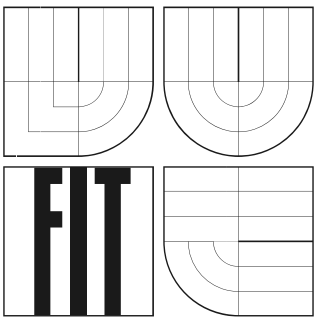
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ



Využití Vertex a Pixel shaderu v OpenGL pro 3D zobrazení 3D obrazových dat v medicíně

Semestrální projekt

2008 **Jiří Vaďura**

Využití Vertex a Pixel shaderu v OpenGL pro 3D zobrazení 3D obrazových dat v medicíně

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Přemysla Kršeka, Ph.D.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……

Jiří Vaďura

4. 1. 2008

Abstrakt

Tato práce se zabývá akcelerovaným zobrazováním 3D medicínských dat, např. z počítačové tomografie, pomocí grafického procesoru a s použitím rozhraní OpenGL. Nezpracované řezy jsou načteny do grafické paměti a zobrazeny metodou ray-casting. Cílem je kvalitně zobrazit 3D data a zároveň umožnit plnou interakci. K dispozici je několik režimů zobrazení jako MIP, simulace rentgenového zobrazení a realistické stínování.

Klíčová slova

medicína, tomografie, grafický procesor, OpenGL, vrhání paprsků, shadery, volume rendering

Abstract

This work deals with accelerated 3D rendering of medical data, e.g. computed tomography, using a graphics processor and OpenGL library. Raw data slices are send to graphic memory and rendered by a ray-casting algorithm. The goal of this project is high quality visual output and full user interaction at the same time. Multiple rendering modes are avaiable to the user: MIP, X-Ray simulation and realistic shading.

Keywords

medical imaging, computed tomography, graphics processing unit, OpenGL, ray-casting, shaders, volume rendering

Obsah

1 Úvod

S příchodem moderních diagnostických zařízení (CT, MRI) nastala i potřeba tato data vhodně interpretovat. Ve většině případů tyto přístroje generují řezy snímaným objektem. Přestože v řadě aplikací je možné analyzovat tyto snímky samostatně – podobně jako rentgenové snímky, v určitých případech je výhodné se na snímaný objekt dívat jako na celek.

Počítačové interpretaci (nejen) medicínských dat se říká volume rendering. Jeho úkolem je zobrazit trojrozměrná data v počítači na 2D zařízení (např. monitor). Jedna z metod volume renderingu je vrhání paprsků. Tyto pojmy budou dále popsány v následujících kapitolách.

S postupným rozvojem grafických procesorů vzrostly možnosti, jak tyto vysoce specializované, ale výkonné, čipy využít pro jiné účely než hry. U poslední řady karet (GeForce 8, 2008) je grafické jádro dostatečně flexibilní k téměř jakýmkoliv obecným výpočtům. V této práci je využita síla grafického procesoru k akceleraci volume renderingu metodou vrhání parsků (ray casting). Použoté grafické rozhraní je OpenGL.

Úkolem je vytvořit aplikaci, která zobrazí 3D objekt na monitoru, a umožní plnou interaktivitu. Ta zahrnuje rotaci objektu, posun a zvětšení. Možné režimy zobrazení budou: MIP, rentgen, segmentace a stínování.

2 Techniky zobrazení objemových dat

Tato kapitola popisuje několik technik používaných při vizualizaci objemových medicínských dat. Výčet není zdaleka kompletní a popisuje jen postupy, které jsem zvažoval při implementaci volume renderingu.

2.1 Texturování

První technika, kterou jsem zvažoval při výběru metody zobrazení 3D dat, byla texturování s použitím blendingu – míchání barev podle průhlednosti. Nejprve se vytvoří sada polygonů, které odpovídají řezům objemem kolmo k pozorovateli (kameře). Na jednotlivé body polygonu se nanesou texturovací souřadnice tak, aby přímo odpovídaly poloze řezu v 3D textuře. Každý bod na polygonu dostane při vykreslování přiřazenu interpolovanou hodnotu intenzity z 3D textury. Z této informace je možné zjistit výslednou barvu pixelu a průhlednost pomocí vyhledávací tabulky. Tyto řezové polygony se v požadovaném směru postupně vykreslují do frame bufferu se zapnutým blendigem. Výsledkem je akumulace informace ze všech vykreslovaných řezů a tedy zobrazení celého objemu.

Tato technika je výhodná z hlediska požadavků na hardware. Je ji možné realizovat téměř na všech grafických akcelerátorech.

2.2 Ray casting

Ray casting jsem zvolil jako základ této diplomové práce. Zobrazení objemu se provede tak, že z každého bodu obrazovky (případně okna, do kterého vykreslujeme) se sleduje paprsek, který prochází tělesem reprezentovaným 3D texturou. Všechny paprsky jsou vyslány rovnoběžně, pokud se jedná o paralelní projekci, a různoběžně v případě perspektivní projekce.

