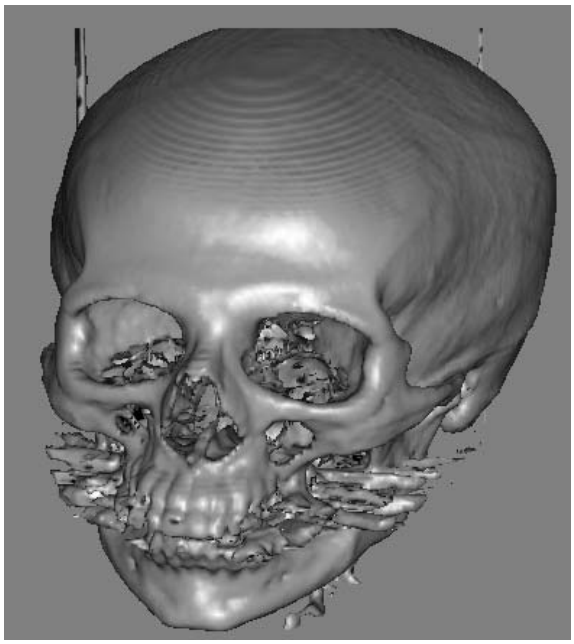


MT12, čas generování izoplochy 2.04 s, vrcholů 468 850, polygonů 937 761

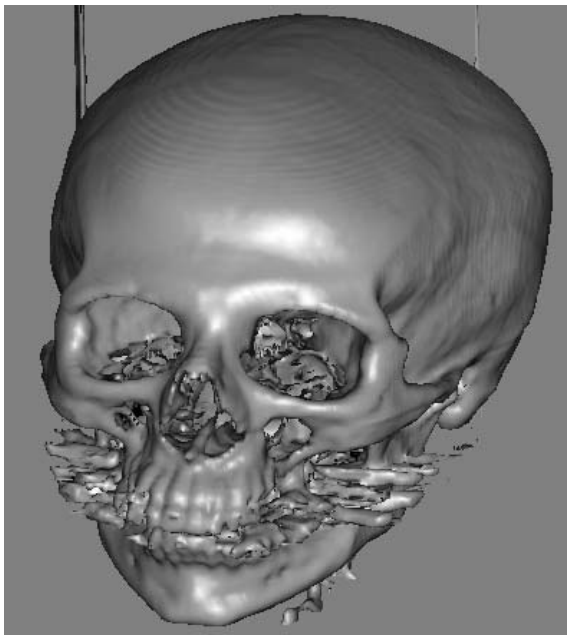
Porovnání metod pro výpočet gradientu



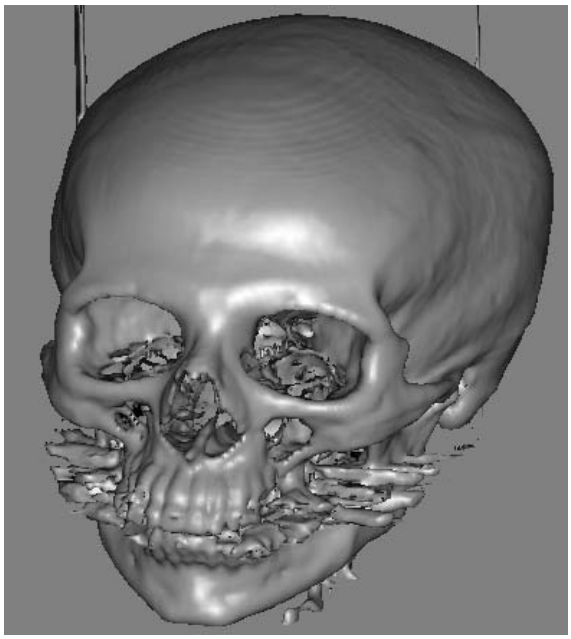
CTHead 256x256x113, MC, výpočet gradientu centrální diferenciální metodou



