1 Obsah

2 Úvod

Volumetrická data obsahují informace o objektu či o jeho části, které nejsou pouhým lidským okem pozorovatelné. Lépe řečeno, obsahují informace o struktuře zkoumaného objektu z pohledu nějaké fyzikální vlastnosti (např. jeho tvrdosti). Jsou používány hlavně v lékařství, kde je například pro správnou diagnózu potřeba vidět do útrob člověka. Podobně ve strojírenství je třeba prozkoumat kvalitu výrobku nejen z vnějších – pouhým okem viditelných částí, ale je také třeba prozkoumat strukturu materiálu, ze kterého je součástka vyrobena. Například je možné rozeznat bublinky v kovové součástce, které vznikají při výrobě litím materiálu do forem. Ty pak způsobují nepředvídatelné vlastnosti výrobku.

Pro zobrazování volumetrických dat existuje v současné době mnoho metod. Jedná se zhruba o tři typy základních přístupů:

- Zobrazování 2D řezů – jednoduchý primární Ray Tracing
- Zobrazování 3D povrchů – Marching Cubes, Marching Tetrahedra (například pro 5, 6 dělení)
- Zobrazování 3D objemů – Ray Tracing

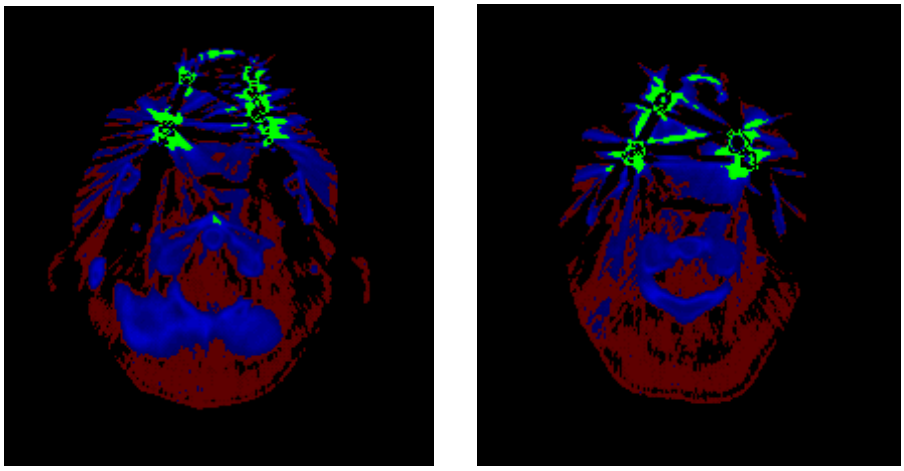
Každá z těchto metod v sobě skrývá výhody i nevýhody. Zobrazování rovinných řezů je výpočetně nejméně složité a také v současné době asi nejpoužívanější v oblasti lékařství. Poskytují však informace pouze o určitém řezu objektu. Metody, které zobrazují 2D řezy objektu jsou poměrně jednoduché, výpočetně nenáročné a zajišťují nezkreslení objektu na výstupu algoritmu. Další metody se již zabývají rekonstrukcí objektu ve 3D. To sebou samozřejmě nese vyšší nároky při výpočtu, zobrazování, ale i jistou míru zkreslení – chyby na výstupu.

Tato práce se zabývá především problematikou generování a zobrazování 3D povrchových modelů z volumetrických dat a též možnostmi implementace v Projektu MVE na ZČU: Computer graphics and data visualization in distributed and parallel environment, Project VS 97 155.

3 Získávání volumetrických dat

Volumetrická data mohou být získávána různými způsoby. Většinou se používají 3D snímače, které vysílají z různých stran na objekt rentgenové paprsky s určitou intenzitou a podle vráceného (odraženého) paprsku detekují, jakou vlastnost má materiál o který se daný paprsek odrazil. Počítač dále sestaví trojrozměrnou mřížku s naměřenými hodnotami.

Někdy lze pozorovat určité chyby v naměřených datech pomocí této metody. Chyby mohou způsobit například materiály, které špatně odráží rentgenové paprsky – například olověné plomby v zubech na následujících obrázcích :



Obrázek 1: Dolní a horní čelist. Zelenou barvou jsou zde vyznačeny chybné odrazy paprsku od zubních plomb pacienta

Rozeznáváme data získaná například pomocí počítačového tomografu (CT – Computed Tomography, SPECT – Single Photon Computed Tomography) či magnetické rezonance (MR – Magnetic Resonance).

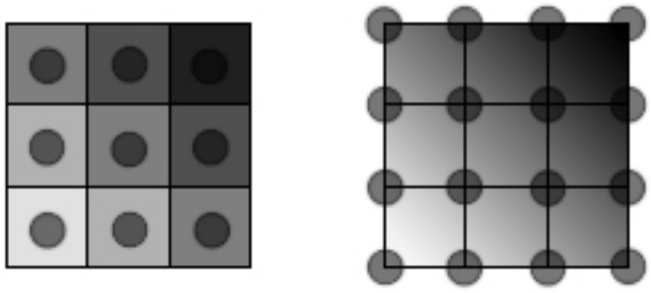
Dále mohou být získaná data v homogenní mřížce – intervaly mezi jednotlivými řezy jsou stejnoměrné v jednotlivých osách. Data snímaná v nehomogenní mřížce mohou být v některých místech snímána hustěji, než v jiných. To lze využít pro lepší zobrazení místa v datech, které je ve středu našeho zájmu. V praxi se většinou setkáváme s daty, které mají x,y souřadnici stejnou (rozlišení snímacího zařízení) a z souřadnice se liší (počet sejmutých řezů).

Algoritmy dále popsané předpokládají homogenní mřížku, ale po menších úpravách je lze použít i pro nehomogenní mřížku.

4 Generování izoploch z volumetrických dat

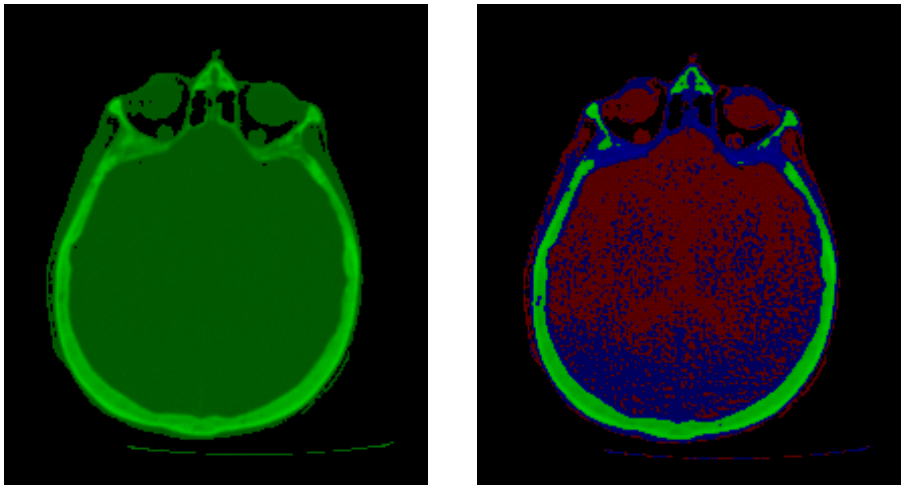
4.1 Použité termíny

- *Volumetrická data* – zde budeme předpokládat data ve tvaru trojrozměrné mřížky h = f (x,y,z), kde h je hodnota reprezentující nějakou vlastnost materiálu o daných souřadnicích. Může to být tvrdost, rychlost proudění, barva, a další.
- *Voxel* – základní element 3D mřížky. Můžeme si jej představit jako krychličku (pro nestejnoměrně dělené osy kvádr) s konstantní hodnotou v celém svém objemu. Tato hodnota má obvykle nějaký fyzikální význam. Můžeme říct, že voxel je něco podobného jako pixel ve dvourozměrném prostoru (x,y). Souřadnice voxelu ve 3D prostoru jsou (x,y,z).
- *Cell* – kvádr, v jehož vrcholech jsou navzorkované hodnoty. Tento kvádr na rozdíl od voxelu nemá v celém svém objemu konstantní hodnotu. Hodnota v libovolném místě uvnitř cell se vypočítává s pomocí tri-lineární interpolace, nebo pomocí interpolace vyššího řádu.



Obrázek 2: Rozdíl mezi Voxel vlevo a Cell vpravo

- *Slice* – řez volumetrickými daty v ose x, y nebo z. Je to množina vzorků v pravidelné mřížce, které mají shodnou souřadnici x, y nebo z.



Obrázek 3: Příklady řezů (slice) generovaných z volumetrických dat (CTHead.vol). Barevný obrázek umožňuje více zviditelnit rozdílnost různých materiálů.

4.2 Metody generování izoploch

Metod pro generování iso-ploch z volumetrických dat je celá řada. Základní metody jsou Marching Cubes a Merching Tetrahedra. Většina ostatních metod vznikla pouze upravením těchto základních metod či použitím speciálních kritérií pro odhad průběhu izoplochy v rámci voxelu.

V následujícím textu budou popsány dvě nejpoužívanější metody generování izoploch v podobě trojúhelníkových sítí z volumetrických dat. Jedná se o metody Marching Cubes a Marching Tetrahedra.

4.3 Marching Cubes

Marching Cubes je jednoduchá a velice populární metoda pro generování izoploch z diskrétních tří-rozměrných (3D) dat. Vstupem algoritmu je prostorová mřížka, v jejíž vrcholech jsou uloženy hodnoty vzorků. Jinak řečeno mřížka sestávající se z množiny krychliček - cells. Postupně jsou zpracovávány vždy dva řezy (slices) mřížky jdoucí po sobě. Algoritmus lze shrnout do těchto základních bodů:

1. Nalezení krychle a určení vrcholových hodnot
2. Kontrola těchto hodnot se zadanou prahovou hodnotou
3. Výpočet normál ve vrcholových bodech krychle
4. Vytvoření indexu do tabulky možných tvarů trojúhelníkové sítě v rámci krychle
5. Určení hran na kterých leží vrcholy trojúhelníků
6. Výpočet přesné pozice vrcholů trojúhelníků v rámci hrany krychle
7. Interpolace normál vrcholů trojúhelníků v závislosti na vrcholových normálách krychle

ad 1) Krychli tvoří vždy dvě čtveřice vzorků ze sousedních řezů.

ad 2) Prahová hodnota určuje jakou izoplochu v daných datech vlastně hledáme. Může například reprezentovat hodnotu tvrdosti materiálu nebo může mít i jiný fyzikální význam. Pokud má prohledávaná krychlička ve všech vrcholech větší (menší) hodnoty než je prahová hodnota, pak můžeme tvrdit, že touto krychličkou hledaná izoplocha neprochází a další výpočty pro tuto krychličku již neprovádíme. V opačném případě pokračujeme bodem 3.

ad 3) Víme, že touto krychlí pravděpodobně prochází hledaná izoplocha. Pro pozdější výpočet normál ve vrcholech jednotlivých trojúhelníků je třeba vypočítat normály ve vrcholech krychle. Směr normál v těchto vrcholech musíme odhadnout. Metoda odhadu vychází z úvahy, že povrch se nachází mezi dvěma objemy lišícími se svou hodnotou a že směr normály je totožný se směrem vektoru gradientu dat $\nabla f = (g_0, g_1, g_2)$. Protože však nelze znát přesný průběh spojité funkce $f(X)$, jejíž vzorky máme k dispozici, musíme

aproximovat gradient pomocí symetrické diference hodnot vzorků $h(X)$ v bodech mřížky o souřadnicích $X = [x_0, x_1, x_2]$. Matematicky vyjádřeno:

$$\begin{aligned} g_0 &= \frac{h(x_0 + a, x_1, x_2) - h(x_0 - a, x_1, x_2)}{2a}, \\ g_1 &= \frac{h(x_0, x_1 + a, x_2) - h(x_0, x_1 - a, x_2)}{2a}, \\ g_2 &= \frac{h(x_0, x_1, x_2 + a) - h(x_0, x_1, x_2 - a)}{2a}. \end{aligned} \qquad \textbf{(1)}$$

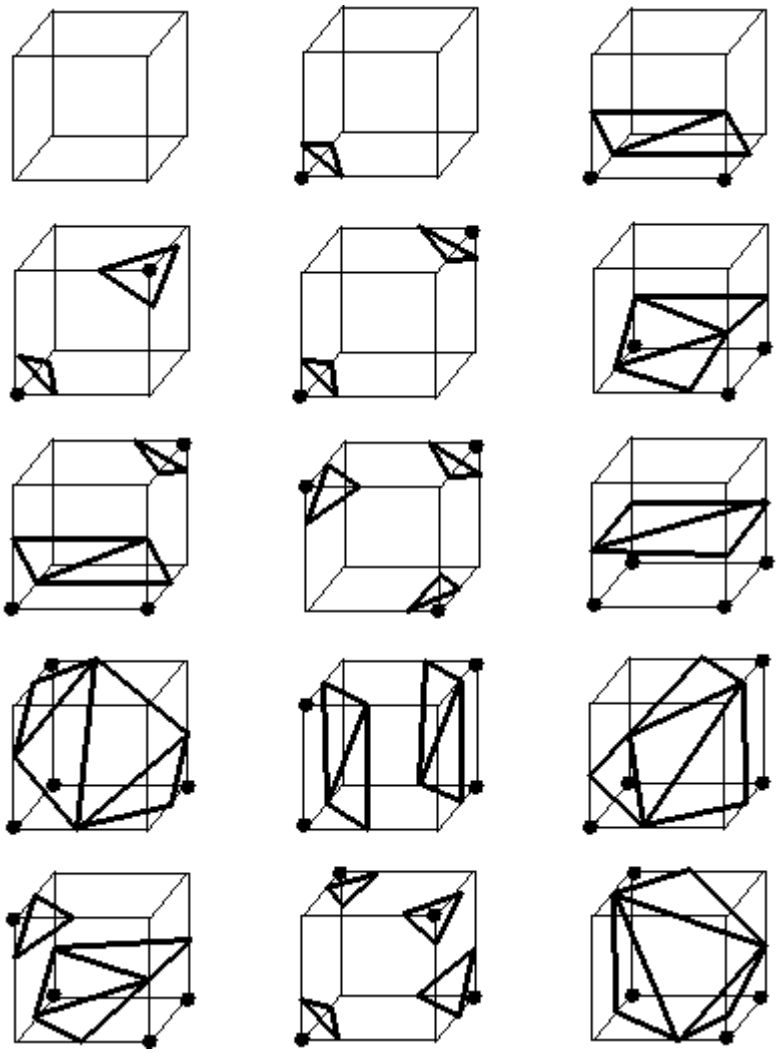
Pokud používáme kartézskou mřížku, kde všechny strany krychle jsou si rovny $a = b = c = konst.$, lze rovnici (2) přepsat na tento tvar:

$$\begin{aligned} g_0 &= h(i+1, j, k) - h(i-1, j, k), \\ g_1 &= h(i, j+1, k) - h(i, j-1, k), \\ g_2 &= h(i, j, k+1) - h(i, j, k-1). \end{aligned} \qquad \textbf{(2)}$$

Tato metoda je však velmi citlivá na lokální změny hodnot. Je tedy třeba ji používat pouze na šumu zbavená (vyhlazená) data.

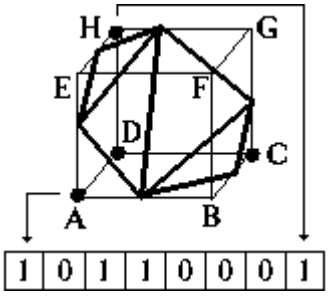
Další metoda pro výpočet normál ve vrcholech krychle (cell) je metoda adaptivního gradientního stínování, která přizpůsobuje velikost okolí pro výpočet gradientu datům.

ad 4) Způsobů, kterými může izoplocha protínat krychli je 256. Je to proto, že krychle má osm vrcholů a každý z těchto vrcholů může být ohodnocen jednou ze dvou hodnot (vrchol je uvnitř izoplochy, nebo vně izoplochy). Existuje tedy $2^8 = 256$ možností. Těchto 256 možností však lze redukovat díky značné symetrii krychle na základních 15 – ostatní vznikají pouze otáčením či inverzí jedné ze základních možností. Patnáct základních konfigurací krychle ukazuje následující obrázek:



Obrázek 4: Základní konfigurace trojúhelníků v krychličce pro metodu Marching Cubes

Index do takovéto tabulky lze vyrobit následujícím způsobem. Hodnoty ve vrcholech krychle, které jsou větší než práh označíme 1. Naopak hodnoty menší označíme 0. Tyto nuly a jedničky složíme dohromady v osmi-bitové číslo které je právě tím kýženým indexem do tabulky konfigurací trojúhelníků.