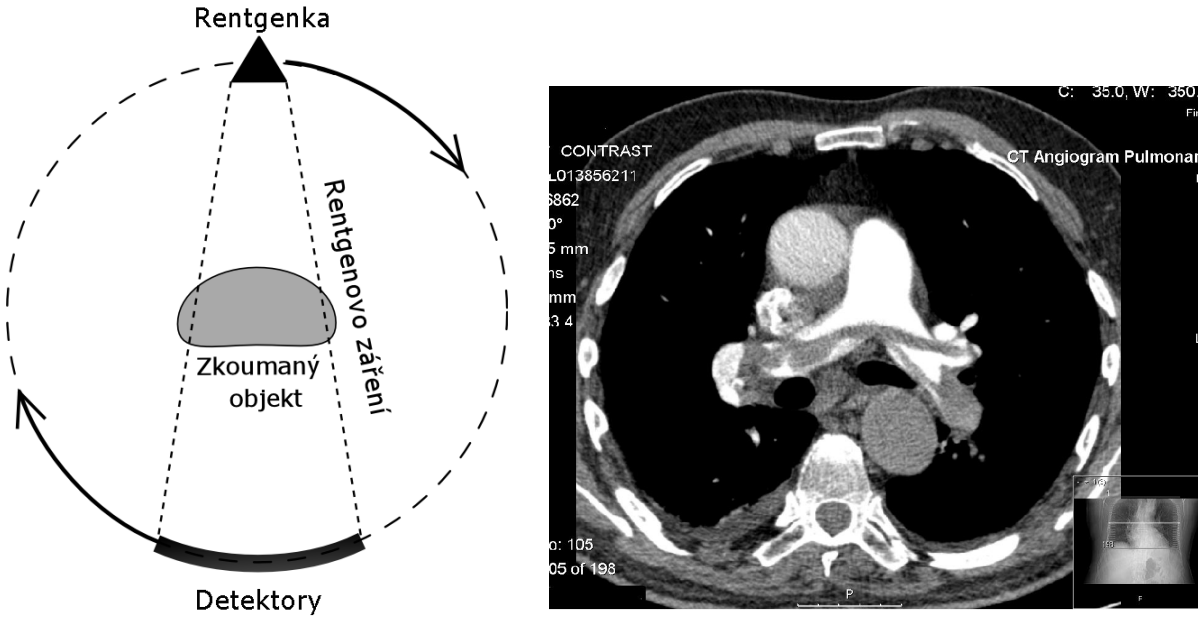
Paprsek po své cestě vzorkuje 3D texturu a akumuluje barvu. Akumulace barvy je větší u méně průhledných částí a naopak. Paprsek (a tím i přičítání nových barev) je tlumen při postupu objemem – opticky neprůhledné části tlumí paprsek více než např. vzduch (např. kolem snímaného člověka na CT skeneru). Paprsek je ukončen v případě, že je zcela utlumen (narazil na kost apod.) nebo pokud opustil 3D texturu.

3 Objemová data v medicíně

3.1 CT snímky

Nejběžnější zdroje volumetrických dat. Rotující rentgenová hlava postupně obkrouží pacienta a z výsledků se dopočítá sada řezů snímaného objektu.

Výsledné řezy jsou obrázky v odstínech šedi, obvykle s 12 bitovou hloubkou (4096 odstínů šedé). Světlá barva reprezentuje oblasti s vysokou absorpcí rentgenového záření (kosti, krev, …), černá barva pak oblasti, kde nedochází k žádné absorpci.



Obrázek 3-1.1 Princip činnosti počítačové tomografie a výsledný řez srdcem

Jednotlivé řezy jsou od sebe vzdáleny řádově milimetry. Kompletní sada snímků hlavy může mít například až 400 řezů. Velikost obrázků závisí na kvalitě použitého skeneru a pohybuje se obvykle v rozmezí 256×256 až 1024×1024. Jeden řez tak může přesáhnout i velikost 1MB.

Vzhledem k tomu, že tato data obsahují pouze informaci o tom, jak dané místo v obrázku absorbuje rentgenové záření, je třeba tato data ještě dále transformovat. V této práci přiřazují každé ze 4096 možných hodnot rentgenové absorpce novou čtveřici čísel – RGBA – každé v 16 bitové barevné hloubce. RGB přestavují barvu, která bude reprezentovat dané množství absorpce a A značí optickou průhlednost. V konečném důsledku vznikne pole 4096 čtveřic RGBA. Tomuto poli se také říká LUT – Look Up Table, případně TF – Transfer Function. Hodnota pro kosti bude blíže konci toho pole, bude mít (obvykle) světlou barvu a hodnota průhlednosti bude nastavena na malou hodnotu.

3.2 Režimy zobrazení

3.2.1 MIP

Zkratka pro Maximum Intensity Projection. Z celé dráhy paprsku se zobrazí pouze nejvyšší hodnota intenzity, kterou paprsek po své cestě 3D texturou navzorkoval. Vzhledem k tomu, že je nutné pouze porovnávat aktuální hodnotu s maximem, je tato metoda nejrychlejší.

Obrázek 3-2 Maximum Intensity Projection

3.2.2 Simulace rentgenu

O něco složitější než MIP. Aby bylo možné simulovat velmi vysoký dynamický rozsah skutečného rentgenu, je třeba zobrazovat data v logaritmické škále.

$$Výstup = \frac{\log \sum_{i=0}^{n} s(i)^4}{\log n}, n = \text{počet vzorků po cestě paprsku}$$

Uvedený vzorec vznikl experimentováním na reálném CT skenu. Výsledný obraz je přirozenější než v případě MIP, ale nese méně detailů.

3.2.3 Stínování

Simulace stínování nemá zásadní diagnostický význam. Je však vizuálně nejzajímavější, protože rekonstruuje strukturu zkoumaného objektu. Zásadní význam má LUT s barvami a průhlednostmi pro jednotlivé hodnoty vstupní intenzity. Pomocí této tabulky lze určit, které hustoty budou zobrazeny určitou barvou a také jejich průhlednost.

Při stínování se kromě stanovení barvy bodu pomocí LUT ještě počítá směr normály a následně Phongovo stínování. Pro výpočet stínování je třeba navíc znát směr světla. Při vhodně zvolené tabulce barev a průhledností je možné dosáhnout fotorealistické kvality zobrazení.

Tento režim je v této práci ještě rozdělen na rychlé náhledové stínování a pomalé kvalitní stínování. Rozdíly jsou např. v přesnosti výpočtu normál a kvalitě osvětlení. Zrychlení je nutné pro zachování interaktivity s modelem na pomalejších grafických kartách.

Obrázek 3-3 Režim stínování

3.2.4 Segmentace

Jedná se o modifikaci metody stínování. Druhá 3D textura, obsahující jednobitové informace o hranách, je zapojena do výpočtu barvy pixelu. Segmentační algoritmus může najít hrany v datech a ty se poté během ray castingu zvýrazní. Tento postup není omezen pouze na zvýraznění hran, může také sloužit k označení oblasti zájmu, která může být zvýrazněna oproti okolí. Další výhodou značkování je snížený výpočetní náročnosti algoritmu, protože náročné počítání normál a osvětlení může být ve většině nezajímavých dat zcela vypnuto.

4 Akcelerace volume renderingu

Vzhledem k velkému objemu zpracovávaných dat a faktu, že je tato data nutné při každém zobrazení výsledného snímku znovu celá zpracovat, vzniká potřeba urychlení výpočtu. V běžném počítači jsou pouze dvě možnosti: hlavní procesor (CPU) a grafická karta (GPU). Výkon grafických karet ve specifických případech až stonásobně převyšuje možnosti CPU. Z tohoto důvodu se v této práci zaměřím na akceleraci volume renderingu pomocí GPU.