CTHead 256x256x113, MC, výpočet gradientu adaptivní metodou

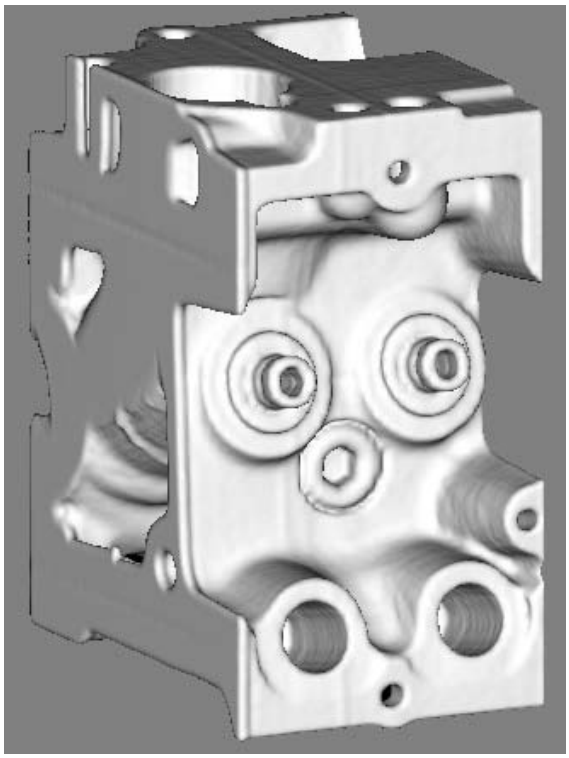


CTHead 256x256x113, MC, výpočet gradientu z 18-ti sousedů

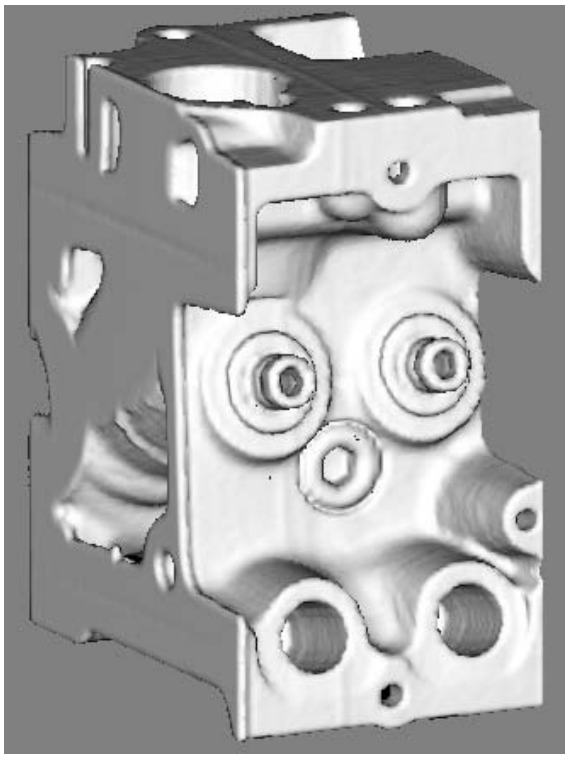


CTHead 256x256x113, MC, výpočet gradientu z 26-ti sousedů

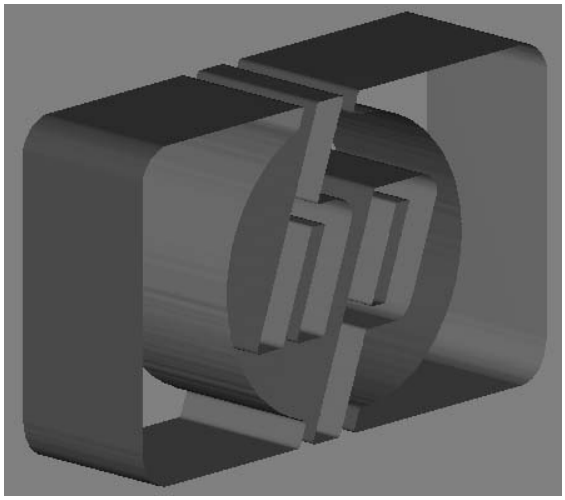
Lineární interpolace a střed úsečky



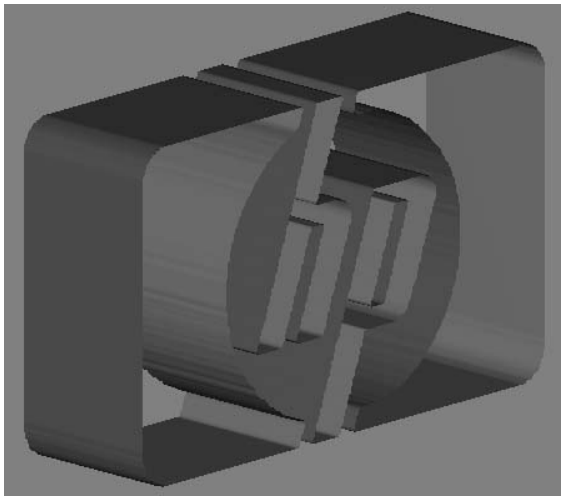
Engine 256x256x110, MC,výpočet průsečíků pomocí lineární interpolace, čas výpočtu 3,19 s, vrcholů 299 907, polygonů 593 398