Obrázek 5: Ukázka sestavení indexu do tabulky pro konkrétní případ

ad 5) Maximální počet trojúhelníků, jak plyne ze všech možných konfigurací jsou čtyři v jedné krychli. V tabulce je pro každý trojúhelník uložen seznam hran krychle, na kterých leží jeho vrcholy.

ad 6) podle hodnoty zadaného prahu a hodnot ve vrcholech krychle lze jednoduše určit přesnou polohu (souřadnici) vrcholu trojúhelníka. Použijeme vztah pro lineární interpolaci.

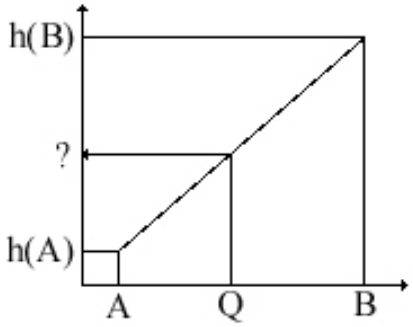
4.3.1 Vztahy pro lineární interpolaci

$$h(Q) = h(A) + (h(B) - h(A))\frac{q_0 - a_0}{b_0 - a_0} \qquad \textbf{(3)}$$

V tomto vztahu interpolujeme hodnotu $h(Q)$ pomocí hodnot $h(A)$ a $h(B)$ při znalosti souřadnic všech bodů. Pro opačnou úlohu použijeme vztah (4).

$$q_0 = a_0 + (b_0 - a_0)\frac{h(Q) - h(A)}{h(B) - h(A)} \qquad \textbf{(4)}$$

Zde hledáme naopak souřadnice bodu $Q = [q_0]$ pomocí souřadnic bodů $A = [a_0]; B = [b_0]$ a hodnot ve všech těchto bodech $h(A), h(B), H(Q) = prah$.



Obrázek 6: Znázornění lineární interpolace

Dále existují i vícenásobné interpolace – bilineární, trilineární, spline, B-spline a podobně. Pro účely metod Marching Cubes i Marching Tetrahedra ovšem postačí pouze

jednoduchá lineární interpolace, neboť vždy hledáme hodnotu souřadnice vzorku ležícího pouze na okraji krychle. Pokud by jsme hledali tuto souřadnici někde uvnitř krychle (například v jejím středu), museli bychom použít trilineární interpolaci.

ad 7) Normály ve vrcholech trojúhelníka vypočítáme použitím vzorce (5). Místo souřadnic vrcholů krychle dosadíme hodnoty jednotlivých složek normál.

$$\begin{aligned} k &= \frac{prah - h(A)}{h(B) - h(A)}, \\ q_x &= a_x + k(b_x - a_x), \\ q_y &= a_y + k(b_y - a_y), \\ q_z &= a_z + k(b_z - a_z). \end{aligned} \qquad \textbf{(5)}$$

4.4 Výhody algoritmu Marching Cubes

Marching Cubes je metoda velice jednoduchá. Lze ji použít jako základ pro další vylepšené metody. Existují zde například různé urychlovací techniky, které generují trojúhelníkovou síť s menší přesností a užívají se zejména při náhledech na kýžený objekt či při nutnosti interaktivního překreslování scény v reálném čase.

4.4.1 Jednoduché urychlovací metody

Pokud je třeba objekt překreslovat v reálném čase za současné změny hledané izoplochy, je třeba oželit detaily výsledného obrazu pro zvýšení rychlosti výpočtu. Je možné například místo výpočtu pozice vrcholů trojúhelníka na hraně pomocí lineární interpolace umístit tento vrchol jednoduše do středu této hrany.

Dalším urychlením by byla možnost nepočítat normály ve vrcholech krychle, a následně neinterpolovat normály ve vrcholech trojúhelníka. Z výsledných souřadnic trojúhelníka jde jednoduše spočítat vektorovým součinem dvou hran tohoto trojúhelníka jeho normála. Ovšem tím se připravíme o možnost stínovat výsledný objekt pomocí Gouraudova stínování. V tomto případě lze použít jedině konstantní stínování.

4.5 Chyby algoritmu Marching Cubes

Přes jednoduchost a zdánlivou spolehlivost tohoto algoritmu byly objeveny i některé nepříjemné nedostatky.

- Nutnost procházet celou 3D mřížku při změně počítané izoplochy.

  Tento problém lze řešit jedině pomocí jistého druhu předzpracování. To spočívá ve vytvoření tzv. „ bitového pole polí“ ve kterém budou pro jednotlivé hodnoty izoploch uloženy ty voxely, kterými daná izoplocha prochází. Tento způsob se ovšem nepoužívá pro jeho velkou paměťovou náročnost a hlavně velmi dlouhou dobu předzpracování. Doba předzpracování je závislá na rozlišení dat a též na rozsahu možných izoploch. Lze ovšem pomocí histogramu určit „zajímavé oblasti“ v datech a ty předpočítat. Při pouhém přepínání mezi těmito „zajímavými oblastmi“ je pak procházení datovou mřížkou o poznání kratší.

- Algoritmus produkuje velké množství trojúhelníků, které se v mnohých případech svou velikostí jen málo liší od velikosti jednoho pixelu obrazovky. Pro „průměrná“ data o velikosti 256x256x108 (CTHead.vol) produkuje algoritmus okolo 600,000 velmi malých trojúhelníčků.

  Jedním z používaných metod pro zmenšení počtu generovaných trojúhelníků je decimace trojúhelníkové sítě. Jedná se o metodu, která redukuje počet trojúhelníků v síti pomocí různých kritérií. Redukce ovšem probíhá s určitou chybou na výstupu. Pro chybu kolem 0.1 dochází ke zmenšení počtu trojúhelníků řádově o 30% - což je při takovém počtu trojúhelníků značný počet.

  Další možností jak snížit počet vykreslovaných primitiv je možnost nedělit krychli na trojúhelníky ale na polygony. Je zde ovšem nutné použít místo lineární interpolace vrcholu polygonu na hraně krychle již dříve zmíněnou metodu umisťování tohoto vrcholu do středu této hrany. Trojúhelníky jsou pak v krychli tvoří rovinu, kterou lze vyjádřit jako polygon.

- Dále může tento algoritmus produkovat trojúhelníky degenerované na jediný bod, či na úsečku.

Trojúhelník degeneruje na bod například v případě, když vrchol krychle má stejnou hodnotu jako je zadaný práh a současně na všech třech hranách vedoucích z tohoto vrcholu leží vrcholy trojúhelníka. Rovnice pro interpolaci bodu na hraně posunou všechny tři souřadnice do vrcholu krychle. Podobným způsobem vznikají i trojúhelníky degenerované na úsečky.

- Může produkovat tak zvané „díry“.

  Pokud se setkají dva doplňkové případy (dvě navzájem opačné konfigurace trojúhelníků v krychli) a tyto buňky navíc současně sdílejí dvojznačnou stěnu, vznikne v daném místě díra. Bylo však zjištěno, že tato možnost nastane pouze v setině procenta případů. Díru lze jednoduše odstranit například zalepením dalšími trojúhelníky. Výsledný povrch je pak souvislý aniž by pro praktické aplikace utrpěla jeho kvalita.

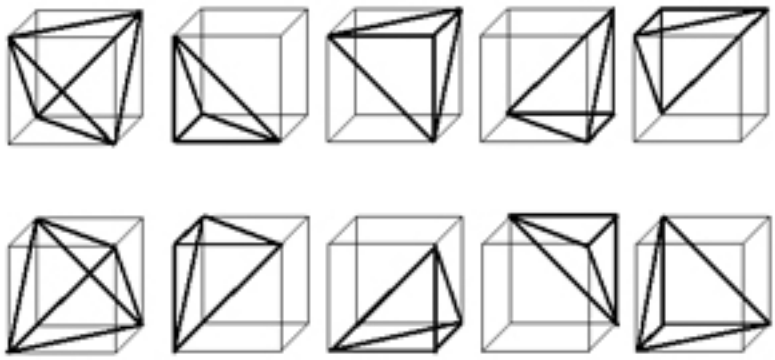
4.6 Marching Tetrahedra

Algoritmus Marching Tetrahedra odstraňuje posledně zmíněný problém metody Marching Cubes – zamezuje vzniku děr. Podobně jako Marching Cubes, zpracovává jednotlivé krychličky (cells) 3D mřížky a snaží se aproximovat hledanou izoplochu pomocí sítě trojúhelníků. Z krychličky ovšem netvoří rovnou index do tabulky konfigurací trojúhelníků, ale dále ji dělí na tetrahedrony (čtyřstěny). V těch potom teprve hledá izoplochu reprezentovanou sítí trojúhelníků. Algoritmus lze shrnout do podobných bodů jako Marching Cubes:

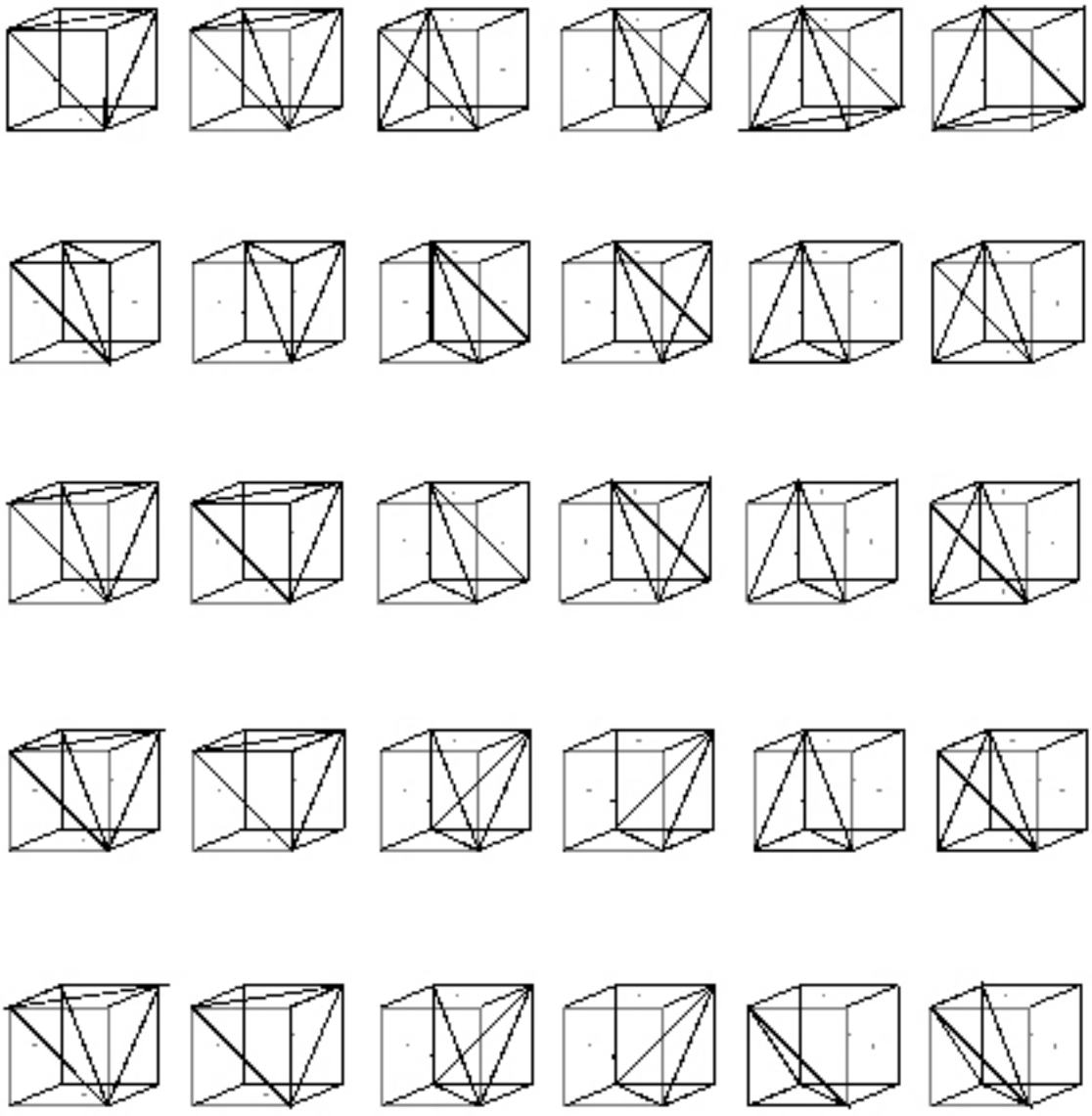
1. Nalezení krychle a určení vrcholových hodnot
2. Kontrola těchto hodnot se zadanou prahovou hodnotou
3. Výpočet normál ve vrcholových bodech krychle
4. Rozdělení krychle na daný počet tetrahedronů
5. Pro každý tetrahedron určit průběh izoplochy pomocí trojúhelníkové sítě
6. Určení hran tetrahedronů na kterých leží vrcholy trojúhelníků
7. Výpočet přesné pozice vrcholů trojúhelníků v rámci hrany tetrahedronu
8. Interpolace normál vrcholů trojúhelníků v závislosti na vrcholových normálách krychle

ad 1-3) Tyto body jsou shodné s dříve popsanou metodou Marching Cubes

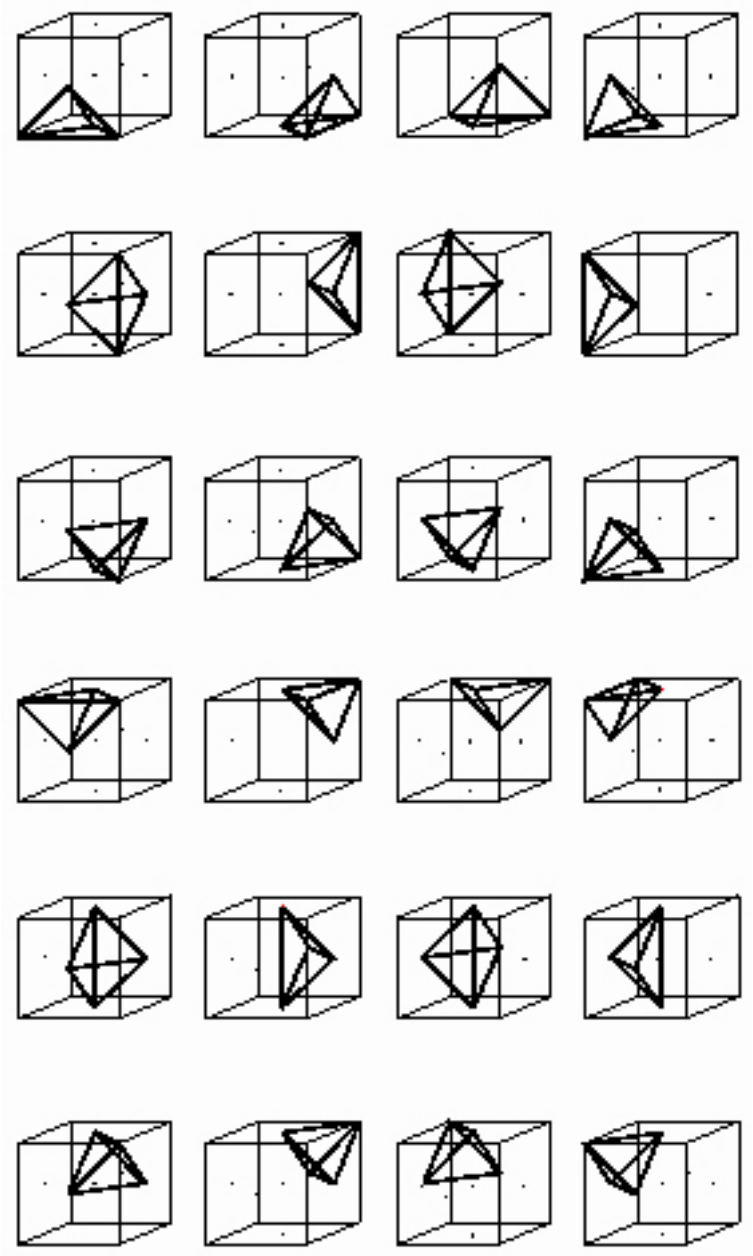
ad 4) Rozdělit krychli na tetrahedrony je možné několika způsoby. Na následujících obrázcích jsou znázorněny tři základní možnosti dělení – na pět, šest a dvacet čtyři tetrahedrony:



Obrázek 7: Dvě možnosti dělení krychle na pět tetrahedronů



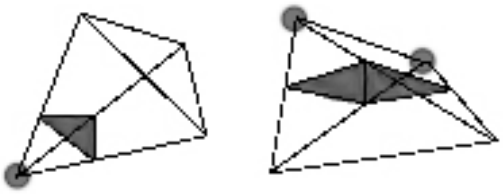
Obrázek 8: Pět možností dělení krychle na šest tetrahedronů



Obrázek 9: Rozdělení krychle na dvacet čtyři tetrahedrony

Pokud dělíme krychli na pět tetrahedronů, je nutné pro zachování návaznosti budoucí trojúhelníkové sítě pravidelně střídat obě možnosti uvedené na obrázku 7. Toto ovšem neplatí pro dělení na šest či více tetrahedronů. V případě dělení na dvanáct a více tetrahedronů (případ pro dvacet čtyři nám ukazuje obrázek 9) vzniká situace, kdy některé vrcholy tetrahedronů nejsou zároveň vrcholy krychle. V tomto případě je nutné dopočítávat hodnoty těchto vrcholů pomocí interpolace vyššího řádu (například trilineární interpolací).

ad 5) V tetrahedronu existuje $2^4 = 16$ různých možností konfigurace trojúhelníků. Lze tedy podobně jako v metodě Marching Cubes vytvořit index do tabulky, který bude čtyřbitový a každý bit bude označovat zda se jemu příslušející vrchol tetrahedronu nachází uvnitř izoplochy či ne. Díky symetrii tetrahedronu existují pouze dvě základní konfigurace trojúhelníků v tetrahedronu.