4.1 Moderní grafické procesory

Algoritmus vrhání paprsků vyžaduje použití cyklu, ve kterém paprsek cestuje 3D texturou z jedné strany na druhou. Při rozlišení dat 512×512 a faktem, že je potřeba každý voxel vzorkovat alespoň dvakrát, je zřejmé, že takový cyklus musí být schopný iterovat přibližně tisíckrát.

Vývoj programovatelných grafických procesorů byl velmi rychlý a první sériově vyráběný procesor, který byl schopný vrhání paprsků zvládnout, byl NV40 od firmy NVIDIA. Tento procesor zavedl nový Shader Model verze 3. Klíčové vlastnosti byly:

- cyklus omezen na 255 průchodů, ale s možností vnoření
- možnost předčasně ukončit cyklus (příkazy break a continue v jazyce C)
- podpora 3D textur do velikosti 512^3
- shader může mít délku až 512 instrukcí
- celkový počet provedených instrukcí max. 65535
- počet obecných registrů 32

Právě tyto vlastnosti umožnily implementaci skutečného vrhání paprsků 3D texturou. I když bylo zrychlení oproti CPU velké, čip stále algoritmus zásadně omezoval. Většina limitů (viz výše) byla stále ještě malá, aby umožnila plnohodnotnou a efektivní implementaci. Další problém představoval způsob, jakým jsou jednotlivé paralelní jednotky spuštěny. Grafické karty byly určeny převážně pro akceleraci her, kde se předpokládá, že sousední body budou podmíněné výrazy a cykly procházet stejně. Blok několika desítek pixelů bude vždy provádět úplně stejný kód, musí se tedy přizpůsobit „nejpomalejšímu“ pixelu a snižuje tak efektivitu právě u výpočtu jakým je vrhání paprsků. Grafický procesor je navíc vysoce paralelní a každá jednotka pracuje vždy se čtyřmi čísly zároveň. Tento fakt je výhodný pro zpracovávání barevných textur, ale u vrhání paprsků zůstává část výkonu nevyužita.

S příchodem čipu G80 (NVIDIA) byla většina těchto problémů odstraněna. Všechny limity vzrostly vysoko nad potřeby ray castingu a výkon hardwaru je možné plně využít. Při porovnání podobných grafických karet série GeForce 6600 a GeForce 8600, dochází k až desetinásobnému nárůstu výkonu ve prospěch novější karty.

Karty na bázi G80 přinesly další novinku – celočíselné operace. Doposud bylo možné v shaderech pracovat pouze s čísly typu float (32bitů). Zavedení bitových operací zrychlilo zobrazení režimu segmentace o tisíce procent.

4.2 Možnosti shaderů

Shadery jsou malé programy psané v asembleru nebo jazyce C. Po zavedení do grafické karty se stejný program spouští paralelně pro každý pixel. Omezení paralelizace vychází z počtu jednotek, které jsou na grafickém čipu k dispozici. Algoritmus vrhání paprsků je tak psán pouze pro jediný paprsek a grafický čip tento program spustí pro každý výstupní obrazový bod.

Pro tuto práci jsem zvolil jazyk GLSL (GL Shading Language). Jedná se o nativní jazyk pro OpenGL 2.0. Při použití tohoto jazyka není nutná žádná další knihovna – překlad probíhá v ovladači grafické karty. GLSL je podobný jazyku Cg od NVIDIA, který ale vyžaduje použití další knihovny. Jazyk Cg však nabízí některé funkce (rychlejší datové typy, specializované funkce), které GLSL postrádá. Ovladače NVIDIA umožňují použití těchto Cg datových typů a funkcí v rámci jazyka GLSL. Pro karty jiných výrobců je možné tyto chybějící funkce emulovat pomocí vestavěných GLSL typů s použitím preprocesoru jazyka C.

Jednotlivé zpracovávané pixely nemají žádnou možnost komunikace s okolními body. Jediný způsob, jak předat shaderu informace je pomocí konstant a textur. Konstanty se nastavují globálně pro všechny body najednou, stejně jako textury. Shadery však mohou získat některé klíčové informace, jako např. jejich souřadnice v rámci vykreslovaného okna.

Obecně existují tři specializované typy shaderů:vertex shader, geometry shader a pixel shader. Pro potřeby ray castingu plně postačuje pixel shader, který spouští zadaný program pro každý pixel tělesa, které je vykresleno (zde obalové těleso 3D textury). Vertex shader ani geometry shader nejsou použity a jejich funkci přebírá statická pipeline.