Engine 256x256x110, MC,výpočet průsečíků jako střed úsečky, čas výpočtu 3,13 s, vrcholů 299 907, polygonů 334 496

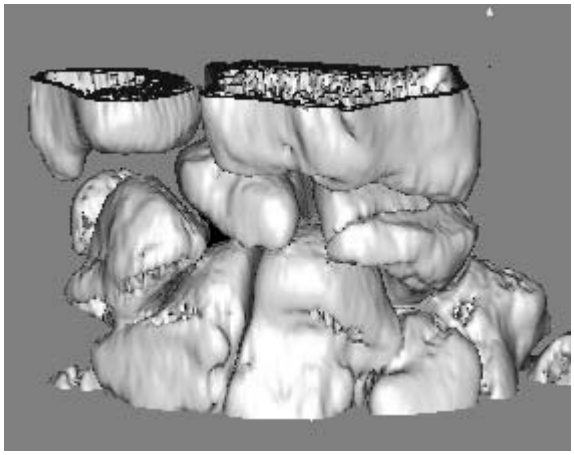


HpLogo 236x150x64, MC, výpočet průsečíků pomocí lineární interpolace, čas výpočtu 1.15 s, vrcholů 20 028, polygonů 236 668

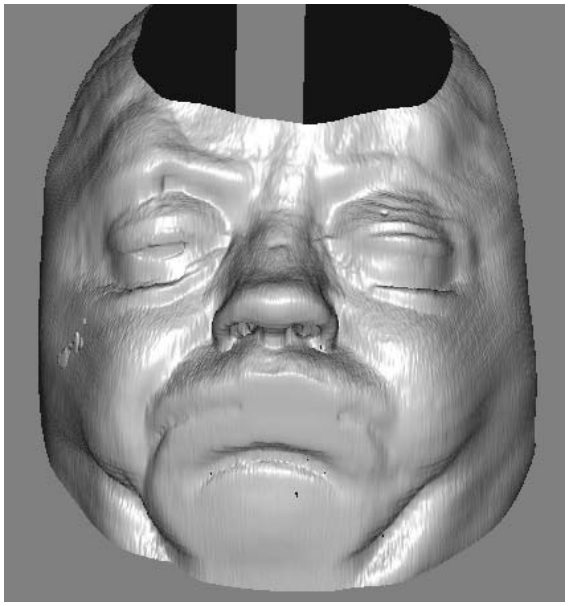


HpLogo 236x150x64, MC, výpočet průsečíků jako střed úsečky, čas výpočtu 1.11 s, vrcholů 20 028, polygonů 118 834