Obrázek 10: Dva možné způsoby umístění trojúhelníků v tetrahedronu

ad 6,7,8) Ostatní body jsou již shodné s dříve popsanou metodou Marching Cubes

4.7 Výhody algoritmu Marching Tetrahedra

Jak již bylo řečeno, odstraňuje tato metoda problém „děr“ pozorovaný v metodě Marching Cubes.

Další významnou výhodou je, že lze tento algoritmus snadno použít i na vzorky, které nejsou uspořádány v uspořádané mřížce. Z takovýchto neuspořádaných dat je nutné vytvořit síť tetrahedronů například pomocí Delaunayovy triangulace či jiné triangulace bodů v tří rozměrném prostoru. Vzniklé tetrahedrony je také nutno nějakým způsobem uspořádat, aby bylo možné postupně procházet všechny tetrahedrony a zjišťovat v nich průběhy izoploch.

4.8 Nevýhody algoritmu Marching Tetrahedra

Hlavní nevýhodou tohoto algoritmu je počet produkovaných trojúhelníků. Při dělení na pět tetrahedronů stoupne počet trojúhelníků asi o 25 %. Čím více dělíme základní krychli na menší a menší tetrahedrony, tím jsou produkovány menší trojúhelníky a samozřejmě roste jejich počet ve výsledné trojúhelníkové síti.

4.9 Srovnání obou metod

Obě metody jsou koncipovány pro výpočet trojúhelníkové sítě, za účelem co možná nejpřesnější interpretace izoplochy pro zadanou prahovou hodnotu.

Metoda Marching Cubes je výpočetně méně náročná a tím samozřejmě i rychlejší, produkuje podstatně méně trojúhelníků a to výrazně ovlivňuje paměťovou náročnost a v neposlední řadě i rychlost zobrazování.

Na druhou stranu se tato metoda hůře implementuje v důsledku rozsáhlosti tabulky možných konfigurací trojúhelníků v krychli, produkuje povrch, který obsahuje „díry“ a je použitelná pouze pro diskrétní data v pravidelné mřížce.

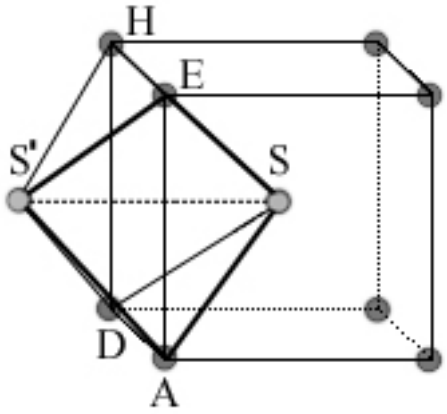
Marching Tetrahedra lze použít i pro data v nepravidelné mřížce, produkuje povrch bez děr, umožňuje adaptivní výběr dělení krychle na tetrahedrony podle konfigurace vrcholů krychle.

Generuje ovšem neúměrně vyšší počet velmi malých trojúhelníků, které se ve výsledném obraze blíží velikosti pixelu což činí tento algoritmus hůře použitelným.

4.10 Nový způsob dělení (body centered)

Další zlepšení metody Marching Tetrahedra přináší speciálního (body centered) dělení krychle na tetrahedrony, které zajišťuje lepší návaznost jednotlivých krychliček na sebe a tím i lepší interpretaci izoplochy pomocí trojúhelníkové sítě. Tato metoda dělí krychli na 24 tetrahedronů, které mají několik základních vlastností:

- Všechny tetrahedrony mají stejný tvar i objem
- Dělení produkuje symetrickou síť tetrahedronů
- Dělení neprobíhá jen v rámci jednoho voxelu, ale tetrahedrony jsou sdíleny sousedními voxely



Obrázek 11: Způsob dělení krychle na tetrahedrony použité metodou body-center

Vrcholy tetrahedronů jsou dvojího typu:

- Vrcholy ležící ve vrcholu krychle – hodnota v těchto vrcholech je daná v mřížce volumetrických dat
- Vrcholy ležící ve středu krychle – tuto hodnotu data neobsahují a je tedy nutné ji dopočítat. Nabízí se zde například metoda trilineární interpolace popsaná v kapitole 4.3.1. Protože bod se nachází přesně ve středu krychle, je možné taktéž použít prostý průměr z hodnot ve vrcholech krychle, které jsou k dispozici.

Kromě dvojího druhu vrcholů můžeme rozlišit i dva typy hran tetrahedronů. „Krátké“ hrany – například SA, SD, SE, SH jsou celé obsaženy v jednom voxelu. Naproti tomu „dlouhé“ hrany – S, S’ zasahují do dvou spolu sousedících voxelů.

Pokud dělíme krychli na 24 tetrahedronů, dalo by se očekávat velký nárůst trojúhelníků ve výsledné trojúhelníkové sítí. Jelikož je ale každý tetrahedron sdílen dvěmi sousedními krychlemi, získává tato metoda pružnosti předchozích způsobů dělení a navíc zlepšuje návaznost izoploch mezi jednotlivými krychlemi.

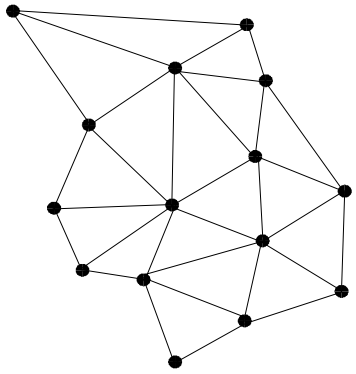
Pro výpočet jednotkových normál ve vrcholech tetrahedronů je použito poněkud upravené metody symetrické diference popsané v kapitole 4.3. Použitou metodu ukazuje rovnice (6).

$$\nabla f_x(i,j,k) = c\left(\frac{f(i+1,j,k)-f(i-1,j,k)}{2a}\right)+$$
$$+\frac{1-c}{4}\left(\frac{f\left(i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}\right)-f\left(i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}\right)}{a}\right)+$$
$$+\frac{1-c}{4}\left(\frac{f\left(i+\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}\right)-f\left(i-\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}\right)}{a}\right)+$$
$$+\frac{1-c}{4}\left(\frac{f\left(i+\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}\right)-f\left(i-\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}\right)}{a}\right)+$$
$$+\frac{1-c}{4}\left(\frac{f\left(i+\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}\right)-f\left(i-\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}\right)}{a}\right) \qquad \textbf{(6)}$$

kde parametr c je váhový koeficient, který nabývá hodnotu $c \in \langle 0;1 \rangle$. Byly testovány různé hodnoty tohoto parametru, avšak bez většího výsledného efektu na kvalitu výstupních dat.

5 Trojúhelníková síť

Trojúhelníkové sítě produkovány v předchozích odstavcích popsanými metodami se snaží co nejvíce aproximovat povrch hledané izoplochy (objektu), a to tak, aby rozdíl objemů výsledné izoplochy byl co nejmenší od hledané izoplochy.



Obrázek 12: Příklad jednoduché trojúhelníkové sítě

Pro snadnou manipulaci s touto sítí by měla splňovat několik dalších požadavků:

- Normály trojúhelníků by měli být stejným směrem (ven z objektu či dovnitř do objektu)
- Měli by mít stejný směr otáčení vrcholů (CW, CCW)
- Sítě by měli být co nejvíce regulární
- Trojúhelníky by neměli být degenerované na body či úsečky

6 Modular Visualisation Environment

6.1 Úvod

V rámci projektu na ZČU byl vyvinut systém MVE – Modular Visualisation Environment, který se skládá ze dvou částí:

- Editor
- Run-time

MVE v sobě zahrnuje programové API, v rámci kterého je možné tvořit moduly. Tyto lze následně vložit do MVE-editoru pospojovat jejich vstupy a výstupy ve výslednou aplikaci . S pomocí MVE-Run-time lze dále s touto aplikací pracovat. Jednotlivé moduly jsou zde spouštěny jako samostatně běžící úlohy (vlákna) v operačním systému, a tudíž mohou běžet paralelně. V současné době je též testována možnost paralelizace mezi více počítačů v rámci sítě – tzv. distribuce modulů mezi více sítí spojených výpočetních systémů.

6.2 Moduly

Moduly spustitelné pod MVE jsou jednoduché DLL knihovny. Jedna DLL knihovna může obsahovat i více modulů. Každý modul musí mít implementováno pět základních funkcí:

- `XXX_MAIN_MODULE_FUNC`
- `XXX_FREE_DATA`
- `XXX_SETUP_FUNC`
- `XXX_FREE_SETUP_DATA`
- `XXX_FREE_STATE`

Kde XXX zastupuje jméno modulu. Kromě těchto funkcí obsahuje DLL knihovna ještě dvě funkce, pomocí kterých MVE pozná, kolik modulů a s jakým jménem je v knihovně přítomno:

- `Get_Modules`
- `Free_DLL_Descr`

7 Implementace vybraných metod

7.1 Úvod – výběr algoritmů

Pro implementaci byly zvoleny tři základní metody : Marching Cubes, Marching Tetrahedra 5, Marching Tetrahedra 6. Tyto metody jsou poměrně dobře použitelné pro prostředí MVE popsaném výše. V tomto odstavci se budeme zabývat vlastní implementaci vybraných metod, použití spolu s MVE, testování a ladění. Dále budou popsány použité datové struktury pro volumetrická data a pro trojúhelníkové sítě. V neposlední řadě bude popsán použitý způsob urychlení základních metod pomocí speciální vyrovnávací paměti.

7.2 Požadavky

Vybrané metody budou implementovány pro systém Modular Visualisation Environment jako moduly ve formě DLL knihoven. Tyto moduly bude možné používat spolu s ostatními moduly v rámci MVE a budou využívat jednotně definované datové struktury projektu. Celý systém MVE je vytvořen pro architekturu Intel a operační systém Win95/98/NT. Tím je také předurčena architektura modulů.

DLL knihovny jsou ovšem nezávislé na MVE a lze je bez problémů použít i v jiném vizualizačním software.

Celý program je psán jazykem Microsoft Visual C++ 6.0. Funkce pro výpočet izoploch a načítání dat jsou psány bez využití objektového programování a dalších nestandardních knihoven. Bez větších úprav je možné tyto funkce exportovat na jinou platformu. Tento export byl úspěšně testován například na Linux RedHat 6.0. Naproti tomu modul rendereru trojúhelníkových sítí se neobejde bez závislosti na operačním systému a tudíž jej nelze

jednoduše přenášet na jiné platformy. Přispívá k tomu i použití knihoven MFC (Microsoft Foundation Classes), která se v prostředí Microsoft Visual C++ 6.0 stará o vykreslování dialogů, a vůbec o celý způsob interakce uživatele s počítačem. V neposlední řadě umožňuje spravovat okno s výřezem OpenGL.

Další požadavek je počet podporovaných formátů. Aplikace bude schopná s pomocí informačního souboru poznat o jaký druh volumetrických dat jde, správně jej načíst a manipulovat s ním.

Interaktivní ovládání zobrazovače spolu s možnostmi jednoduchého testování a zobrazení informací o datech je samozřejmostí (počet trojúhelníků, vrcholů, rozlišení dat atd.). Zobrazovač bude využívat rozhraní OpenGL. Bude též podporovat stavové veličiny OpenGL jako například typ projekce, použití OpenGL Listů, nastavení typu stínování. Bude dále umožňovat způsob vykreslení objektu (čárový, konstantně stínovaný, Gouraud shading). V omezené míře bude umožňovat jednoduché animace, ukládání BMP souborů.

Dále bude možné vypočítaný objekt exportovat do STL nebo TRI formátu na vnější paměťové médium.

7.3 Datové struktury

Datové struktury byly navrženy v projektu MVE na ZČU. Jsou to standardní paměťové struktury pro tento projekt, které zajišťují kompatibilitu datových vstupů a výstupů jednotlivých modulů.

Pro účely generování izoploch z volumetrických dat je třeba těchto dvou datových struktur:

- Volumetrická data – vstup modulu
- Trojúhelníkové sítě – výstupní izoplocha

7.3.1 Volumetrická data

Volumetrická data byla zběžně popsána v úvodní kapitole této práce. Zde se soustředíme na fyzický formát těchto dat na vnějším paměťovém médiu a také organizaci těchto dat v paměti výpočetního systému.

7.3.1.1 Diskový formát

Jelikož existuje více formátů volumetrických dat a každý používá jiný způsob identifikace (jinou hlavičku souboru), je nutné nějakým způsobem sjednotit způsob přistupování k těmto datům. Pro účely načítání volumetrických dat byl vyvinut speciální formát souboru, který je distribuován společně s volumetrickými daty a popisuje jejich strukturu. Tento soubor má příponu **.vif* (Volume Info File). V souboru jsou uloženy všechny potřebné informace ne jen k jeho správného načtení, ale i jeho počátečního umístění v okénku rendereru, případně otočení, změny velikosti atd.

Příklad souboru Volume info file je textový soubor následujícího formátu:

```
# Generated for CThead.vol file
[Main]
 Caption = CT Head
 Source = CT
 Header Size = 0
 Bytes per voxel = 1
 Signed = no
 Byte order file = file

[Resolution]
 x = 256
 y = 256
 z = 108

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

[correction - rotation]
 x = 90
 y = 0
 z = 0
```

```
[Distortion]
 x = 1.0
 y = 1.0
 z = 0.42
```

Jak je z příkladu vidět, je text rozdělen do sekcí tvořených názvem sekce v hranatých závorkách. Jednotlivé sekce pak obsahují parametry, blíže určující vlastnosti čtených dat.