5 Návrh řešení

5.1 Základní stavba programu

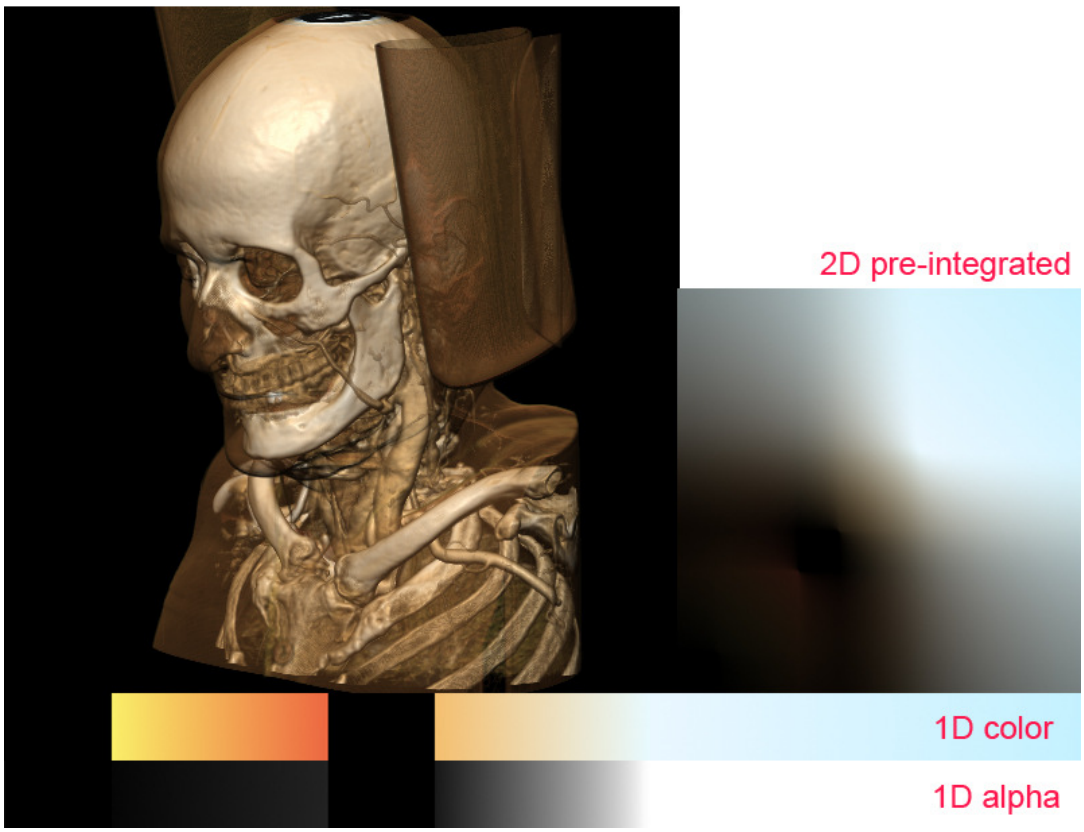
5.1.1 Vstupní data

Maximální možný rozměr a bitová hloubka vstupních dat je limitována pouze aktuálně použitým hardwarem. Běžné rozměry jsou do 512^3 s 16 bitovou hloubkou (pouze prvních 12 bitů je použito). U karet řady G80 je rozměrový limit 2048^3 a hloubka je omezena na 32 bitů.

Všechny řezy jsou beze změny sestaveny do jediné 3D textury, která je celá uložena v grafické paměti. Nad 3D texturou je zapnuto trilineární filtrování, které zajistí hardwarovou interpolaci a dat a zvýšení vizuální kvality. Dopad na výkon je minimální za předpokladu, že jsou použity dostatečně rychlé paměti.

5.1.2 Look-up tabulky

Tyto vyhledávací tabulky je možné použít ve všech režimech zobrazení. Největší dopad mají ovšem při stínování. Převádí 12 bitovou informaci o míře pohlcování rentgenových paprsků na odpovídající optickou barvu a průhlednost. Tyto hodnoty nemusí odpovídat skutečným barvám.



Obrázek 5-1.1 Ukázka 1D LUT, 2D LUT a výsledného výstupu

5.1.2.1 1D tabulky

Každé hodnotě z 12 bitového rozsahu je přiřazena právě jedna barva. Celá tabulka je uložena v grafické paměti jako 1D textura o délce 4096. Každá položka tabulky se skládá ze čtyř 16 bitových hodnot RGBA. Na textuře je zapnuta lineární interpolace pro minimalizaci nežádoucích artefaktů.

5.1.2.2 2D předintegrované tabulky

Při průchodu paprsku objemem dochází nevyhnutelně k vzorkování dat. Při přechodu paprsku z průhledné části textury do neprůhledné dochází na hranici k nežádoucím barevným přechodům. Tyto přechody jsou způsobeny nedostatečně jemným vzorkováním dat. Zmenšení vzorkovacího kroku ale přináší citelné zpomalení výpočtu. Řešením je použití předintegrované vyhledávací tabulky (pre-integrated look-up).

Paprsek si po své dráze vždy pamatuje hodnotu předchozího vzorku. Společně s aktuálním vzorkem tato dvojice tvoří dvě souřadnice do 2D vyhledávací tabulky reprezentované 2D texturou. Zatímco 1D LUT přičítá při vzorkování pouze barevné hodnoty v jednom bodě, kde došlo k navzorkování 3D textury, pomocí předintegrované tabulky je možné přičítat celý úsek, který byl paprskem přeskočen mezi vzorkovacími body. Na pozici, kde se setkají dva indexy v tabulce, je možné nalézt integraci všech hodnot, které v 1D tabulce leží mezi těmito dvěma indexy. V diagonále 2D předintegrované tabulky je uložena původní 1D LUT.

Tato předintegrovaná LUT je schopná pracovat korektně pouze s jednou velikostí vzorkovacího kroku. Pro více délek kroku je třeba použít 3D předintegrovanou tabulku, kde ve třetí dimenzi je právě délka kroku.

$$A_{preint}(s_f, s_b, d) = \frac{d}{s_b - s_f}\Big(T(s_b) - T(s_f)\Big),$$

$$T(s) = \int_0^s \alpha(s)ds, \qquad \alpha = \text{1D alfa LUT}$$

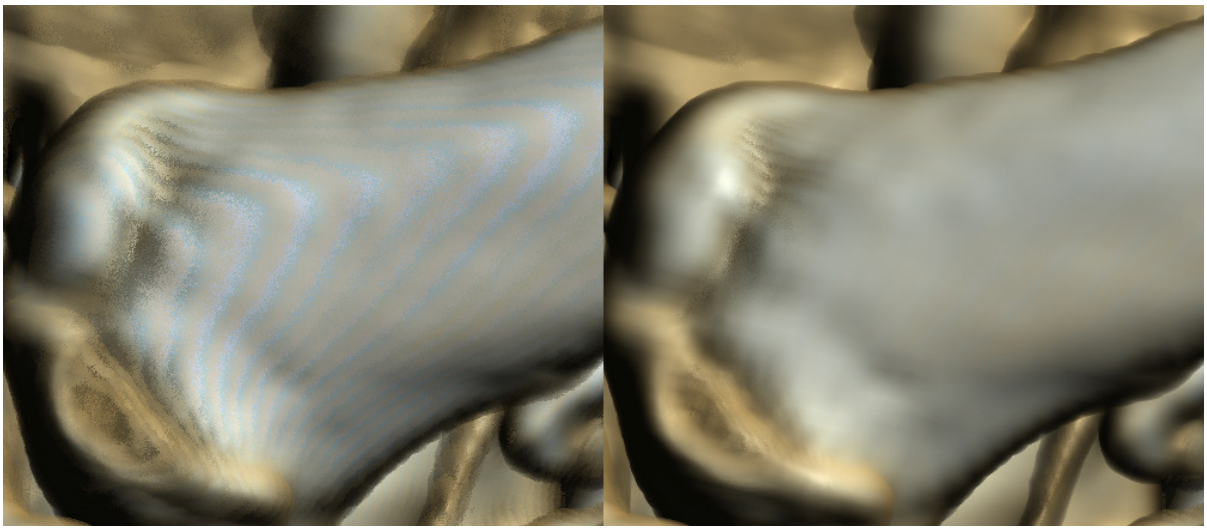
Rovnice 5.1 Výpočet alfa kanálu předintegrované 2D tabulky

$$RGB_{preint}(s_f, s_b, d) = \frac{d}{s_b - s_f}\Big(K(s_b) - K(s_f)\Big),$$

$$K(s) = \int_0^s \alpha(s)\delta(s)ds, \qquad \alpha = \text{1D alfa LUT a } \delta = \text{1D rgb LUT}$$

Rovnice 5.2 Výpočet barvy předintegrované 2D tabulky

Použité 2D tabulky v tomto projektu mají velikost 2048×2048, RGBA, 16 bit na kanál, zapnutá interpolace. Tato velikost je zvolena jako kompromis mezi maximální kvalitou výstupu a velikostí textury (32MB v uvedeném případě).