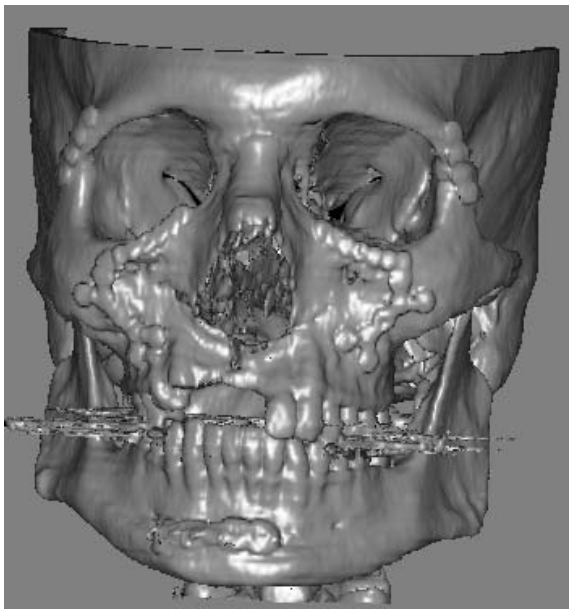
Ukázky rekonstruovaných povrchů



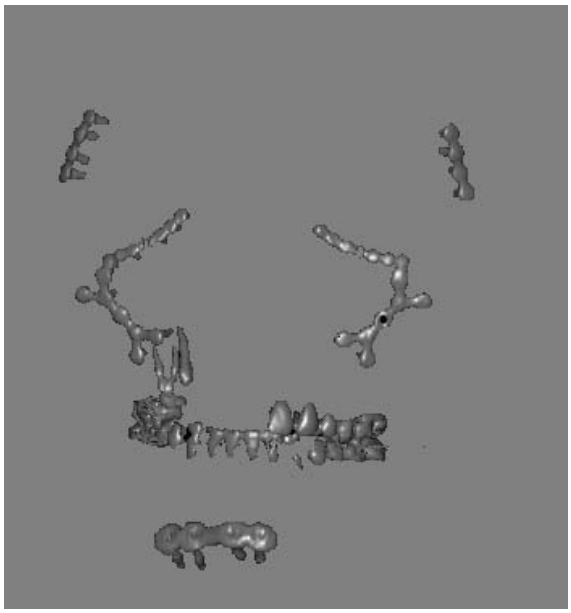
Rozlišení 512x512x45, čas generování izoplochy 3.65 s, vrcholů 177 717, polygonů 344 465



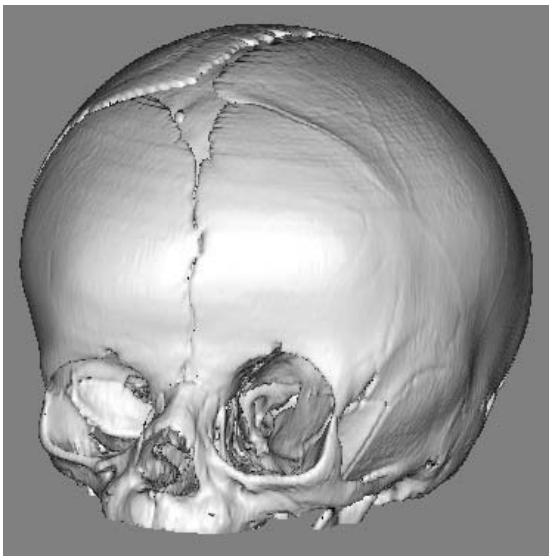
Rozlišení 512x512x48, čas generování izoplochy 5.01 s, vrcholů 415 548, polygonů 826 647