- **Main** – obsahuje základní informace o souboru
  Caption – název souboru
  Source – typ použitého vzorkovacího přístroje či jiný popis zdroje dat
  Header Size – velikost hlavičky souboru, kterou je nutno při čtení přeskočit
  Bytes per voxel – počet bytů na jeden vzorek (obvykle jeden nebo dva)
  Signed – typ vzorkované hodnoty – znaménkový či neznaménkový
  Byte order file – převrácené či nepřevrácené pořadí horního a dolního byte při dvoubytovém uspořádání hodnoty vzorku
- **Resolution**
  x – rozlišení dat ve směru osy x
  y – rozlišení dat ve směru osy y
  z – rozlišení dat ve směru osy z
- **Correction – position** (platné pro vykreslovací modul)
  x – počáteční posunutí dat ve směru osy x
  y – počáteční posunutí dat ve směru osy y
  z – počáteční posunutí dat ve směru osy z
- **Correction – rotation** (platné pro vykreslovací modul)
  x – počáteční rotace dat ve směru osy x
  y – počáteční rotace dat ve směru osy y
  z – počáteční rotace dat ve směru osy z
- **Distortion** (platné pro vykreslovací modul)
  x – změna měřítka dat ve směru osy x
  y – změna měřítka dat ve směru osy y
  z – změna měřítka dat ve směru osy z

7.3.1.2 Paměťový formát

Jako paměťovou datový formát bylo nutno dodržet následující strukturu MVE (Visual C++):

```
typedef struct {
  // základní informace
  char Caption[255];       // Titulek či jméno dat
  char Source[255];        // Zdroj dat

  // ************ statická část ************
  T_sInt     Res_x, Res_y, Res_z; // Resolution
  T_sInt     Pos_x, Pos_y, Pos_z; // Correction position
  T_sInt     Rot_x, Rot_y, Rot_z; // Correction rotation
  T_sFloat   Dis_x, Dis_y, Dis_z; // Distortion

  T_Int StartTreshold;     // Doporučený počáteční práh
  T_Int Min, Max;          // Maximum a minimum v datech
                             (intenzita voxelů)

  // ************ dynamická část ************
  P_sInt Data;             // Volumetrická Data
                             (typ short - 2byte integer)
 } T_Volume_Data, *P_Volume_Data;
```

Tato paměťová datová struktura je navržena tak, aby do ní mohla být načtena většina používaných typů volumetrických dat. To znamená, že je schopna pojmout znaménkové i neznaménkové čísla, jednobytové i dvoubytové hodnoty vzorků.

Jak je z této struktury zřetelné, většina proměnných v paměťové datové struktuře je shodná s informacemi obsaženými ve **.vif* souboru. Tyto lze tedy jednoduše do struktury zkopírovat. Min a Max hodnoty je nutné počítat při načítání souboru. Z těchto hodnot lze pak následně počítat i počáteční doporučený práh (treshold). V našem případě je použita hodnota:

$$prah = Min + 0{,}4(Max - Min) \quad \rightarrow \quad 40\%\ \text{práh} \qquad (7)$$

Vlastní volumetrická data jsou uložena v položce příhodně nazvané ***Data***. Je to vlastně trojrozměrné pole, uložené dynamicky v jednom vektoru. Pro přístup do takto vytvořeného pole je použito mapovací funkce, která zajistí vytvoření indexu do tohoto jednorozměrného pole pomocí souřadnic hledaného vzorku (x,y,z).

$$index = Data[(z + (zres/2)) * (xres * yres) + resy(y + (yres/2)) + (x + (xres/2)) + 1] \quad \textbf{(8)}$$

Tato rovnice je platná pro obecná data (pro obecné rozlišení dat). Výpočet by se mohl zdát poměrně složitý, vzhledem k velice vysokému počtu ke vzorkům v datové struktuře. Jelikož rozlišení volumetrických dat bývá většinou mocnina dvou – což vyplívá z konstrukce vzorkovacích zařízení, je možné mapovací funkci zjednodušit užitím speciálních instrukcí procesoru:

$$index = Data[(z + (zres/2))shl16 + (y + (yres/2))shl8 + (x + (xres/2)) + 1] \quad \textbf{(9)}$$

Zde instrukce ***shl*** znamená posun bitů v hodnotě čísla doleva o daný počet. Jeden posun bitu představuje vynásobení posouvaného byte dvěma. Tím by se mělo ušetřit mnoho násobení než při použití obecné mapovací funkce.

Praxe však ukazuje, že moderní kompilátory, jakým Visual C++ bezesporu je, však dokáží takovéto přístupy do polí při optimalizaci kódu detekovat z větší části optimalizovat sami.

7.3.2 Trojúhelníková data

7.3.2.1 Diskové formáty – STL a TRI

Formátů pro uložení trojúhelníkových sítí je celá řada. Každý formát je určen pro řady různých účelů. Některé formáty dokáží exportovat 3D animační software – jako např. 3D Studio (MAX), Light Wave a podobné. Další skupinou formátů mohou být technické 3D výkresy exportované například z programu AutoCad.

V našem případě se však soustředíme na jeden formát všeobecně používaný pro jednoduché trojúhelníkové sítě – ***STL*** a druhý formát, který byl vyvinut speciálně pro potřeby projektu MVE – ***TRI***.

Formát ***STL*** obsahuje pouze trojúhelníky s normálou. Neobsahuje tedy normály pro každý vrchol trojúhelníka a to poněkud omezuje způsob vykreslování načtené trojúhelníkové

sítě. Další nevýhodou tohoto formátu je způsob ukládání trojúhelníků. Trojúhelníky jsou ukládány postupně za sebou se všemi svými vrcholy. Pokud sousedí dva trojúhelníky a mají společné vrcholy, objeví se ve výstupním souboru tyto společné vrcholy vícekrát. Při načítání takového souboru je tedy nutné znovu a znovu prohledávat seznam doposud načtených vrcholů a kontrolovat, jestli neobsahuje právě načítaný vrchol. Pokud ne, pak je vše v pořádku. Pokud však ano, je nutné tento vrchol zahodit a změnit index na tento vrchol v datové struktuře.

Tyto nedostatky řeší formát ***TRI***, speciálně vyvinutý pro MVE, který ukládá nejprve pouze vrcholy trojúhelníků – vznikne tím jakýsi seznam všech vrcholů celé trojúhelníkové sítě, potom teprve ukládá jednotlivé trojúhelníky, které jsou určeny třemi indexy do již existujícího seznamu vrcholů. Takováto organizace je daleko bližší paměťové datové struktuře MVE a dovoluje lepší přístup k trojúhelníkové síti. Složitost čtení takovéto sítě je lineární - $O(n)$ naproti kvadratické složitosti načítání STL - $O(n^2)$.

Složitost načítání STL lze sice vylepšit použitím hashovacích tabulek, ale ne o mnoho a uspěšnost urychlení silně závisí na datech.

Příklad jednoho trojúhelníka ve formátu STL:

```
solid jmeno
  facet normal -9.602787e-001 1.460201e-001 2.377875e-001
    outer loop
      vertex 4.190084e+001 -3.809612e+001 3.045739e+001
      vertex 4.160745e+001 -4.403940e+001 3.292221e+001
      vertex 4.246319e+001 -3.809612e+001 3.272838e+001
    endloop
  endfacet
```

Ukázka části souboru TRI: (jedná se pouze o ukázku – data jsou zkrácena a tím nemají smysl)

```
# test triangle file
[Vertices]
0.5 0.0 0.0
-0.5 -0.5 0.5
0.5 -0.5 0.5

[Triangles]
1 0 2
2 0 3

[Triangles' Normals]
-0.0 0.5 -0.5
-0.5 0.5 -0.0

[Vertices' Normals]
-0.0 0.5 -0.5
-0.5 0.5 -0.0
-0.0 0.5 -0.5
```

7.3.2.2 Paměťový formát

Opět zde byla vyvinuta paměťová struktura kompatibilní se strukturou MVE (Visual C++):

```
typedef struct {
  // information part for Renderer
  T_Float xmin, xmax, ymin,ymax, zmin, zmax; //************
  T_Float Pos_x, Pos_y, Pos_z; // Correction position
  T_Float Rot_x, Rot_y, Rot_z; // Correction rotation
  T_Float Dis_x, Dis_y, Dis_z; // Distortion
```

```
  // static part
  T_Index NV_M;     // Space allocated for vertices
  T_Index NT_M;     // Space allocated for triangles
  T_Index NV;       // Number of valid vertices
  T_Index NT;       // Number of valid triangles
  T_Status Orientation;

  // dynamic part
  P_Coord P_VCoor[MAX_DIM];  // Coordinates (x,y ...)
  P_Coord P_VNorm[3];        // Vertex normals
  P_Vertex_Status P_VStatus; // State flags of verticies
  P_Index P_TV[3]; // Triangle verticies
  P_Index P_TT[3]; // Triangle neighbours
  P_Coord P_TNorm[3]; // Triangle normals
  P_Triangle_Status P_TStatus;
  T_Byte z_valid;
} T_Triangle_Mesh, *P_Triangle_Mesh;
                        // Information about triangle mesh
```

Opět se v této struktuře nachází všechny informace o posunu, rotaci, rozlišení, distorzi a dalších základních údajích. Ve statické části této struktury se dále nachází informace o počtu trojúhelníků a počtu vrcholů zde přítomných – ***NT*** a ***NV***. Při ukládání do této struktury se vždy vytvoří nějaké místo – řekněme pro asi tak 6000 trojúhelníků a 2000 vrcholů = (6000/3). Pokud během ukládání místo dojde, zvětší se alokovaná oblast. Po ukončení výpočtu nám může vzniknout volná paměť za všemi zapsanými daty. Tím se tedy nemusí shodovat velikost obsazené paměti a velikost paměti alokované. Pro velikost alokované paměti jsou zde tedy další proměnné – ***NT_M*** a ***NV_M***.

Vrcholy trojúhelníků jsou uloženy v ***P_VCoord*** a samotné trojúhelníky pak v ***P_TV***. Normály k bodům jsou v ***P_Vnorm*** a podobně normály k trojúhelníkům jsou uloženy v ***P_TV***. Všechny posledně jmenované proměnné jsou pole tří ukazatelů na pole hodnot postupně pro x,y,z. Ostatní proměnné datové struktury nebudeme zvětší části využívat.

7.4 Marching Cubes

Výpočet izoplochy pomocí metody Marching Cubes byl implementován jako modul spustitelný pod MVE. Tento systém klade jistá omezení na způsob výpočtu izoploch. Především je nutné sestavit další modul ***VolumeLoader***, který data načte z vnějšího média do paměťové struktury. Poté předá ukazatel na tuto strukturu modulu ***MarchingCubes***. Metoda tedy dostane k dispozici celé 3D skalární pole hodnot v paměti. Není tedy možné pomýšlet na různé druhy šetření paměti postupným načítáním jen potřebného množství dat a po skončení partikulárního výpočtu tyto data zase uvolňovat. Algoritmus také nesmí samovolně vstupní data měnit, neboť mohou být používána souběžně i jinými moduly (například další metody pro výpočet izoploch nebo moduly pro odstranění šumu z dat atd.).

Implementována byla základní metoda Marching Cubes s využitím vyrovnávací paměti popsané v následujícím kapitole. Algoritmus postupně probírá všechny buňky a zjišťuje výskyt izoplochy. Pokud buňku protíná, pak vypočte standardní metodou všechny trojúhelníky, které jí aproximují a uloží do výstupní datové struktury. Pokud ne, pak pokračuje další buňkou v pořadí.

Při ukládání trojúhelníků do výstupní datové struktury je též nutné dopočítat parametry vypočtených trojúhelníků. To je třeba maximální a minimální hodnoty v jednotlivých souřadnicích, distorze – závislá na rozlišení volumetrických dat.

Výsledná data mohou být dále zpracována například modulem pro decimaci trojúhelníkové sítě, který se rovněž vyvíjí v rámci projektu na ZČU, rovnou předána modulu předána modulu ***Renderer***, který je určen pro vykreslení trojúhelníkových sítí.

Metoda Marching Cubes je základní metodou, kterou je možné použít k ověření funkčnosti jí podobných metod pod MVE.

7.5 Marching Tetrahedra

Implementace metod Marching Tetrahedra je podobná, jako u metody Marching Cubes. Opět jsou k dispozici všechny potřebné informace v paměti předané načítacím modulem. Vzhledem k podobnosti přístupů obou těchto metod k datům, bylo možné pro urychlení použít podobnou vyrovnávací paměť popsanou v následující kapitole.

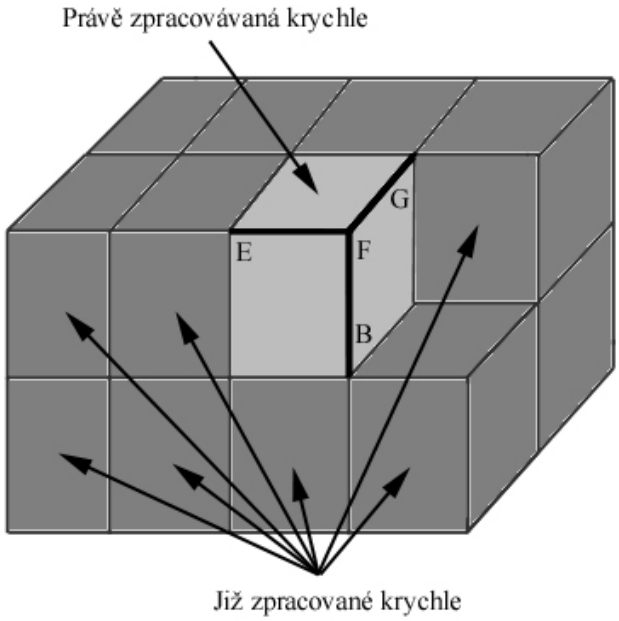
Pro porovnání byly implementovány dva způsoby dělení základní buňky (cell) na tetrahedrony:

- Marching Tetrahedra 5
- Marching Tetrahedra 6

Obě metody se liší hlavně počtem generovaných trojúhelníků, a tím i rychlostí zpracování dat, ale také kvalitou vypočtené izoplochy.

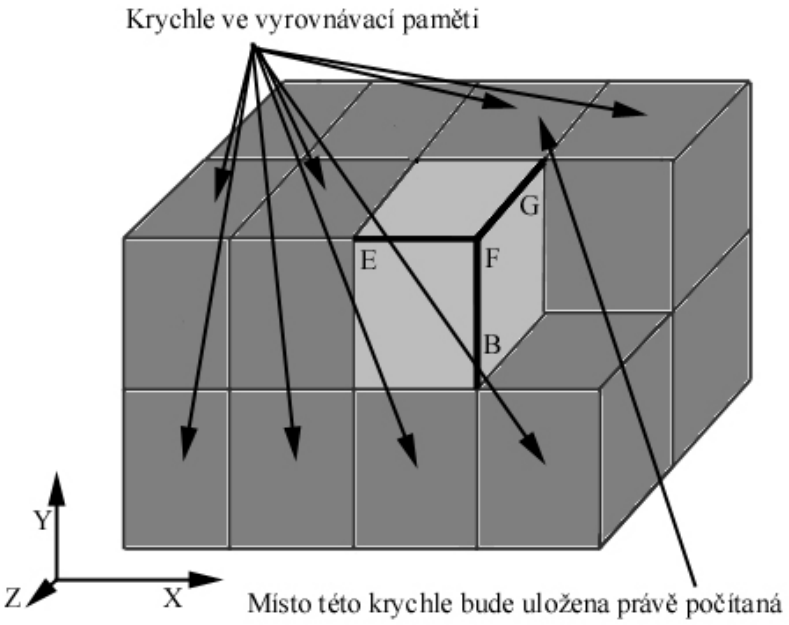
7.6 Vyrovnávací paměť

Pro urychlení obou algoritmů a zefektivnění přístupu do paměti byly základní metody rozšířeny o speciální způsob použití vyrovnávací paměti.



Obrázek 13: Dopočítávané hrany pro každou krychli

Každá krychle, ve které počítáme normály v jejích vrcholech sousedí s šesti dalšími krychlemi (obrázek 13). Pokud postupujeme při výpočtu postupně v jednotlivých souřadnicích – nejprve se mění x-ová souřadnice, poté y-ová a naposled z-ová (jak je vidět na obrázku 14), můžeme s určitostí tvrdit, že hledaný bod na hranách společných s předchozími (již vypočtenými) krychlemi byl již spočten a tudíž jej není potřeba znovu dopočítávat.



Obrázek 14: Způsob ukládání do vyrovnávací paměti

Vyrovnávací paměť je velikostně shodná s rozlišením dat ve směru x a ve směru y . Její velikost tedy závisí na rozlišení dat a je rovna $size = xres \cdot yres \cdot sizeof(buff)$. Kde $sizeof(buff)$ je velikost paměti potřebná k uložení všech informací o vypočítané krychli.