Obrázek 5-2.2 Srovnání 1D LUT a 2D předintegrované LUT

5.1.3 Ray Casting

Při vrhání paprsků musí být jasné, kde paprsek začíná a kde končí. Jinými slovy chceme znát úplnou dráhu paprsku 3D texturou a to bez ohledu na použitou perspektivu v OpenGL. Můžeme vykreslit libovolný polygonální objekt, opatřit ho korektními texturovacími souřadnicemi a zobrazit se zapnutým texturováním. Výsledkem bude prostorový řez 3D texturou v místech, kde polygony protnou virtuální objem. Další krok by byl vržení paprsku v každém bodě výsledného řezu a tím i zobrazení celého objemu uvnitř polygonálního objektu. Cílem je volumetrické zobrazení pouze té části 3D textury, která se nachází uvnitř toto objektu. Problémem ovšem zůstává určení směru paprsku z tohoto bodu a jeho ukončení v místě, kde polygonální objekt končí na druhé straně.

Řešení je v předběžném vykreslení pouze zadních stran polygonů objektu (ve většině případů se bude jednat o krychli). Toto se dosáhne použitím funkce OpenGL nazvané *culling*. Celá scéna se vykreslí ne na obrazovku, ale do pomocné textury. Do výstupních hodnot RGBA se nebudou ukládat barvy, ale texturovací souřadnice všech bodů odvrácené strany objektu (RGB složky) společně s jejich hloubkou v deph bufferu (A složka). Tuto funkci zajistí velmi jednoduchý pixel shader. Pro maximální přesnost je ideální zvolit výstupná formát jako čtveřici 32 bitových čísel typu float.

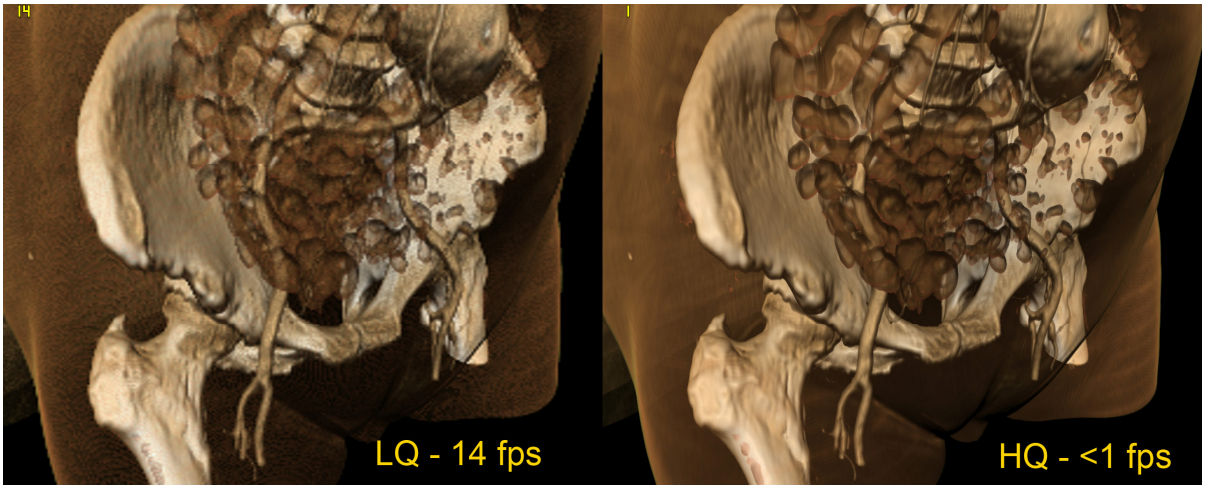
V druhém průchodu algoritmu se vykreslí přední části polygonů společně s aktivním shaderem pro ray casting. Uvnitř shaderu se spočítá přesná dráha paprsku s ohledem na pohled a perspektivu kamery a také hloubka v deph bufferu, kde se paprsek zastavil. Pixel shader zná svoji polohu v okně OpenGL a může tedy přistoupit do textury z předchozího kroku. Ze znalosti výchozí a cílové pozice lze snadno dopočítat směr a délku průchodu.

Díky znalosti hloubky pro začátek i konec paprsku lze uložit do deph bufferu korektní hloubku (odpovídající např. vykreslené kosti uvnitř 3D textury) zastaveného paprsku. Díky tomu se nabízí možnost omezené interakce objektů zobrazených pomocí volume renderingu a polygonálních struktur v OpenGL. Podmínkou je zobrazení neprůhledných struktur, jako jsou kosti apod.

5.1.4 Režimy zobrazení

Protože plná interaktivita a maximální vizuální kvalita jdou proti sobě, rozhodl jsem se implementovat dva režimy zobrazení – rychlý náhledový a pomalý kvalitní.

Při extrémním zvětšení zkoumaného objemu ve velkém okně se zbytečně v každém bodě vrhá paprsek, přestože by stačil pouze zlomek těchto paprsků k dosažení stejného výsledku. Z tohoto důvodu jsem omezil velikost vykreslovacího okna. Výsledek ray castingu se renderuje do textury o zadané velikosti a teprve ta je roztažena přes výstupní okno.



