České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Jakub Groll**

Studijní program: Otevřená informatika
Obor: Softwarové systémy

Název tématu: **Přímé vykreslování volumetrických dat na GPU**

Pokyny pro vypracování:

Seznamte se s přímým vykreslováním volumetrických dat (DVR - Direct Volume Rendering) a s knihovnou dodanou vedoucím práce. Navrhněte a implementuje do knihovny možnost načítat volumetrická data a vykreslovat je na grafické kartě. Umožněte ovlivnit vykreslování pomocí 1D a 2D transfer funkce. Implementujte obě varianty výpočtu vykreslovací rovnice (zepředu dozadu a odzadu dopředu) a porovnejte jejich výkon za použití alespoň čtyř volumetrických modelů různé složitosti a alespoň čtyř různých transfer funkcí.

Seznam odborné literatury:

K. Engel et al. Real-time Volume Graphics, AK Peters, Ltd. 2006

Vedoucí: Ing. Ladislav Čmolík, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

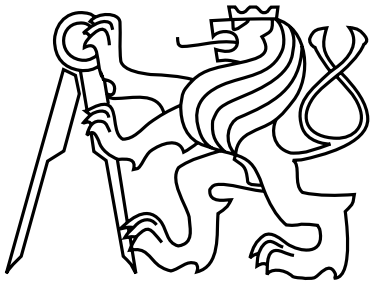
L.S.

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 25. 3. 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce



Bakalářská práce

Přímé vykreslování volumetrických dat na GPU

Jakub Groll

Vedoucí práce: Ing. Ladislav Čmolík, Ph.D.

Studijní program: Otevřená informatika, Bakalářský

Obor: Softwarové inženýrství

20. května 2015

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Ladislavu Čmolíkovi, Ph.D. za cenné rady a připomínky.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 20. 5. 2015 .....

Abstract

This bachelor thesis deals with an issue of volume rendering, specifically Direct Volume Rendering. The goal of this thesis is to introduce the mechanism of volume rendering on GPU to the reader and show him, how can we affect the data with 1D and 2D transfer functions.

My implementation extends the functionality of a Tiger library provided by the supervisor. The library uses OpenGL API, which is accessed through JOGL. The implemented application was tested on four different volumetric datasets, with different transfer functions and with different composition schemes. The results of the testing gives insight into the behavior of direct volume rendering under various circumstances.

Abstrakt

Tato bakalářská práce se zabývá problematikou vykreslování volumetrických dat, konkrétně jejich přímým vykreslováním (Direct Volume Rendering). Cílem práce je seznámit čtenáře s mechanismem vykreslování volumetrických dat na grafickém procesoru a ukázat, jak lze data ovlivnit pomocí 1D a 2D transfer funkce.

Na základě analýzy a návrhu řešení je provedena implementace, která rozšiřuje funkčnost knihovny Tiger, dodanou vedoucím práce. Tato knihovna využívá rozhraní OpenGL API, ke kterému přistupuje skrze JOGL. Výsledná aplikace pro přímé vykreslování byla otestována na čtyřech různých volumetrických modelech, pro různé transfer funkce a pro různá kompoziční schémata. Tím je umožněn vhled do chování přímého vykreslování volumetrických dat za různých podmínek.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Vykreslování volumetrických dat je široký pojem, který popisuje techniky pro generování 2D obrazu ze 3D skalárních dat. Na rozdíl od hraniční reprezentace, využívané zejména v počítačových modelech pro hry a filmy, při vykreslování volumetrických dat pracujeme s celým objemem 3D objektu. Proto se tato reprezentace často označuje jako objemová.