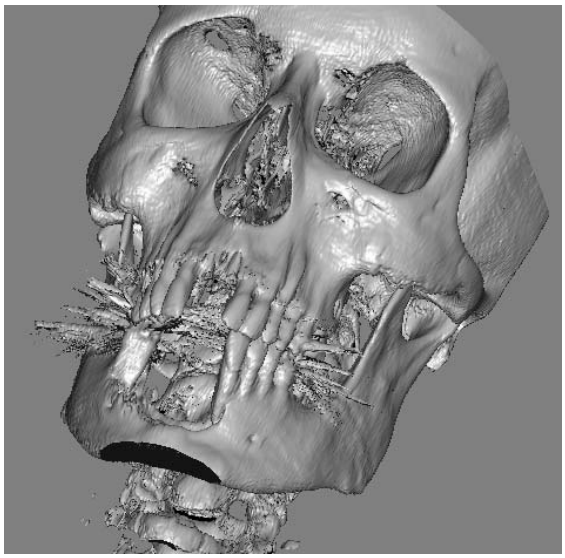
Rozlišení 512x512x152, čas generování izoplochy 15.35 s, vrcholů 1 120 208, polygonů 2 238 239



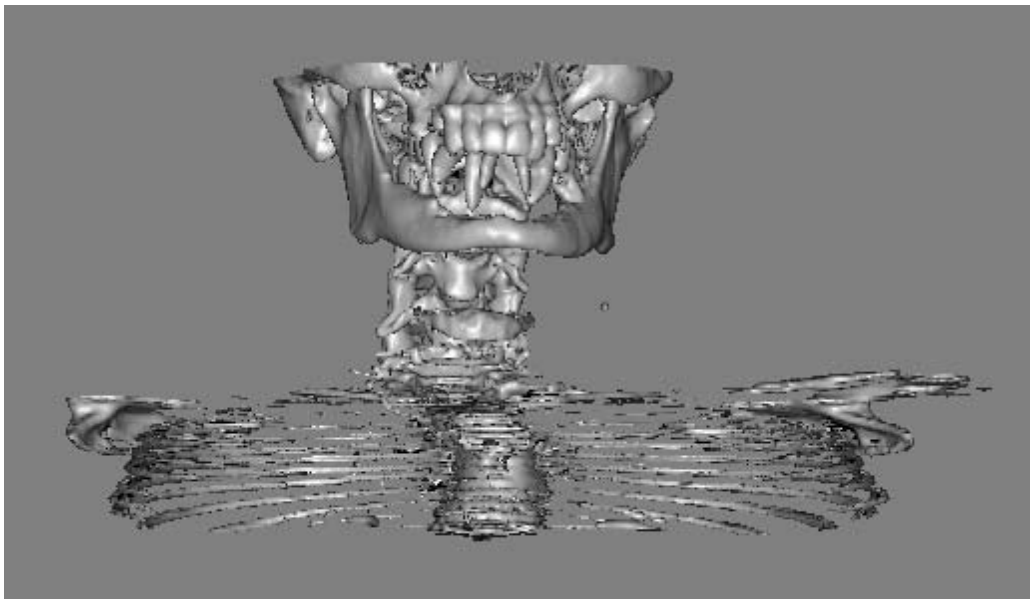
Rozlišení 512x512x152, čas generování izoplochy 8.02 s, vrcholů 50 093, polygonů 101 127. Stejný soubor jako na obrázku vlevo, pouze jiná izoplocha.



Rozlišení 512x512x112, čas generování izoplochy 10.15 s, vrcholů 630 080, polygonů 1 124 904



Rozlišení 512x512x137, čas generování izoplochy 16.92 s, vrcholů 1 181 619, polygonů 3 359 146



Rozlišení 512x512x117, čas generování izoplochy 9.17, vrcholů 333 953, polygonů 666 619

Příloha B – Uživatelská příručka

Instalace

Instalaci programu, zdrojových textů knihovny a ukázkové aplikace lze snadno nainstalovat spuštěním programu *Setup.exe* v adresáři *Install* na přiloženém CD. Po spuštění je uživatel vyzván k zadání cílového adresáře. Ostatní akce spojené s instalací proběhnou automaticky.

Odinstalovat program z počítače lze pomocí funkce *Přidat nebo Ubrat programy* v okně *Nastavení/Ovládací panely* systému Windows.