Obrázek 5-3.3 Při velkém zvětšení je výhodnější rychlý režim

5.1.4.1 Režim maximální rychlosti

- výstupní rozlišení je omezeno na 256×256 případně 512×512 bodů
- výpočet normál je proveden pouze z šestice okolních bodů
- nepočítá se spekulární složka Phongova stínování
- používá se pouze 1D LUT
- vzorkovací krok je prodloužen o 50%

5.1.4.2 Režim vysoké kvality

- výstupní rozlišení je typicky 1024×1024 bodů
- výpočet normál 3D Sobelovým operátorem
- plné Phongovo stínování
- předintegrovaná 2D LUT
- vzorkovací krok 2× pro každý bod v textuře (tedy přes 1000 kroků u tex. 512^3)

5.1.5 Zobrazení

Výsledek z volume renderingu je třeba zobrazit do výstupního okna bez ohledu na jeho velikost a na velikost samotného výstupu volume renderingu. Pokud je výstupní rozlišení 256×256 a výsledek má být zobrazen na celé obrazovce je třeba obraz přepočítat na větší velikost. Toho je možné dosáhnout

například nanesením obrázku jako texturu na čtvercový polygon roztažený přes celé výstupní okno. OpenGL nabízí pouze lineární interpolaci, která nenabízí nejlepší možné výsledky.

Z těchto důvodů jsem implementovat bikubickou interpolaci pro přizpůsobení výstupu velikosti okna. Interpolace je řešena jako jednoprůchodový pixel shader, který využívá pomocnou 1D texturu s kernel funkcí (sinc).

5.2 Shadery

Všechny použité pixel shadery sdílejí stejnou základní kostru. Rozdíly jsou pouze v tom, jakým způsobem se zpracovává navzorkovaná 3D textura. Základní shadery jsou popsány v kapitole 3.2, stínování je podrobněji rozebráno v následující podkapitole 5.2.2.

5.2.1 Struktura shaderu

5.2.1.1 Preprocesor

Protože se v shaderech používají funkce a datové typy jazyka Cg, je nutné v preprocesoru ošetřit jejich možné použití na hardwaru jiného výrobce než NVIDIA. S použitím příkazu

```
#ifndef __GLSL_CG_DATA_TYPES
```

je možné názvy těchto funkcí a datových typů předefinovat na hardwaru jiných výrobců.

Druhé použití preprocesoru je v detekci Shader Model 4.

```
#ifndef GL_EXT_gpu_shader4
```

Tento příkaz je použit při definici funkce, která čte bit z druhé 3D textury obsahující informace o segmentaci. Zatímco Shader Model 3 musí tento bit získat složitým výpočtem s čísly typu float, Shader Model 4 a bitové instrukce AND, OR, SHIFT stejnou práci vykonají mnohonásobně rychleji. Díky preprocesoru je možné mít obě verze v jednom souboru a jejich výběr se provádí až při překladu v ovladači grafické karty.

5.2.1.2 Vstupní textury

Volumetrická data

Trojrozměrná trilineárně interpolovaná textura obsahující vykreslované objekty.

Look-up tabulka

1D nebo 2D textura používaná k indexaci navzorkovanou hodnotou.

Souřadnice zadních polygonů

2D textura obsahující souřadnice určující místo ukončení paprsku a hloubku deph bufferu v tomto místě.

Textura se šumem

2D textura o velikosti 1024×1024 obsahují osmibitový šum. Náhodná informace se používá pro malé posunutí začátku paprsku, kterým lze předejít artefaktům v obraze.

Segmentační informace

3D textura s rozměry 8× menšími než samotná data. Pro každý voxel existuje v textuře jeden bit, který udává přídavné vlastnosti pro tento bod. Textura je tedy vždy 16× menší než původní data a je u ní vypnutá interpolace.

5.2.1.3 Vstupní konstanty

Informace, které je nutné předat všem paprskům zahrnují rozměry datové textury, rozměry okna, podmínky ukončení paprsku apod.

5.2.1.4 Výpočet délky kroku

Délka vzorkovacího kroku by měla být z každého pohledu vždy stejná – zhruba dva vzorky na každý voxel. Počet řezů např. CT skenu je většinou jiný než hodnota šířky jednoho řezu. Přesto je každý rozměr v 3D textuře indexován čísly od 0 do 1. Z toho vyplývá, že z různých pohledů bude vzdálenost mezi jednotlivými voxely v textuře různá. Aby byl zajištěn vždy stejně dlouhý skok, je třeba tento skok vypočítat individuálně pro každý paprsek.

Pro tento účel v shaderu existuje funkce `float StepLength(vec3 path)`. Je tedy znám směr paprsku, ale není známa vzdálenost (v souřadném systému texturovacích souřadnic 3D textury) od jednoho voxelu ke druhému v tomto směru. Zmíněná funkce počítá průsečík s rovinami kolmými k osám. Každá rovina je vzdálena od počátku o 1/N, kde N je rozlišení textury v dané ose. Z průsečíků se vybere minimum, které indikuje první – a tedy správný – průsečík. Vzdálenost od počátku k průsečíku je přibližně vzdálenost dvou voxelů ve směru paprsku.

5.2.1.5 Zarovnání všech paprsků vzhledem k pozorovateli

Další technika, která redukuje nepříjemné artefakty způsobené tím, že paprsky začínají na stěnách obalového tělesa. Stěny na sebe navazují v ostrých úhlech a při dlouhém vzorkovacím kroku dochází k jejich zvýraznění. Funkce provádějící zarovnání není prozatím aktivní, protože se ukazuje, že její použití není zásadní. V současném stavu projektu není možné tuto funkci aktivovat v perspektivní projekci kvůli nelineárnosti depth bufferu v tomto zobrazení.

5.2.1.6 Hlavní smyčka

Z délky dráhy paprsku a velikosti kroku je možné vypočítat počet skoků v textuře. I přesto je tento údaj použit pouze k předčasnému ukončení cyklu, jehož počet opakování je pevně nastaven na 2048. Tento postup umožňuje vytvořit jednoduchý cyklus o délce větší než 255 (maximální počet iterací cyklu v Shader Model 3). Překladač může sám rozhodnout, zda použije dva vnořené cykly (SM3)

nebo pouze jeden (SM4). Pomocná proměnná udržuje během cyklu pozici uvnitř textury a pomocí této hodnoty je textura při každém vzorkování indexována.

5.2.2 Stínování

Při režimu stínování se v každém kroku navzorkuje 3D textura interpolované intenzity. Tato hodnota se použije jako index do look-up textury. Pokud je alfa (průhlednost) v tomto místě velmi malá, stínování se přeskočí a cyklus (paprsek) pokračuje dál. Pokud je v tento okamžik paprsek již téměř utlumen (tedy že nasbíral dostatečné množství barvy), cyklus se ukončí.