K jednotlivým položkám ve vyrovnávací paměti přistupujeme pouze jako do dvou rozměrného pole pomocí souřadnic x a y. Algoritmus ukládá mezivýsledky v právě počítané krychli do prozatímní paměti $buff$, která se po skončení výpočtu nakopíruje na místo ve vyrovnávací paměti o souřadnicích x a y (souřadnice právě dopočítané krychle). Tímto smažeme krychli, kterou již zcela jistě nebudeme k výpočtu potřebovat, neboť všechny její sousedy již máme spočteny.

Výhody tohoto uspořádání vyrovnávací paměti jsou převážně v tom, že je celá alokována na začátku výpočtu a není potřeba zdržovat jeho průběh neustálou alokací a dealokací. Tato metoda velmi výrazně urychlí všechny tři algoritmy pro výpočet izoploch. Rovněž také zajistí shodu vypočítaných bodů společných pro více trojúhelníků. Nemělo by se tedy stát, že trojúhelníky nenavazují na sebe.

8 Renderer

8.1 Úvod

Renderer je určen pro zobrazování trojúhelníkových sítí. Je to vlastně takový univerzální zobrazovač v prostředí MVE, který je schopen zobrazit trojúhelníkové sítě přejaté od ostatních výpočetních modulů systému. Je to tedy jeden z nejpoužívanějších modulů pro zobrazení výsledků ve formě 3D scény.

8.2 Předpoklady

Aby bylo možné zobrazit výstup více modulů zároveň, je nutné, aby byl kód rendereru schopen reentrantní práce. Dále musí nabízet volby pro jednoduchou manipulaci se zobrazovanou scénou. Jedná se především o posun, rotace a změna měřítka objektu. Dále bude umožňovat změnu parametrů projekcí, barev, a dalších užitečných nastavení.

Renderer využívá OpenGL jako grafický engine. Použití OpenGL přináší řadu výhod. Především je to snadná podpora grafického hardware (akcelerátoru) a v neposlední řadě i možnost snadnějšího přepracování částí kódu, které jej využívají, pro jiné platformy.

1.3 Možnosti Rendereru

Renderer je stále ve stádiu vývoje. To je ovšem způsobeno současným stavem projektu MVE, kde neustále přibývají nové a nové moduly a každý uživatel rendereru jako výstupního modulu má poněkud jiné požadavky. Z tohoto důvodu je renderer neustále rozvíjen o nové a nové nastavení. V době přípravy tohoto textu jsou podporovány tyto nastavení:

- Změna typu projekce
- Způsob stínování
- Způsob vykreslení objektu
- Nastavení barev
- Dialog s informacemi o objektu
- Vypnutí/zapnutí vykreslování os souřadného systému

- Podpora OpenGL list
- Volitelný parametr glCullFace
- Volitelný parametr glSingleSide
- Vypnutí/zapnutí světel
- Nastavení parametrů světel ve scéně
- Změna orientace normál
- Možnost použití stereo-projekce
- Export obrázků ve formátu BMP (jednoduchý BMP, více BMP – animace)
- Export trojúhelníkových sítí (TRI a STL formáty)
- Texturování pomocí environmental textur
- Efekt ChromaDepth pro vykreslování stereo obrázků

Typy projekce jsou podporovány dva – paralelní a perspektivní. Pokud je ovšem zapnuta i možnost stereo projekce, je automaticky zapnuta perspektivní projekce. Je to proto, že perspektivní projekce je lidskému vnímání 3D scény daleko bližší.

Pro zlepšení vjemu tří-rozměrné scény a vnímání vlastního objektu je velice důležité stínování. Stínování je možné použít buď konstantní, nebo Gouraudovo. Jelikož pro Gouraudovo stínování je potřeba mít k dispozici normály pro všechny vrcholy trojúhelníka a ne pouze jednu normálu pro celý trojúhelník, jsou tyto normály v případě že nejsou na vstupu rendereru dopočítány v modulu pro čtení trojúhelníkových dat z disku. Stejně tak i normály pro celý trojúhelník potřebné pro konstantní stínování mohou být dopočítány pokud nejsou ve vstupních datech přítomny.

Objekt může být vykreslován třemi různými způsoby:

- Stínované trojúhelníky
- Drátěný model
- Pouze vrcholy – bodový model

Při užití grafického akcelerátoru je však nejvhodnější používat stínované trojúhelníky – konstantní stínování, neboť je podporováno přímo v hardware. Ostatní mohou ale také nemusí být vykreslovány s pomocí grafického akcelerátoru a tím může doba vykreslování neúměrně

narůst. Použití drátěného modelu je například při zkoumání tetrahedronových sítí či nějaké triangulace ve tří-rozměrném prostoru.

Barvy jsou nastavitelné zvlášť pro vykreslovaný objekt a zvlášť pro pozadí objektu. Je zde ještě jedna nastavitelná barva, která určuje barvu „podlahy“ pokud je tato funkce zapnuta. Výpis informaci o objektu obsahuje všechny informace předané v trojúhelníkové datové struktuře – rozlišení, korekce posunu a rotace, tzv. bounding-box (ohraničující krychle), a hlavně také počet vykreslovaných trojúhelníků a k nim náležejících vrcholů. Tyto informace jsou důležité pro vlastní testování metod implementovaných do generujících modulů.

Další nastavení se týkají parametrů OpenGL. Knihovna OpenGL disponuje řadou funkcí pro vykreslování objektů na obrazovce. Tyto funkce se ale volají neustále pro každý trojúhelník zvlášť a tím se zpomaluje doba vykreslení scény. Funkce *glCallList* umožňuje vykreslení trojúhelníkové sítě do jakési vyrovnávací paměti a tu potom vykresluje bez volání dalších funkcí. Vykreslování je znatelně rychlejší. Je ovšem třeba si uvědomit, že při výrobě OpenGL listu se do něj kopírují vstupní trojúhelníky, a tím zabírá vykreslovaný objekt mnohem více paměti. Je nutné mít dostatek této paměti, protože pokud začne počítač odkládat stránky kódu nebo dat na disk, je tento způsob o mnoho pomalejší než klasický přístup.

Další urychlení přináší funkce *glCullFace*, která vypíná vykreslování „neviditelných“ trojúhelníků. Tato „neviditelnost“ trojúhelníka je určována podle směru otáčení jeho vrcholu – a tím tedy i směru normály v závislosti na poloze oka pozorovatele. Pokud je tedy trojúhelník odvrácen od pozorovatele (resp. přivrácen opačnou stranou), není jej potřeba vykreslovat, neboť by stejně nebyl ve výsledném obraze vidět. Funkce *glSingleSide* umožňuje při vypnutém *glCullFace* nastavit stejný materiál vnitřní straně trojúhelníka jako má strana vnější. Toto nastavení je dobré v případě zobrazování dat, které nemají správně orientaci normál – některé normály směřují dovnitř povrchu a některé zase ven.

Pokud jsou však všechny normály orientovány špatným směrem, pak je možné použít další funkci rendereru pro jejich obrácení. Tyto normály jsou obráceny nejen pro výpočet výsledného obrazu (renderovaného), ale jsou upraveny i normály v datech které je pak možné přímo exportovat do výstupního souboru.

Možnost použití stereo-projekce je jednou z nejzajímavějších vlastností rendereru. Je ovšem také třeba grafický hardware, který tuto možnost podporuje. Renderer byl testován na s grafickou kartou Intense/WildCat firmy Intergraph a s brýlemi Crystal Eyes. Při stereo projekci dochází ke kreslení zvlášť obrazu pro levé oko a zvlášť pro pravé. Obrazy jsou při tom vzájemně posunuty a otočeny přesně tak, aby co nejvíce napodobovali přirozený pohled na objekt. Nastavení vzdálenosti pohledu od objektu a posunu očí od sebe je samozřejmě možné měnit.

Export obrázků je možný dvěma způsoby. První možnost je exportovat jednotlivé bitmapy ve formátu Windows BMP souboru, druhá možnost je tvorba jednoduché animace a export této animace jako soubor BMP souborů číselně indexovaných. Pojmem jednoduchá animace je myšleno například animování otáčení objektu kolem jedné z os souřadného systému, posun či změna velikosti. Výsledné BMP soubory je pak možné pomocí externího software sloučit do nějakého formátu určeného pro ukládání animací a videa – AVI, MPEG, MOV atd.

Trojúhelníkové sítě je možné exportovat do stejných typů formátů (STL, TRI), které mohou být čteny modulem pro čtení trojúhelníkových sítí – ***Triangle_Loader***. Čtecí moduly byly popsány v předchozích kapitolách.

Další funkce - texturování, speciálně environmental texturování, je určena pro emulaci nějakého materiálu na povrchu objektu. Můžeme tedy nadefinovat materiál například zlato, kov atd. a ten přiřadit objektu. Objekt pak tedy získá více z reálného světa.

Zcela podobně jako environmental texturování je proveden efekt CHROMADEPTH, který zobrazovaný objekt obarvuje od předu ve směru pohledu do zadu barevným spektrálním přechodem – (červená - modrá). Tento efekt umožňuje vnímání objektu 3D při pohledu speciálními brýlemi. Více v [7].

9 Výsledky

9.1 Tabulka testovaných volumetrických dat a naměřené hodnoty

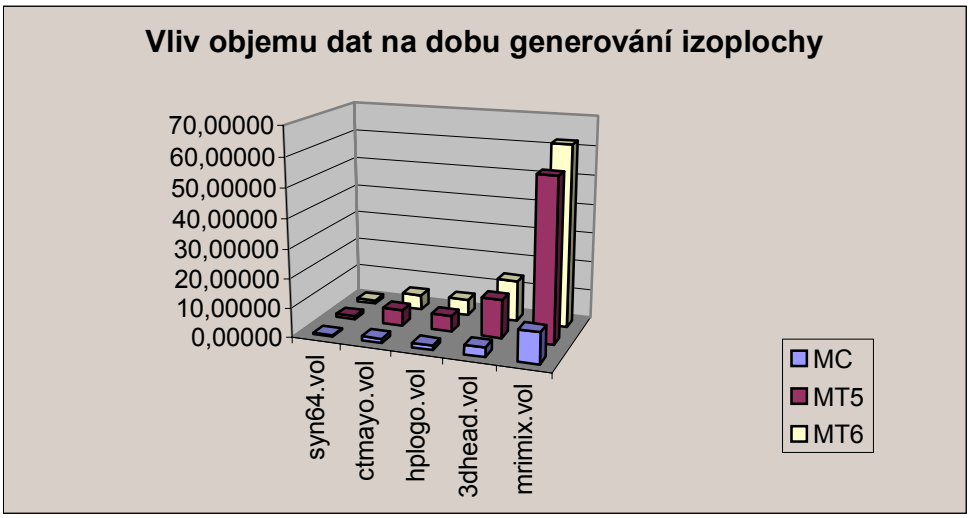
Algoritmus výpočtu izoplochy :	**Marching Cubes**		
Jméno	*Rozlišení*	*Čas generování [s]*	*Počet trojúhelníků*
syn64.vol	***64x64x64***	0,36000	14 796
ctmayo.vol	***128x128x128***	1,47722	196 206
hplogo.vol	***236x150x64***	1,47722	232 552
3dhead.vol	***256x256x109***	3,14317	433 768
cthead.vol	***256x256x108***	3,17800	576 190
engine.vol	***256x256x108***	3,47722	556 012
mrbrain.vol	***256x256x109***	4,14317	838 424
mrimix.vol	***256x256x180***	10,50000	1 742 204

Algoritmus výpočtu izoplochy :	**Marching Tetrahedra 5**		
Jméno	*Rozlišení*	*Čas generování [s]*	*Počet trojúhelníků*
syn64.vol	***64x64x64***	1,14317	41 592
ctmayo.vol	***128x128x128***	5,47722	487 792
hplogo.vol	***236x150x64***	5,47722	653 128
3dhead.vol	***256x256x109***	13,12000	1 049 994
cthead.vol	***256x256x108***	15,14317	1 423 758
engine.vol	***256x256x108***	15,81000	1 428 968
mrbrain.vol	***256x256x109***	18,85000	1 990 444
mrimix.vol	***256x256x180***	55,47722	4 156 168

Algoritmus výpočtu izoplochy :	**Marching Tetrahedra 6**		
Jméno	*Rozlišení*	*Čas generování [s]*	*Počet trojúhelníků*
syn64.vol	***64x64x64***	1,14317	54 152
Ctmayo.vol	***128x128x128***	5,14317	609 206
Hplogo.vol	***236x150x64***	5,47722	797 972
3dhead.vol	***256x256x109***	14,14317	1 281 812
Cthead.vol	***256x256x108***	17,47722	1 755 590
Engine.vol	***256x256x108***	18,47722	1 844 436
Mrbrain.vol	***256x256x109***	24,12100	2 421 418
Mrimix.vol	***256x256x180***	62,14317	5 048 012

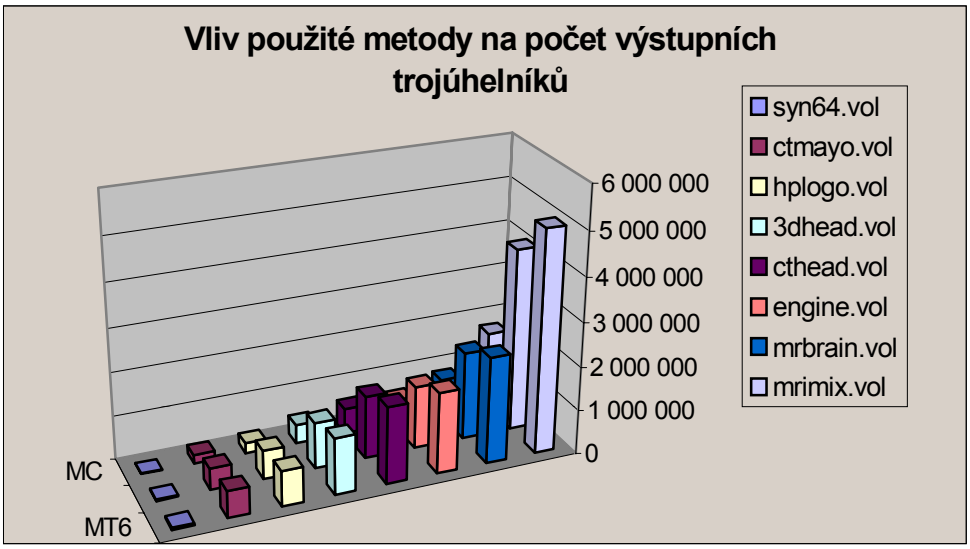
9.2 Doba generování izoplochy v závislosti na objemu dat

Algoritmus výpočtu izoplochy :	**MC**	**MT5**	**MT6**
Jméno	*Rozlišení*	*Čas generování [s]*	*Čas generování [s]*	*Čas generování [s]*
syn64.vol	***64x64x64***	0,36000	1,14317	1,14317
ctmayo.vol	***128x128x128***	1,47722	5,47722	5,14317
hplogo.vol	***236x150x64***	1,47722	5,47722	5,47722
3dhead.vol	***256x256x109***	3,14317	13,12000	14,14317
mrimix.vol	***256x256x180***	10,50000	55,47722	62,14317



9.3 Počet polygonů izoploch v závislosti na použité metodě

Algoritmus výpočtu izoplochy :		**MC**	**MT5**	**MT6**
Jméno	*Rozlišení*	*Počet trojúhelníků*	*Počet trojúhelníků*	*Počet trojúhelníků*
syn64.vol	***64x64x64***	14 796	41 592	54 152
ctmayo.vol	***128x128x128***	196 206	487 792	609 206
hplogo.vol	***236x150x64***	232 552	653 128	797 972
3dhead.vol	***256x256x109***	433 768	1 049 994	1 281 812
cthead.vol	***256x256x108***	576 190	1 423 758	1 755 590
engine.vol	***256x256x108***	556 012	1 428 968	1 844 436
mrbrain.vol	***256x256x109***	838 424	1 990 444	2 421 418
mrimix.vol	***256x256x180***	1 742 204	4 156 168	5 048 012



10 Poděkování

Především bych chtěl poděkovat jak panu prof. Václavu Skalovi za jeho velmi cenné rady při vývoji vhodné implementace algoritmů a datových struktur, tak i ostatním kolegům diplomantům a pracujícím na projektu, kteří pracují na dalších modulech pro MVE i na samotném MVE na Západočeské Univerzitě v Plzni. Také děkuji za jejich návrhy a připomínky, které vedly k nesčetným vylepšením – hlavně modulu rendereru.