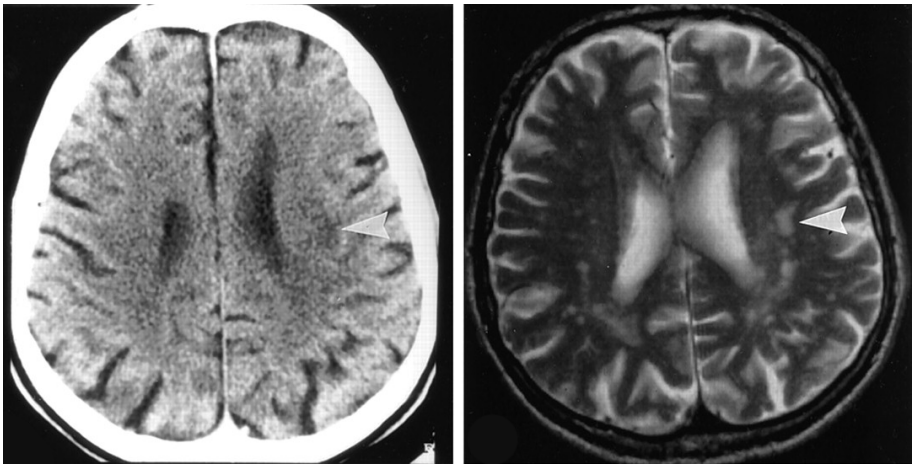
Úvodem práce je nezbytné vysvětlit a popsat co vlastně volumetrická data jsou a kde se s nimi můžeme setkat. Tradičně jsou objemová data získána z měření nebo jsou generována a vizualizována v simulacích pro vědecké účely. Typickým příkladem je využití v medicíně, kdy data umožňují zobrazit vnitřní orgány a stavbu lidského těla pomocí výpočetní tomografie, známé pod zkratkou CT, nebo pomocí magnetické rezonance (MRI). Tato metoda slouží k diagnostice širokého spektra poranění a chorob. Porovnání výsledků obou metod naleznete v Obrázku 1.1. Dalšími příklady jsou dynamické zobrazování kapalin (CFD), geologická data či seismická aktivita. Zdroj [10].

Dle Levoy [17] se s vývojem efektivních vykreslovacích technik objemová data stávají více a více důležitá například v odvětví počítačových her, kdy se dají použít na objekty s nejasnou hranicí, které bychom jen těžce reprezentovali jinými technikami. Jsou to například plyny, kapaliny, přírodní jevy jako oheň, mraky a v neposlední řadě mlha či kouř.

Značnou výhodou, ale zároveň i nevýhodou je fakt, že data nesou informace o svém objemu, neboli v každém bodě objektu máme informaci o materiálu, který se v tomto bodě nachází [5]. S touto skutečností přichází fakt, že tyto informace jsou velmi datově náročné. Dalším negativem je komplexnost dat, která téměř vylučuje ruční modelování [5].

1.1 Cíle práce

Cílem této bakalářské práce je analyzovat techniky přímého vykreslování volumetrických dat, navrhnout a implementovat algoritmus, který bude data vykreslovat na grafickém procesoru a ukázat, jak lze data ovlivnit pomocí 1D a 2D transfer funkce. K tomu je zapotřebí objasnit a analyzovat proces od samotného získání volumetrických dat až po jejich finální vykreslení. V implementační části je poté vysvětleno, jak je realizováno načítání dat a jakým



Obrázek 1.1: Výsledek výpočetní tomografie (vlevo) a magn. rezonance (vpravo). Zdroj [2].

způsobem jsou tyto data vykreslována. Poznatky jsou poté otestovány vhodnými metodami, které ověří závěry vydedukované na konci práce.

1.2 Motivace

Knihovna Tiger, která je používána pro ilustrativní vizualizaci 3D sítí (3D meshů) dosud nemohla zobrazovat volumetrická data a vyzkoušet tak některé z technik i na volumetrických datech. Tato práce umožní zobrazovat volumetrická data a vyzkoušet tyto techniky. K výběru tématu přímého vykreslování dat přispěl fakt, že počítačová grafika je můj velký koníček. Vzhledem k tomu, že se v magisterském studiu budu počítačové grafice s největší pravděpodobností věnovat (byl jsem přijat do navazujícího studia programu Otevřené informatiky na obor Počítačové grafiky a interakce), snažil jsem se téma co nejvíce přiblížit tomuto odvětví.

1.3 Struktura bakalářské práce

Práce je rozdělena do pěti kapitol. V kapitole 1, která je označena Úvod, se nachází uvedení do problematiky vykreslování volumetrických dat obecně. V kapitole 2, nazvané Analýza a návrh řešení, se poté konkrétně zaměříme na metodu přímého vykreslování volumetrických dat a budeme analyzovat způsoby, jakými lze data vykreslit. V následující části 3 si ukážeme příklady použití některých implementovaných metod a programů, a také si popíšeme konkrétně realizovaný kód. V předposlední kapitole 4, označenou Testování, si poté dokážeme, že tvrzení vyřčená v předcházejících kapitolách jsou funkční a platná pro různé volumetrické modely či transfer funkce. Závěrečné zhodnocení práce je možno nalézt v poslední kapitole – Závěr (5).