Po instalaci lze program spustit výběrem položky *Volume Explorer* z nabídky *Start/Programy/Volume Explorer* na hlavním panelu.

Načtení souboru a generování povrchu

Načtení požadovaného souboru se provede výběrem položky *Soubor/Otevřít* z hlavního menu aplikace. Po načtení souboru je možné vygenerovat a zobrazit konkrétní izoplochu vybraným algoritmem. Algoritmus Marching Cubes se spouští pomocí menu *Marching Cubes/Spustit* a analogicky algoritmus Marching Tetrahedra položkou menu *Marching Tetrahedra/Spustit*.

Po výběru algoritmu je nutné zadat konstantu určující izoplochu v dialogovém okně *Určení izo-plochy*, které se otevře po výběru algoritmu. Poté následuje výpočet a otevření okna pro 3D zobrazení povrchového modelu. Současně se otevřou informační okna *Poloha*, *Povrch* a *Doba výpočtu* a okno pro změnu konstanty určující izoplochu *Určení izo-plochy*.

Nastavení parametrů

V programu je možné provést nastavení parametrů algoritmu Marching Cubes a Marching Tetrahedra.

Nastavení parametrů algoritmu Marching Cubes je možné v dialogovém okně *Parametry MC*, které lze otevřít vybráním položky menu *Marching Cubes/Parametry*. V tomto dialogu lze nastavit způsob výpočtu gradientu, způsob výpočtu průsečíku izo-

plochy s hranami objemových primitiv a možnost použití akcelerační struktury „min-max“.

Analogicky se nastavují parametry Marching Tetrahedra. V menu vybrat položku *Marching Tetrahedra/Parametry* a v dialogovém okně *Parametry MT* nastavit požadované parametry.

Protože zobrazování povrchové modelu pomocí display listu může být paměťově náročné, je implementováno i zobrazování bez použití display listů. Tato vlastnost se nastaví v dialogovém okně *Open GL* aktivovaném z menu *Nastavení/Open GL*. Tato položka je přístupná pouze tehdy, není-li otevřen žádný soubor.

Okno 3D zobrazení

V okně pro 3D zobrazení je možné s objektem rotovat kolem všech tří os souřadného systému pomocí myši. Při stisku levého tlačítka myši a posunu kurzoru (tlačítko je stále stisknuto) je možné rotovat kolem osy x a y (pohyb myši vertikálně a horizontálně). Pomocí pravého tlačítka pak lze rotovat s objektem kolem osy z.

Zobrazený objekt lze zvětšit a zmenšit pomocí kláves „+“ a „-“ na numerické klávesnici.

Pokud je aktivní právě okno 3D zobrazení povrchu, přibude v hlavním menu položka *Render*. V rozbalené nabídce této položky jsou pak přístupné funkce pro kopírování scény do schránky a uložení do souboru ve formě rastrového obrazu.

Příloha C – Formát souboru VIF

Volume Info File (VIF) soubor je textový soubor podobný strukturou INI souborům Windows. Obsahuje 5 sekcí. Začátek sekce je dán jménem sekce uzavřeným do hranatých závorek. V každé sekci je definováno několik proměnných.

Sekce souboru VIF

VIF soubor obsahuje následující sekce :

- [Main] – základní informace o formátu dat
- [Resolution] – informace o počtu vzorků v jednotlivých osách
- [Correction - position] – korekce pozice
- [Correction - rotation] – korekce polohy
- [Distortion] – distorze

Sekce - [Main]

V sekci *Main* jsou obsaženy všechny nutné informace o vstupním souboru. Následuje seznam proměnných. Pokud není uvedeno jinak je zadání hodnoty proměnné povinné.