11 Závěr

Tato práce byla zaměřena na implementaci modulů pro práci s volumetrickými daty pod MVE systémem. Moduly pro generování izoploch metodami Marching Cubes a Marching Tetrahedra 5-6 společně s rendererem a ostatními pomocnými moduly byly předvedeny a implementovány.

S těmito moduly mohou být tvořeny jednoduché i složitější aplikace pod systémem MVE, ale také není vyloučeno i jiné použití - jedná se o DLL knihovny, které exportují funkce použitelné při dodržení určitých pravidel a datových struktur například v samostatně spustitelných programech.

Pod MVE je možné vkládat další moduly vyrobené ať již v rámci projektu či jinými uživateli MVE např. mezi Loader a Marching Cubes, které nějakým způsobem filtrují volumetrická data od přítomných rušivých elementů (např. odrazy od plomb pacienta), či mezi Marching Cubes a Renderer pro úpravu výstupní trojúhelníkové sítě – např. decimace, která zmenší počet trojúhelníků izoplochy s určitou chybou a tím ovšem zrychlí vykreslování v modulu rendereru.

Z tohoto hlediska se jeví nejen moduly ale i celý systém MVE jako velmi vhodný nástroj pro vývoj nových metod a jejich jednoduchého začleňování do funkční spustitelné aplikace.

12 Literatura

[1] Zara, J.: Moderni pocitacova grafika, Computer press, 1998

[2] Csebfalvi,B.: An Incremental Algorithm for fast Rotation of Volumetric Data, TR-186-2-99-07, 1999

[3] Chan, S., Purisima, E.: A New Tetrahedra Tesselation Scheme for Isosurface generation, Computers&Graphics, Vol.22, No.1., pp.83-90, 1998

[4] Bartos,P.(supervisor Skala,V.): Volumetric Data and Surface Models (in Czech), MSc. Thesis, Univ.of West Bohemia, Plzen, Czech Republic, 1999.

[5] Baxa,P., Skala,V., Moucek,R.: Error Estimation for Iso-surfaces, In Proceedings of COMPUGRAPHICS '97, 1997.

[6] Treece, G., Prager, R., Gee, A.: Regularised Marching Tetrahedra: improved iso-surface extraction, Computers&Graphics, Vol.23, pp.583-598, 1999

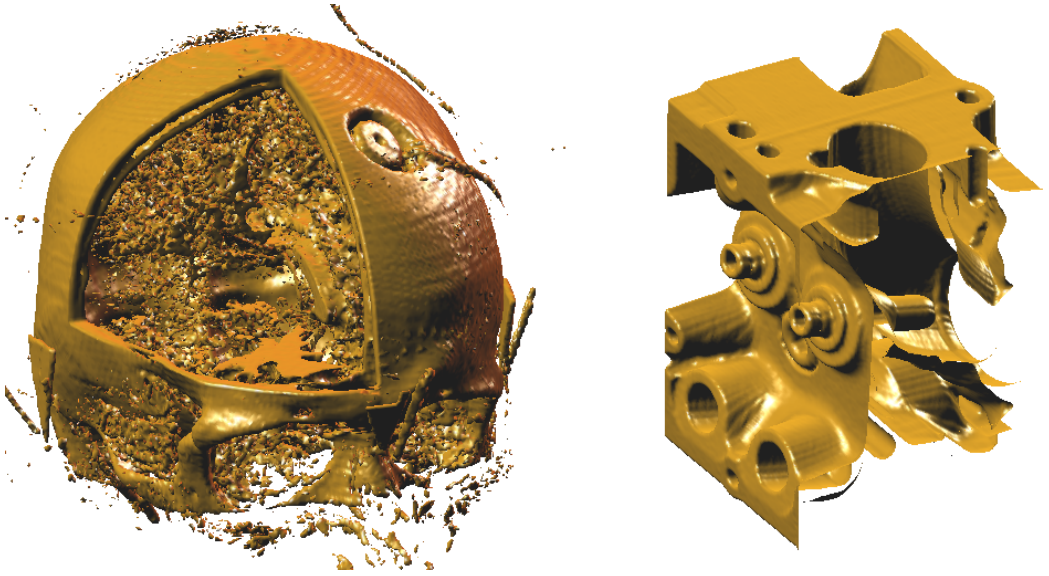
[7] Bailey, M., Clark, D.: Using ChromaDepth to obtain Inexpensive Single-image Stereovision for Scientific Visualization, San Diego Super Computer Center, Journal of graphics tools

13 Přílohy

13.1 Výstupní izoplochy jednotlivých metod

13.1.1 Výstup algoritmu Marching Cubes

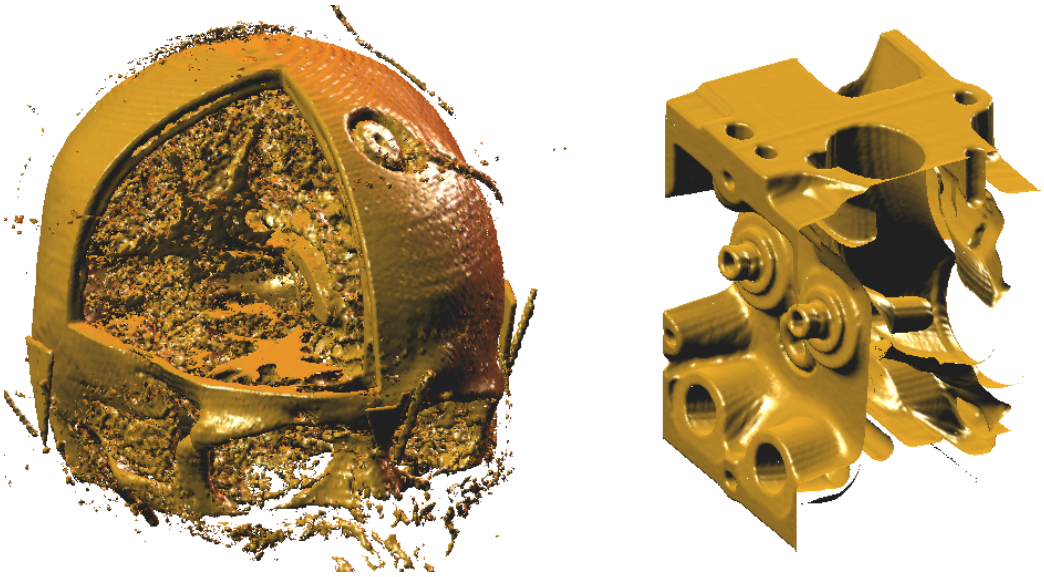
Obrázek 15: cthead.vol (práh 40%), ctmayo.vol (práh 40%)



Obrázek 16: mrimix.vol (práh 43%), engine.vol (práh 40%)

13.1.2 Výstup z algoritmu Marching Tetrahedra 5

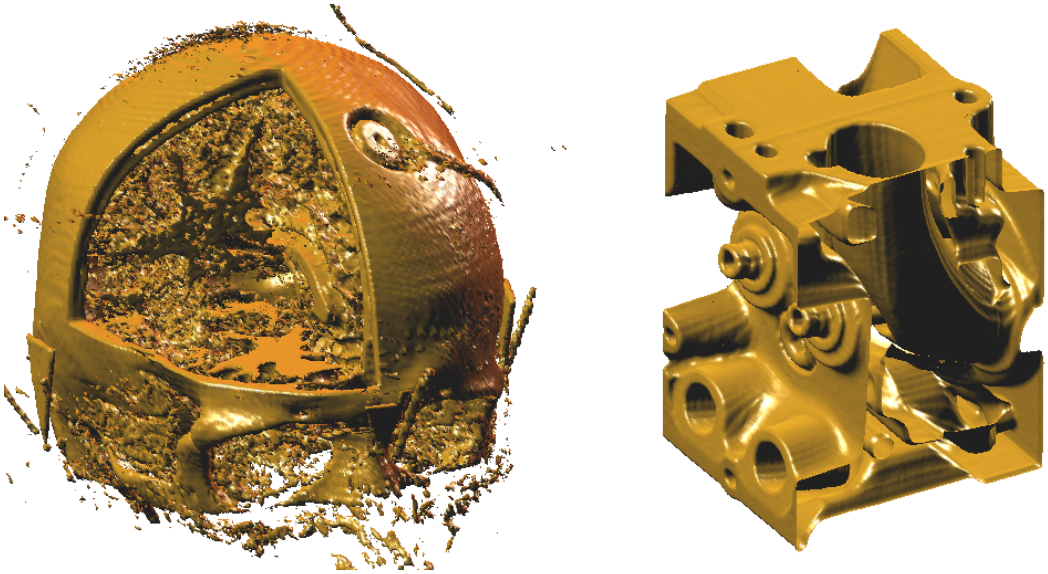
Obrázek 17: cthead.vol (práh 40%), ctmayo.vol (práh 40%)



Obrázek 18: mrimix.vol (práh 43%), engine.vol (práh 40%)

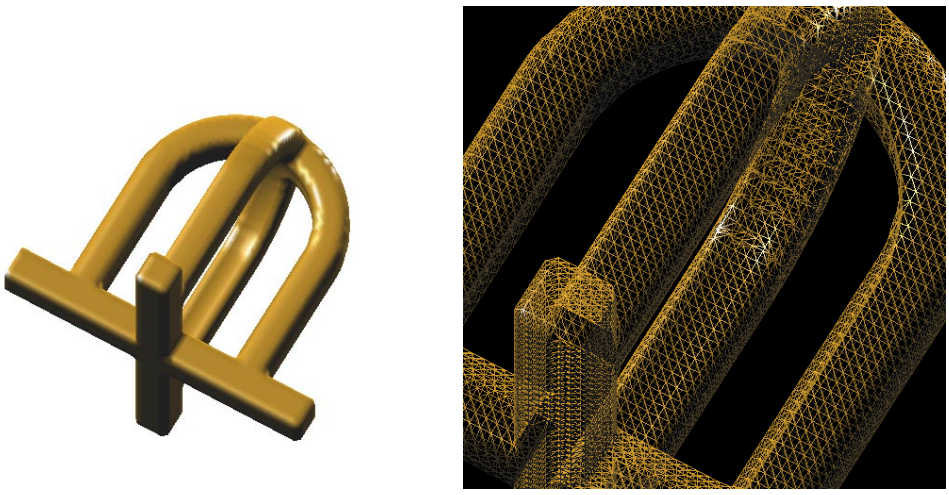
13.1.3 Výstup z algoritmu Marching Tetrahedra 6

Obrázek 19: cthead.vol (práh 40%), ctmayo.vol (práh 40%)

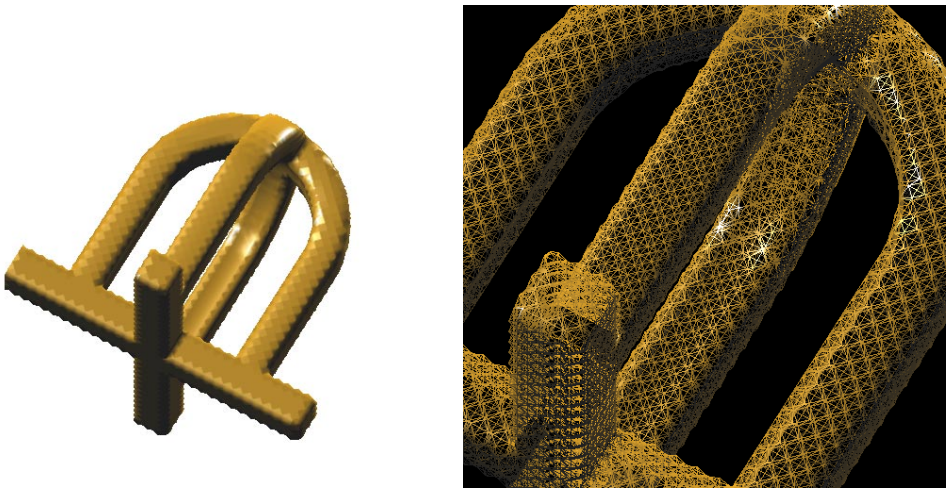


Obrázek 20: mrimix.vol (práh 43%), engine.vol (práh 40%)