V opačném případě se pokračuje odhadem normálového vektoru. Podle aktivního režimu se navzorkuje potřebný počet okolních voxelů a opačné strany se odečtou. Normálový vektor se dále normalizuje a je připraven k výpočtu osvětlení.

Následuje Phongovo stínování. Malá modifikace spočívá v násobení alfa hodnoty spekulární složkou, čímž se dosáhne zesílení odlesků na „pevnějších“ površích. Vektor pohledu kamery je stejný jako vektor samotného paprsku. Vektor světla je možné libovolně měnit.

Inverzní hodnota průhlednosti se poté vynásobí s proměnnou útlumu paprsku (tato proměnná začíná na hodnotě 1 a je postupně násobením snižována až do nuly). Aktuální pozice v textuře se posune o délku jednoho kroku a cyklus začíná znovu.

5.2.3 Segmentace

S použitím další 3D textury s jednobitovými informacemi bude možné dramaticky zvýšit výkon a zvýraznit vybrané části objemu (např. lidského těla) přidáním jednoduché podmínky. Data pro segmentaci by měla být generována za pomoci segmentačního toolkitu.

5.3 OpenSceneGraph

OpenScenGraph je knihovna postavená nad rozhraním OpenGL. Pracuje na principu grafu scény a původní OpenGL kód zcela zastiňuje. Volume rendering v pixel shaderech je natolik nízkoúrovňový, že použití této knihovny pouze obaluje základní příkazy v OpenGL, v některých případech velmi komplikovaně.

Volba OSG je dána potřebou začlenit tento projekt do již existující aplikace, která je na OSG založena. Z tohoto důvodu se ve zbylých částech technické zprávy budu zmiňovat pouze o samotném řešení v OpenGL.

6 Současný stav práce

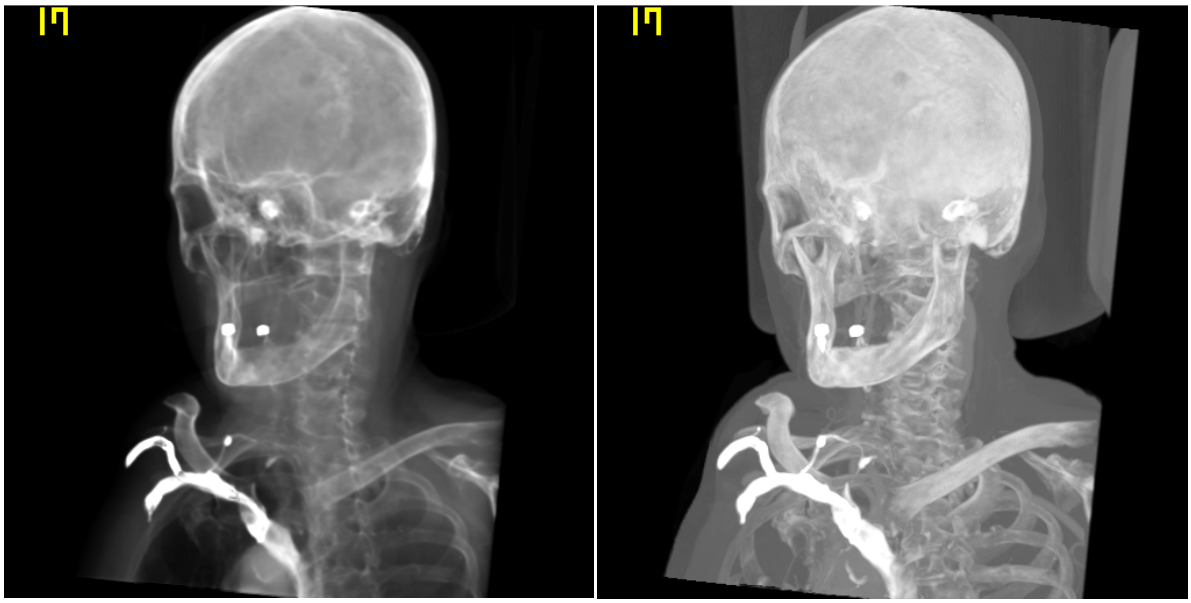
Z kvalitativního hlediska je práce v závěrečné fázi. Zbývá propojení se segmentačním toolkitem a vytvoření grafického uživatelského rozhraní, které bude kromě natavování parametrů volume renderingu, také obsahovat nástroj pro interaktivní kreslení look-up tabulek a generovaní předintegrovaných 2D verzí.

6.1 Možnosti softwaru

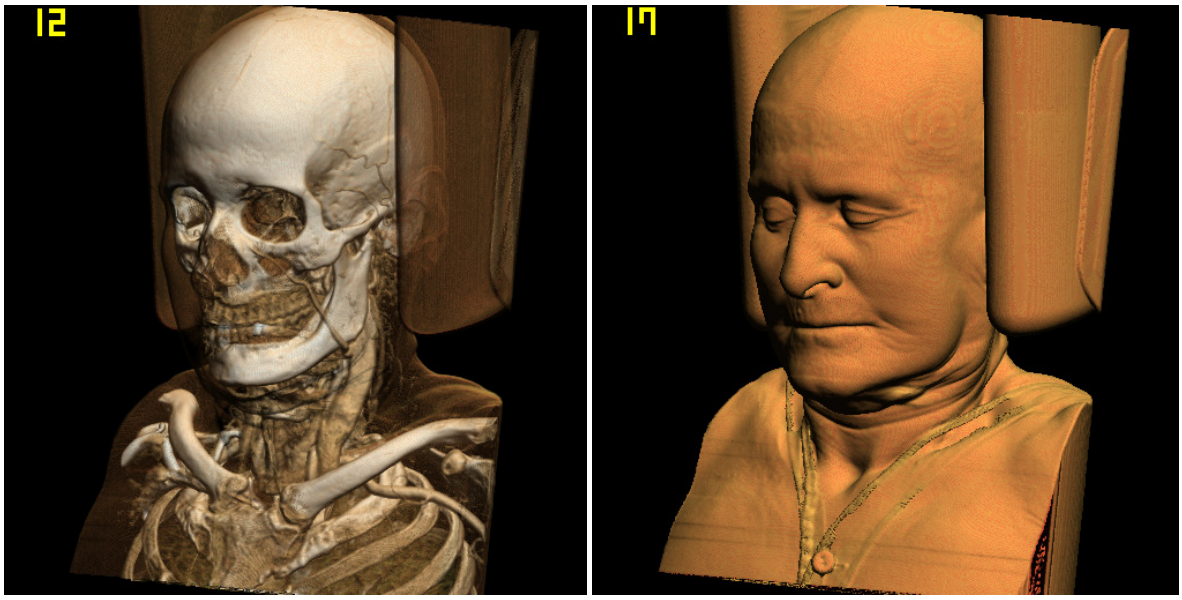
V současné fázi je možné přepínat kvalitativní režimy, zvolit shader (MIP, x-ray, stínování) a volně pohybovat a otáčet s vykreslovaným objektem. Implementováno je dále: možnost provádět vrhání paprsků přes libovolný polygonální objekt; korektní určení hodnoty z-bufferu a interakce s polygonálními objekty OpenGL; segmentace přes jednobitovou 3D texturu; bikubické převzorkování obrazu.