Proměnné :

- Caption – alfanumerická hodnota, uživatelem definovaný popis souboru, nepovinná položka.
- Source – alfanumerická hodnota, název zdrojového zařízení (skeneru), nepovinná položka.
- Header size – celočíselná hodnota, určuje velikost hlavičky v bytech vstupního souboru, implicitně „0“.
- Bytes per voxel – celočíselná hodnota, udává počet bytů, na kterém je uložena jedna hodnota vzorku, implicitně „1“.
- Signed – logická hodnota zadaná ve tvaru „yes“ nebo „no“, určuje, zda hodnoty vzorků jsou uloženy znaménkovém, implicitně „no“.

- Byte order – udává, zda dvou bytové vzorky jsou v souboru uloženy podle konvence firmy Intel nebo opačně. Přípustné hodnoty jsou „file“ a „reverse“. Pokud je zadána hodnota „file“ data jsou načtena bez úprav. Pro hodnotu „reverse“ jsou byty 16-ti bitových vzorků zaměněny. Nastavení této proměnné má smysl pouze tehdy, je-li hodnota proměnně *Bytes per voxel*= 2, implicitně „file“.
- Mask bits – celočíselná hodnota, která udává bity pro maskování po načtení do paměti. Pokud např. vstupní soubor obsahuje 16-bitové vzorky, dolní byte obsahuje vlastní hodnotu vzorku a horní byte obsahuje nějakou další informaci je nutné vymaskovat všechny bity horního bytu na nulu. Hodnotu lze zadat jak v dekadické tak v hexadecimální podobě.

Sekce - [Resolution]

Sekce *Resolution* obsahuje informace o rozlišení vstupních dat. Hodnoty všech proměnných této sekce je nutné vyplnit. Implicitní hodnota pro všechny proměnné je „0“. Hodnoty všech proměnných musí být celé kladné číslo.

Proměnné :

- x – počet vzorků v ose **x**
- y – počet vzorků v ose **y**
- z – počet vzorků v ose **z**

Sekce - [Correction - position]

Pomocí proměnných této sekce lze provést korekci pozice objektů obsažených v souboru relativně k ohraničujícímu boxu. Položky této sekce jsou nepovinné. Implicitní hodnota všech proměnných je „0“. Hodnoty všech proměnných musí být celé číslo.

Proměnné :

- x – posunutí objektu v ose **x**
- y – posunutí objektu v ose **y**
- z – posunutí objektu v ose **z**

Sekce - [Correction - rotation]

Pomocí proměnných této sekce lze provést korekci polohy objektů. Položky této sekce jsou nepovinné. Implicitní hodnota všech proměnných je „0“. Hodnoty všech proměnných musí být celé číslo.

Proměnné :

- x – rotace objektu kolem osy **x** ve stupních
- y – rotace objektu kolem osy **y** ve stupních
- z – rotace objektu kolem osy **z** ve stupních

Sekce - [Distortion]

Sekce *Distortion* obsahuje informaci o distorzi vstupních dat. Položky této sekce jsou nepovinné. Implicitní hodnota všech proměnných je „1.0“. Hodnoty všech proměnných musí desetinné číslo. Tvar čísla v tzv. vědeckém formátu není přípustný.

Proměnné :

- x – distorze ve směru osy **x**
- y – distorze ve směru osy **y**
- z – distorze ve směru osy **z**

Příklad souboru VIF

```
[Main]
Caption =
Source =
Header size = 38
Bytes per voxel = 1
Signed = no
Byte order = file

[Resolution]
x = 64
y = 64
z = 64
```

```
[Correction - position]
x = 0
y = 0
z = 0

[Correction - rotation]
x = 90
y = 0
z = 0

[Distortion]
x = 1.0
y = 1.0
z = 1.0
```

Příloha D – Obsah CD

Na přiloženém CD jsou následující adresáře :

- *DATA* – datové soubory, které byly použity při testování algoritmů. Tato data byla získána z renomovaných univerzitních pracovišť.
- *INSTALL* – instalační program pro instalaci aplikace Volume Explorer a zdrojového kódu na pevný disk počítače
- *PAPERS* – některé články citované v této práci v elektronické podobě získané na Internetu
    - *RUNIMAGE* – dekomprimované soubory aplikace Volume Explorer a zdrojový kód