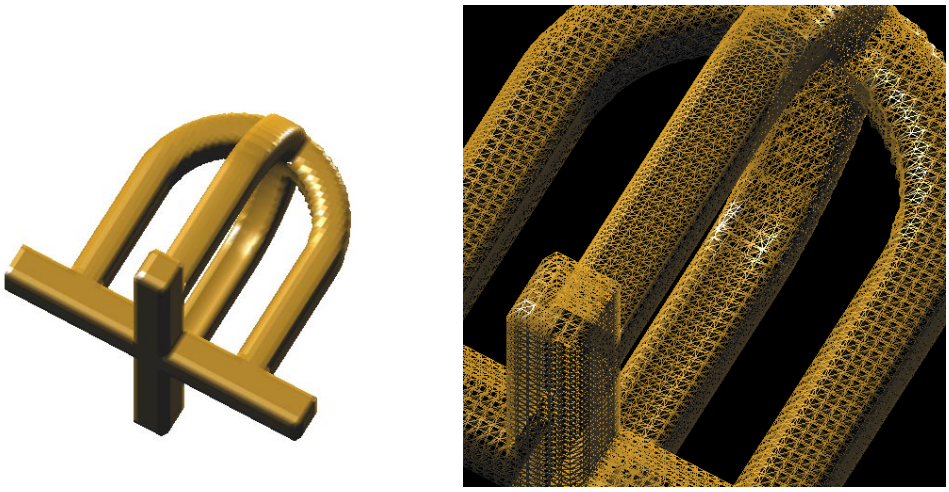
13.2 Porovnání výstupů metod



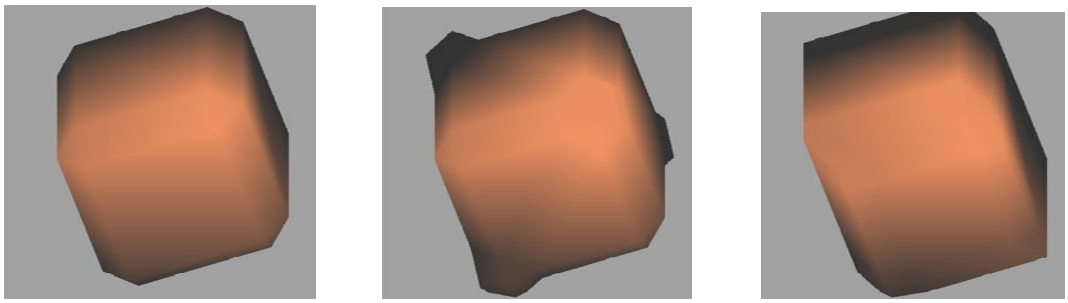
Obrázek 21:syn64.vol - Marching Cubes



Obrázek 22: syn64.vol - Marching Tetrahedra 5

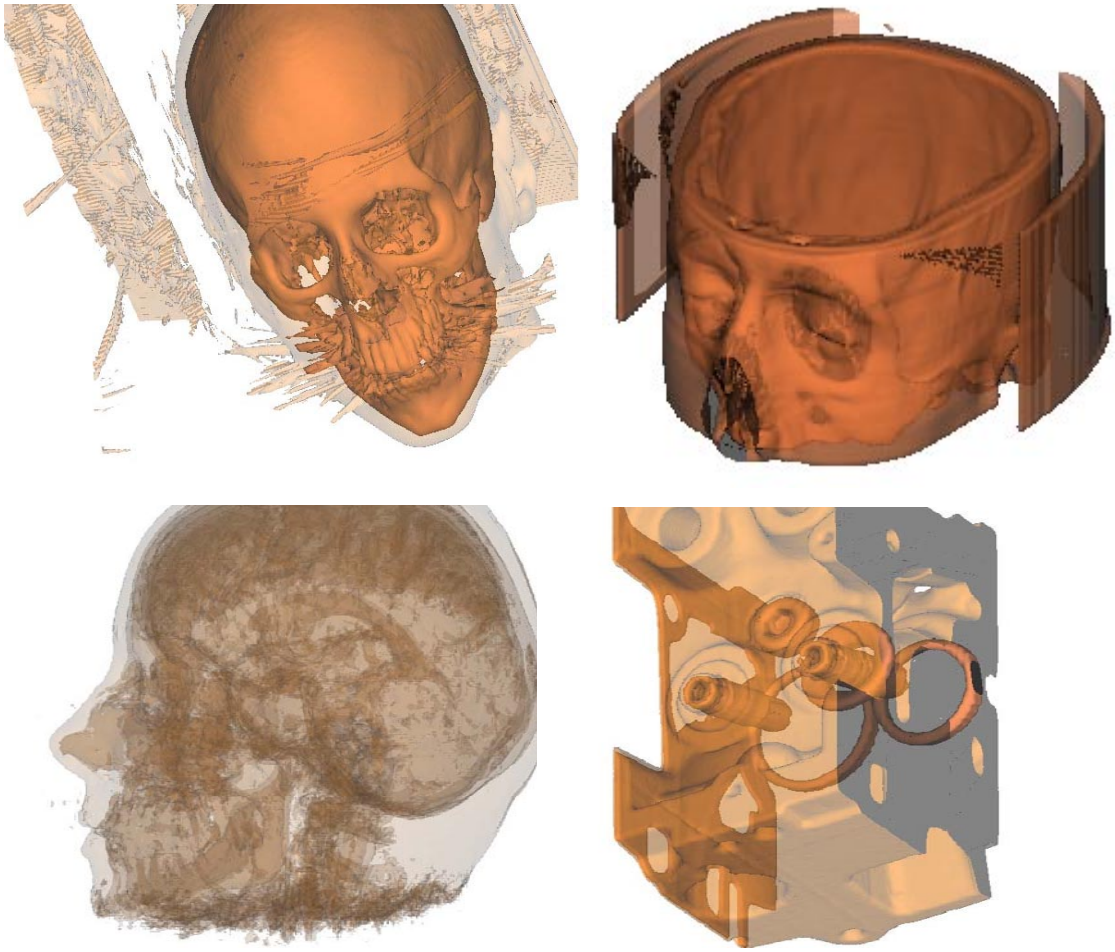


Obrázek 23: syn64.vol - Marching Tetrahedra 6



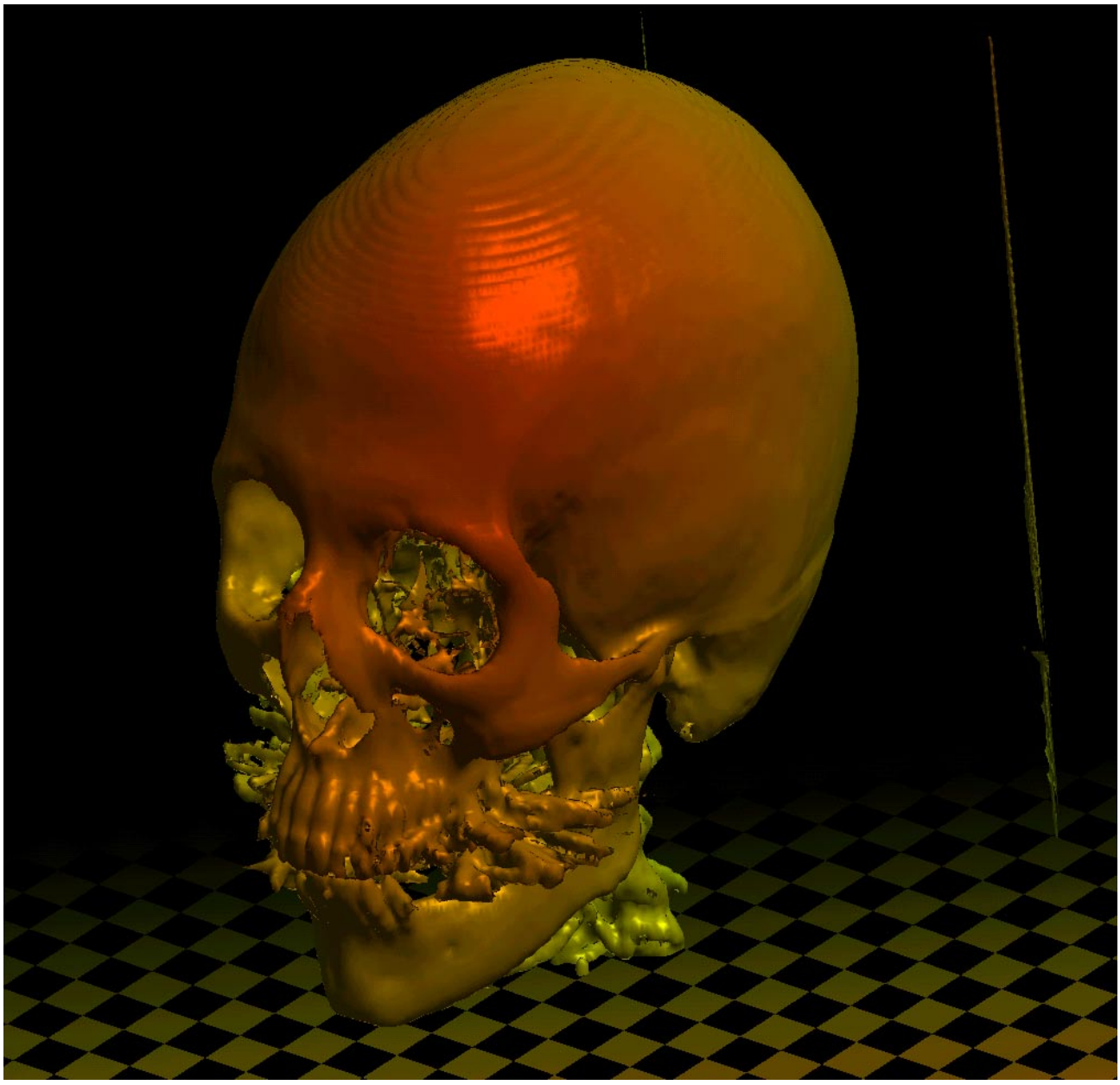
Obrázek 24: Syntetická krychle postupně počítaná algoritmy MC, MT5, MT6

13.3 Výstup programu MFCVolVis



Obrázek 25: Dvě izoplochy kreslené přes sebe s průhledností (MFCVolVis). Postupně jsou to tyto data: cthead.vol, ctmayo.vol, 3dhead.vol, engine.vol

13.4 CHROMADEPTH efekt



Obrázek 26: Ukázka 3D obrázku vykresleného pomocí ChromaDepth efektu. Od předu do zadu ve směru pohledového vektoru je obarven barevným spektrem od červené až po modrou barvu

13.5 Organizace CD

CD disk přiložený k této práci obsahuje následující:

- Volumetrická data testovaná moduly pro generování izoploch
- Trojúhelníková data ve formátech STL a TRI používaná k testování rendereru
- Textury použitelné v Rendereru
- Zdrojové soubory Volume_Modules, Triangle_Modules, MFCVolVis
- Aplikace MVE, MFCVolVis
- DLL knihovny Volume_Modules, Triangle_Modules
- Výstupní obrázky ve formátu TIFF – bez ztráty kvality
- Animace ve formátu AVI souboru (ukázka CHROMADEPTH stereo efektu)
- Použité materiály

Data a materiály však jsou k dispozici v úplném tvaru pouze na originálním CD, které je k dispozici na ZČU. Volumetrická data jsou roztříděna do dvou skupin – 1byte a 2byte, podle druhu těchto dat. Trojúhelníková data jsou rozdělena na STL a TRI. Kompletní data jsou uložena v adresářích ***/private/data/vol, /private/data/tri resp. /private/data/stl***. Nekompletní data určená pouze pro ukázku jsou v podadresáři ***/public/data***, se stejnou strukturou jako předchozí.

Textury jsou používány rendererem pro generování např. efektu CHROMADEPTH – barevné přechody, nebo environmental mapování. Jsou k dispozici v adresáři ***/public/data/textures***.

Zdrojové tvary programů jsou rovněž k dispozici pouze na originálním CD v adresáři ***/private/src***, neboť jsou majetkem ZČU. Jsou rozděleny do tří projektů Visual C++ 6.0:

- Volume_Modules – DLL určeny pro práci s volumetrickými daty
- Triangle_Modules – DLL určeny pro práci s trojúhelníkovými daty
- MFCVolvis – samostatný program určený pouze pro testování implementovaných metod

Tyto jsou pak také k dispozici v binární – spustitelné podobě. Lze je nalézt v adresáři ***/public/bin***. Zde je také spustitelná verze systému MVE.

V adresáři ***/public/rendered*** se nachází výstupní soubory algoritmů Marching Cubes, Marching Tetrahedra – podadresář **volume**. Dále jsou zde renderovaná trojúhelníková data v adresáři ***triangle*** a v neposlední řadě je zde i animace s renderovaná s využitím ChromaDepth efektu (3D) a Marching Cubes algoritmu (cthead.vol).

13.5.1 Struktura adresářů

- **/papers** - některé referenční texty
  - ***my***
  - ***other***
- **/public** - obsahuje vše, co je volně použitelné
  - ***data*** - obsahuje pouze ukázková data
  - ***bin*** - spustitelné obrazy aplikací a DLL knihovny
    - mve
    - volume_modules
    - triangle_modules
    - mfcvolvis
  - ***rendered*** - renderované obrázky, výstupy metod, animace
    - *volume*
      - *mc*
      - *mt5*
      - *mt6*
    - *compare* - obrázky pro porovnání metod generování izoploch
    - triangle - obrázky různých trojúhelníkových sítí
    - animation - CHROMADEPTH animace , v různých rozlišeních
- **/private** - vše co není volně dostupné (zdrojové soubory), data
  - ***data***
  - ***src***
    - volume_modules
    - triangle_modules
    - mfcvolvis

13.6 Programátorská dokumentace

13.6.1 Rozhraní MVE

Pro implementaci rozhraní MVE prostředí byly do projektů převzaty následující hlavičkové soubory:

- *MVE_Include.h*
- *ModuleObj.h, ModuleObj.cpp*

13.6.2 Definice datových struktur

Všechny datové struktury jsou definovány v následujících souborech a jsou platné pro všechny moduly (Volume_Modules i Triangle_Modules) i pro spustitelnou aplikaci MFCVolvis.

- *Types.h* - definice základních datových typů
- *Volume_types.h* - definice typu volumetrických dat
- *Triangle_types.h* - definice typu trojúhelníkových dat

13.6.3 Volume_Modules

Moduly pro výpočet trojúhelníkových sítí z volumetrických dat jsou realizovány jako DLL knihovna, která je kompatibilní s definovaným rozhraním MVE.

Každý modul se skládá z pěti hlavních funkcí popsaných v kapitole 6.2. Ve výsledné DLL knihovně je pět modulů:

- VolumeLoader
- MarchingCubes
- MarchingTetraheda5
- MarchingTetraheda6
- A pouze pokusně také MarchingTetrahedaI – implementace Body centered mechanismu

Celé rozhraní těchto modulů je implementováno v souboru *Volume_Modules.cpp*. Jsou zde tedy implementovány všechny funkce „viditelné“ z DLL knihovny systémem MVE. Tyto funkce jsou zapsány také v souboru *Volume_Modules.def*, kde jsou definovány všechny exporty DLL knihovny. Jednotlivé výkonné funkce jsou implementovány v souborech *LoadVolumetricData.cpp*, *LoadVolumetricData.h**,*** *MarchingCubes.cpp*, *MarchingCubes.h*, *Tetra5.cpp*, *Tetra5.h* resp. *Tetra6.cpp* a *Tetra6.h*.

Všechny tyto výpočtové funkce využívají metod definovaných v *CommonVolume.cpp*, *CommonVolume.h*.

Tímto jsou vyčerpány všechny zdrojové soubory DLL knihovny Volume_Modules.

13.6.4 Triangle_Modules

Knihovna Triangle_Modules je koncipována podobně jako předchozí knihovna. Jako hlavní implementační zdrojový soubor je zde *Triangle_Modules.cpp. Zde* jsou opět definovány všechny exportované funkce DLL knihovny *Triangle_Modules.dll*. Exportované funkce je možné zjistit i ze souboru *Triangle_Modules.def*.

Funkce pro práci s trojúhelníkovými daty, jako třeba čtení z disku, ukládání na disk, vykreslování pomocí OpenGL jsou implementovány v souborech *triangle_utils.cpp, triangle_utils.h*.

Ostatní zdrojové soubory se týkají především modulu ***Renderer***. Jelikož implementace tohoto modulu je silně závislá na použitém hardware a systému, je tento modul jako jediný psaný za použití jazyka C++ a nadstavby jazyka Visual C++ - MFC. Tyto knihovny umožňují do jisté míry visuálně vytvářet aplikaci a hlavně její komunikační rozhraní člověk versus počítač. Jelikož celé rozhraní MVE je psáno právě s těmito knihovnami, je také modul rendereru psán podobně.

K pochopení struktury ostatních zdrojových souborů je třeba základních znalostí Visual C++ a MFC.

Objekty použité v modulu ***Renderer*** jsou potomci objektů CDialog – hlavní okno rendereru, CPropertySheet – panel nastavování parametrů vykreslování scény, atd. Se základní znalostí MFC není problém pochopit všechny ostatní zdrojové soubory.

13.6.5 MFCVolVis

Program MFCVolvis je jakýmsi předchůdcem všech předchozích projektů a byl určen pouze k testování implementovaných metod. Není jej tedy možné brát za plnohodnotnou aplikaci i když některé jeho vlastnosti se jí blíží. Těmi vlastnostmi je myšleno například schopnost tisku vypočteného obrázku na tiskárně, schopnost zobrazovat dvě izoplochy přes sebe, schopnost velmi jednoduché animace. Hlavní význam této aplikace však nadále zůstává testování metod – rychlost výpočtu i rychlost vykreslování.

Jelikož program testuje především metody generování povrchů z volumetrických dat, přejímá tyto funkce z knihovny *Volume_Modules*. Všechny funkce pro výpočet izoploch (MC, MT5, MT6) jsou tedy stejné jako v tomto modulu. Ostatní zdrojové soubory se opět, stejně jako v případě rendereru, týkají rozhraní MFC a komunikace člověk počítač.
Z programátorského hlediska tedy ne moc zajímavé.

13.7 Uživatelská dokumentace

13.7.1 Instalace MVE a všech potřebných modulů

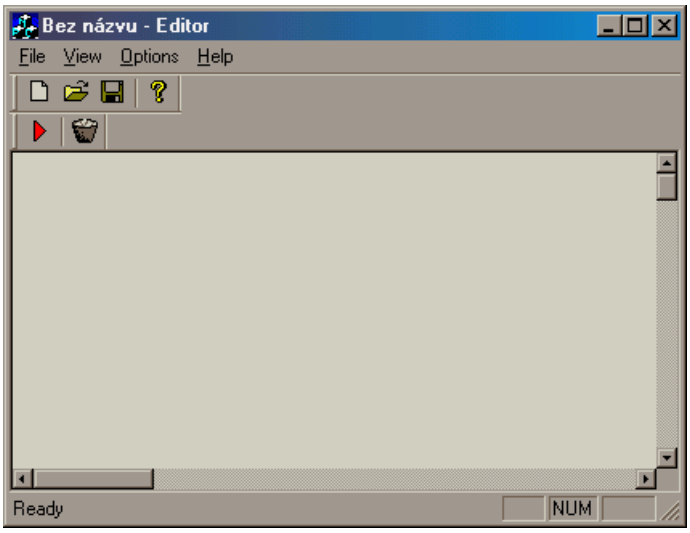
Instalace MVE i všech modulů je velice jednoduchá. Postačí pouze zkopírovat obsah adresáře ***/public/bin*** z přiloženého CD na pevný disk např. do adresáře ***C:/MVE***. Jméno disku je samozřejmě možné změnit podle libosti. Pro rychlejší práci s daty též doporučuji zkopírovat na pevný disk i adresář ***/public/data*** případně i ***/private/data*** např. do adresáře ***C:/MVE/DATA***. Je dobré zbytečně neměnit strukturu podadresářů, neboť DLL knihovny jsou vázány na MVE prostředí a to musí samozřejmě vědět, kde je má hledat.