6.2 Výkonnost

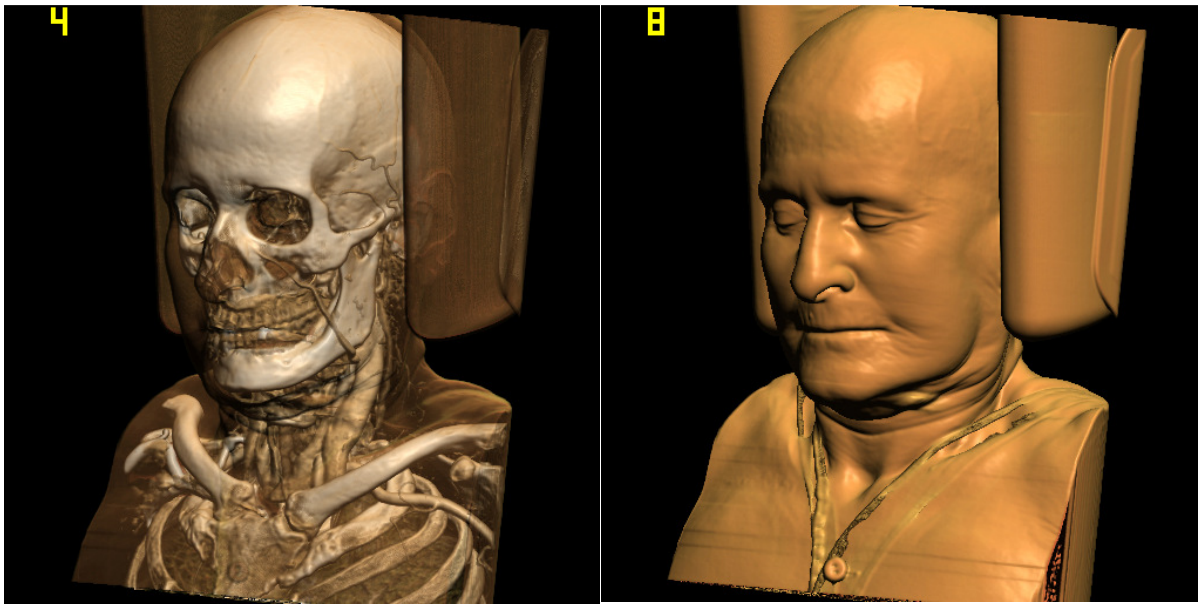
Na současných grafických kartách je počet obrázků za sekundu v rozmezí několika jednotek až desítek v závislosti na zvolené kvalitě. Následující ukázky jsou vykresleny do okna o rozměrech 512×512 obrazových bodů. Test byl proveden na kartě GeForce 8600GTS. Počet obrázků za sekundu je v levém horním rohu.



Obrázek 6-1.1Rychlost vykreslování rentgenového zobrazení a MIP



Obrázek 6-2.2 Rychlost zobrazování v režimu vysoké rychlosti



Obrázek 6-3.3 Rychlost zobrazování v režimu vysoké kvality

6.3 Vizuální kvalita

Následující podkapitola rozebírá různé příčiny artefaktů ray castingu a popisuje stav v rámci řešení projektu.

6.3.1 Možné zdroje nežádoucích artefaktů

Vzorkování

Nízká vzorkovací frekvence může výrazně zhoršit kvalitu výstupu. Optimální vzorkování je alespoň dvakrát pro každý voxel. Toto je dodrženo v režimu maximální kvality. V náhledovém režimu je tato

hodnota prodloužena o 50%. Chyby vzniklé vzorkováním také pomáhá vyhlazovat náhodné posunutí počátku paprsku pomocí šumové look-up textury.

Filtrování

Grafický procesor obsahuje hardwarovou podporu pro trilineární filtrování 3D textur. Toto však není zcela ideální řešení, protože se jedná o velmi jednoduchou metodu. Implementace softwarové trikubické interpolace by přicházela v úvahu pouze na nejnovějších GPU a pravděpodobně pouze v režimu segmentace, kde by tento náročný výpočet byl omezen pouze na důležité oblasti dat.

Klasifikace

V případě, že 1D look-up tabulka obsahuje větší množství vysokých frekvencí, dochází ke vzniku artefaktů. Tyto nežádoucí barevné skoky jsou však viditelné i u relativně plochých LUT. Řešení problému klasifikace je zavedení předintegrované vyhledávací tabulky, která pracuje s celým úsekem mezi vzorky.

Stínování

Problém spojený s předpočítanými normálovými vektory. V tomto projektu jsou normály počítány s vysokou přesností v reálném čase a následně normalizovány.

Integrace

Chyby integrace jsou spojeny s metodou blendingu, kde je barva akumulována v 8 bitech na kanál. V ray casting shaderu je barevná hodnota akumulována s 32 bitovou přesností float.

7 Závěr

Dosažené výsledky lze považovat za dobré. Většina úkolů byla splněna. Další vývoj diplomové práce bude zahrnovat kvalitní grafické uživatelské rozhraní a propojení se segmentačním toolkitem. Segmentační pixel shader je třeba dále otestovat a v případě potřeby implementovat lepší algoritmus interpolace dat.

Literatura

[1] Klír, J., Seidl, L. *Syntéza logických obvodů*. Praha, SNTL 1996.

[2] Pěchouček, M. Toleranční analýza logických obvodů. Sdělovací technika, 8, 1973, s. 293-298.