Po instalaci máme k dispozici na pevném disku následující strukturu adresářů:

- /MVE/DATA - testovací data
- /MVE/MODULES - moduly *Volume_Modules.dll* a *Triangle_Modules.dll*
- /MVE/MVE - spustitelná verze MVE
- /MVE/MFCVOLVIS - testovací aplikace MFCVolVis

13.7.2 Práce s MVE editorem a moduly

Po spuštění souboru ***Editor.exe*** v adresáři /MVE/MVE se objeví následující okno:

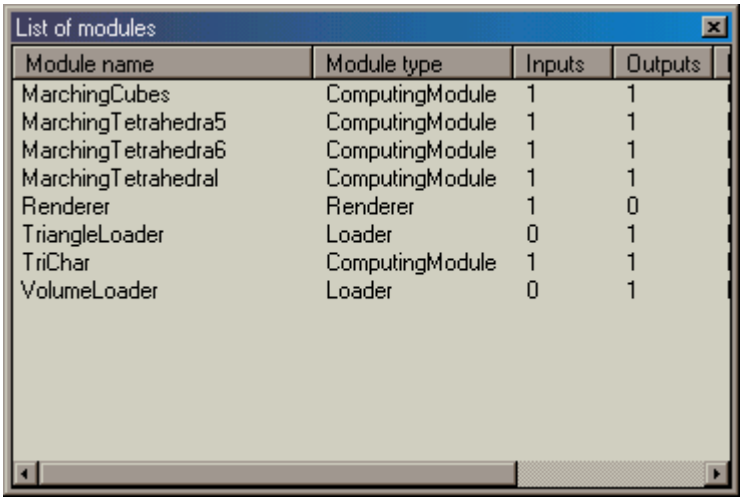


Obrázek 27: Pracovní plocha MVE editoru

Položka v File v menu slouží pro ukládání a načítání navrhované aplikace. To znamená, že pokud vytvoříme nějakou aplikaci pospojováním určitých modulů, je možné ji ukládat na disk a zpětně pak i číst. Celou aplikaci je tedy možné uložit i se současným nastavením všech modulů do souboru **.mve*. Pokud použijeme jako název *souboru default.mve*, bude se tento automaticky načítat při startu MVE editoru. Všechny soubory **.mve* jsou též s aplikací editoru asociovány, takže spuštění editoru lze provést mimo jiné i poklepáním na soubor **.mve*.

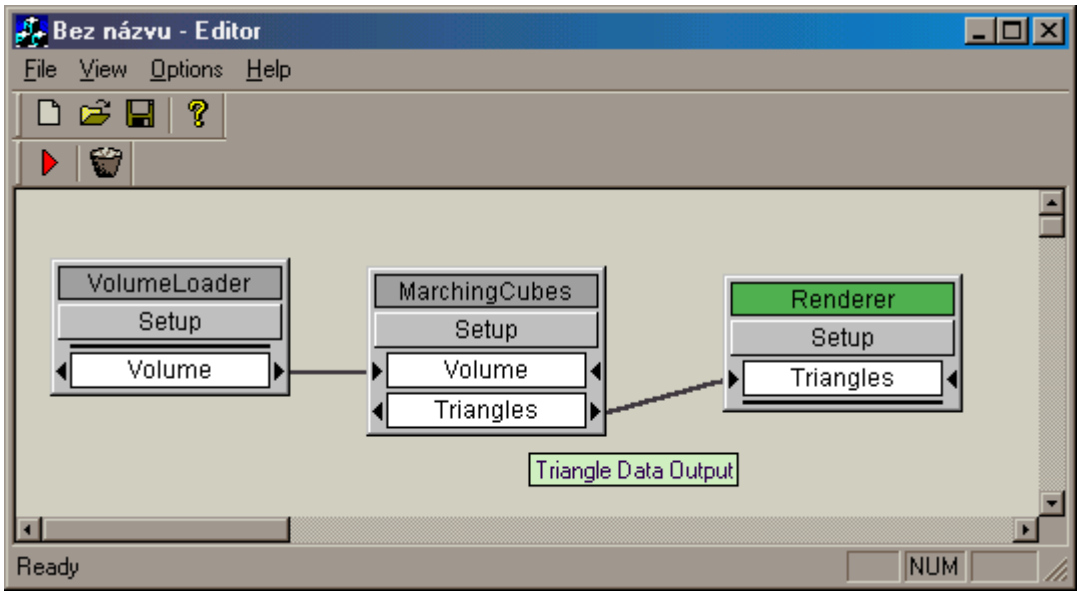
Registrování modulů, respektive knihoven DLL, ve kterých jsou moduly obsaženy, provádíme volbou ***Options/Modules*** v menu. Otevře se dialog, ve kterém je možné nastavit všechny DLL knihovny, ve kterých se mají hledat moduly. POZOR: Je nutné zkontrolovat, zdali zde registrované moduly opravdu existují či nejsou umístěny jinde na pevném disku, než je zde uvedeno.

Vkládání jednotlivých modulů na plochu editoru je možné pomocí volby ***View/Modules***. Objeví se okno, ve kterém jsou všechny moduly, které je možné použít. Pokud máme registrované DLL knihovny *Triangle_Modules.dll* a *Volume_Modules.dll*, pak uvidíme tyto moduly:



Obrázek 28: Zobrazení registrovaných modulů v MVE editoru

Libovolné zobrazené moduly lze zcela jednoduše pomocí myši přetahovat na pracovní plochu editoru a klávesou ***Delete*** je možné tyto moduly opět mazat. Takto si vybereme všechny potřebné moduly a umístíme je na pracovní plochu editoru. Poté spojíme vstupy a výstupy jednotlivých modulů tak, aby výsledná aplikace dávala rozumný smysl. Datové typy jsou kontrolovány. To znamená, že nelze napojit výstup modulu na vstup jiného, pokud se neshodují oba datové typy modulů. Příklad jednoduché aplikace je na následujícím obrázku:



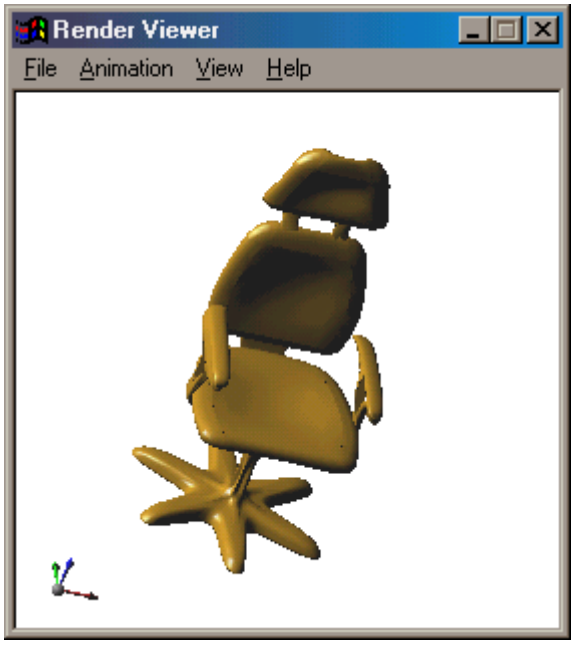
Obrázek 29: Jednoduchá aplikace v MVE

Zde jsou použity tři moduly – *VolumeLoader, MarchingCubes* a *Renderer*. Tyto moduly jsou pospojovány do aplikace, která načte volumetrická data, spočte z nich izoplochu a zobrazí ji v rendereru. Výstup Modulu *VolumeLoader* je tedy kompatibilní se vstupem výkonného modulu *MarchingCubes* – volumetrická data. Stejně tak výstup modulu *MarchingCubes* je kompatibilní se vstupem modulu *Rendereru*.

Parametry jednotlivých modulů je nutné nastavit stisknutím tlačítka ***Setup*** příslušného modulu. V našem případě nastavíme jméno souboru s volumetrickými daty (modul *VolumeLoader*), poté hledanou izoplochu (modul *MarchingCubes*). Renderer nepotřebuje žádná nastavení před spuštěním aplikace, neboť jako jediný modul otevírá okno, ve kterém je možné měnit všechny parametry přímo za běhu aplikace. Výslednou aplikaci spustíme tlačítkem s červenou šipkou.

13.7.3 Práce s modulem Renderer

Jelikož ovládání celého programu není příliš složité a vzhledem k jeho určení se předpokládá „věci znalý“ uživatel, omezíme se zde jen na základní popis. Pokud spustíme v MVE aplikaci, která obsahuje modul rendereru, objeví se okno rendereru se zobrazovaným objektem.

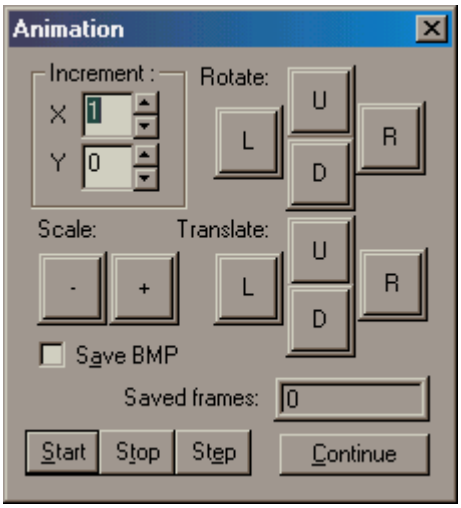


Obrázek 30: Okno Rendereru

Otáčení objektu provádíme myší se současně stisknutým levým tlačítkem. Zvětšení a zmenšení provádíme pohybem myší ve vertikálním směru se současně stisknutým pravým tlačítkem. Při stisknutých obou tlačítkách provádíme posun objektu.

Příkazem ***File/ExportBMP*** v menu můžeme exportovat BMP soubor právě vykresleného obrázku. Celou 3D scénu je možné uložit volbou ***File/Save As …***. Výsledkem je soubor ve formátu STL, nebo soubor TRI.

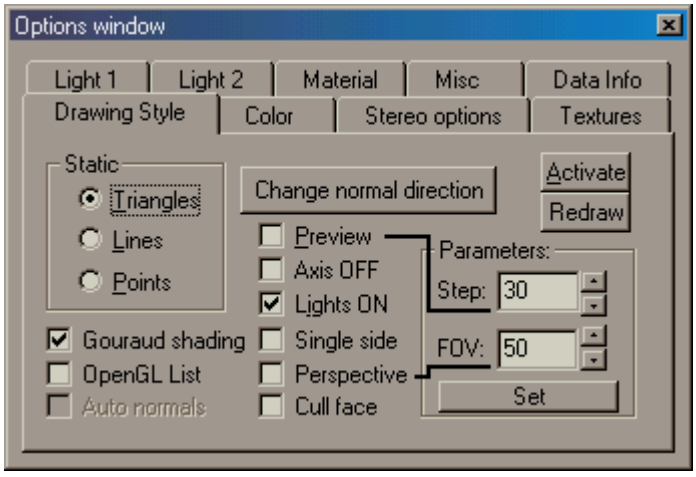
Menu ***Animation*** slouží k tvorbě jednoduchých animací. Pomocí této možnosti byla též tvořena animace na přiloženém CD. Dialog animace je velmi jednoduchý. Získáme jej volbou ***Animation/Animate***.



Obrázek 31: Dialogové okno pro vytváření animací (sady BMP souborů)

Zde je možné nastavit přírůstky rotace v jednotlivých osách scény a tím nechat objekt automaticky rotovat. Za běhu animace je možné měnit velikost objektu i jeho polohu. Rotaci lze zrychlovat zvětšováním přírůstku nebo naopak zpomalovat či zastavit. Volbou ***Save BMP*** provedeme ukládání jednotlivých obrázků do číslovaných BMP souborů. Z těch je pak možné pomocí nějakého animačního software vytvořit například AVI soubor (soubor pro uchování animací a videa). Počet snímků uložených na disk je zobrazován v okénku ***Saved frames***.

Nejdůležitějším dialogem, ve kterém je spousta užitečných nastavení je dialog ***Options,*** který je možné zobrazit volbou ***View/(Un)Hide Options***.



Obrázek 32: Dialog nastavení parametrů Rendereru

Tento dialog obsahuje devět záložek, které obsahují nejrůznější nastavení parametrů zobrazované scény. Jedná se o všechny parametry popsané v kapitole 8.

Záložky obsahují od různých všeobecných nastavení, jako je například způsob vykreslení objektu, některá nastavení OpenGL, přepínač způsobu projekce a další, až po speciální možnosti – např. stereo projekce, texturování, vykreslování „podlahy“ pod objektem a podobně.

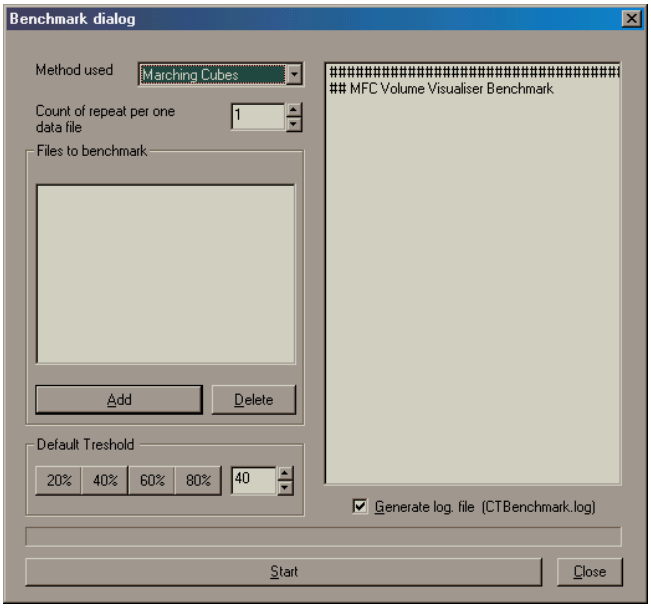
Po provedení příslušných nastavení je nutné změny aktivovat tlačítkem ***Activate*** pro každou záložku zvlášť. Pokud tedy chceme, aby se námi provedené změny provedly, je nutné vždy stisknout toto tlačítko. Pokud zároveň chceme překreslit scénu, je možné stisknout tlačítko ***Redraw***. Tlačítko ***Redraw*** v sobě zahrnuje i funkci tlačítka ***Activate***, a tudíž postačí právě toto tlačítko k aktivování změn v záložce dialogu a následnému překreslení scény.

Tento způsob aktivování změn byl zvolen z jednoduchého důvodu. Občas totiž vykreslujeme obsáhlejší trojúhelníkové sítě, a právě vykreslování takových dat trvá nějakou nezanedbatelnou dobu. Není tedy vhodné překreslovat scénu ihned za každou změnou parametru, ale je lepší nejprve nastavit všechno potřebné a nakonec scénu překreslit.

13.7.4 Použití aplikace MFCVolVis

Ovládání aplikace MFCVolvis je zcela shodné s ovládáním Rendereru popsaného výše. Jelikož se nejedná o plnohodnotnou aplikaci, nejsou zde implementovány všechny volby, které jsou k dispozici v modulu rendereru. Jsou zde ale dvě možnosti navíc. Tou první je možnost generování dvou izoploch a kreslení je přes sebe. Druhá vrstva se kreslí přes první s určitou průhledností. Vzniká tím velice pěkný efekt, kdy například pod průhlednou kůží může být vidět lebka pacienta.

Hlavní význam této aplikace je měření rychlosti implementovaných metod metod. K tomu slouží nabídka v menu ***Benchmark/Set benchmark properties***. V následujícím dialogu je možné nastavit vše potřebné ke spuštění testu výkonosti jednotlivých metod. Je možné určit která data se mají testovat, jakou metodou, kolik opakování měření se má pro každá data provádět a pro jaký práh se budou počítat izoplochy. Výsledek testu je ukládán do souboru *CTBenchmark.log* v aktuálním adresáři a zároveň je vypisován přímo v okně dialogu. V dialogu je též zobrazovaný celkový průběh testu.



Obrázek 33: Dialog pro testování metod MC, MT5 a MT6