Kapitola 2

Analýza a návrh řešení

V následující kapitole analyzujeme součásti potřebné pro metodu přímého vykreslování volumetrických dat. Sekce 2.6 rozebírá technologie použité v implementační části bakalářské práce. Následující oddíl 2.7 věnuji hardwarové stránce, kdy si vysvětlíme události odehrávající se na grafické kartě. Hlavní roli zde hraje zobrazovací řetězec (viz 2.7.1). Předtím, než si rozebereme princip zpracování dat po hardwarové straně, zaměříme se na samotnou metodu přímého vykreslování volumetrických dat (Sekce 2.1), na kterou navazuje velmi podstatná pasáž zabývající se vykreslovací rovnicí. V neposlední řadě se seznámíme s pojmy kompozičních schémat a s použitím transfer funkcí.

2.1 Přímé vykreslování volumetrických dat

Předtím, než si objasníme princip zpracování dat na hardwarové straně, musíme se zaměřit na hlavní cíl této práce - přímé vykreslování volumetrických dat. Velká část této kapitoly je věnována pojmu přímého vykreslování, který si vysvětlíme, přičemž analyzujeme jeho součásti a metody, ze kterých se skládá.

Dle [18] přímé vykreslování dat označuje techniku, při které vykreslujeme výsledný obraz *přímo* z volumetrických dat. Nevyužíváme zde mezivýsledků či přechodných konstruktů jako například vytváření kontur povrchů (angl. Contour surface polygons).

Jak již bylo zmíněno, hlavní myšlenkou této metody je získat 2D obraz z dat přímo. Přesněji se snažíme vizualizovat extrahované informace z trojrozměrného pole skalárních hodnot, což může být zapsáno jako zobrazení

$$\phi = \mathbb{R}^3 \mapsto \mathbb{R}, \tag{2.1}$$

neboli funkce ze 3D prostoru do skalární hodnoty. (Zdroj [10].) U těchto dat předpokládáme, že reprezentují semi-transparentní objekt vyzařující (odrážející) světlo. Přímé vykreslování volumetrických dat je výpočetně velmi náročné a může být provedeno několika metodami (viz Sekce 2.3).

2.2 Vykreslovací rovnice

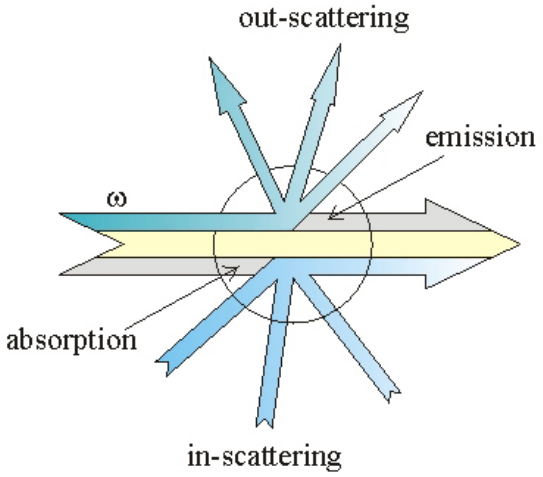
Následující dvě sekce (2.2 a 2.3) čerpají kvůli své komplexnosti z knihy Real-time volume graphics [10], která je díky svým autorům Hadwiger, Kniss, Rezk-Salama, Weiskopf„ Engel, jedním z nejkompetentnějších zdrojů dané problematiky. Účel této kapitoly je popsat a vysvětlit některé z fyzikálních a matematických vlastností rovnic vyskytujících se ve výpočtu přenosu světla při vykreslování volumetrických dat.

Fyzikální základ pro vykreslování volumetrických dat se opírá o geometricku optiku, kde se předpokládá, že se světlo šíří prostorem přímočaře, vyjma situací, kdy se světlo střetne s jinými objekty. Tudíž vzájemné působení mezi světlem a hmotou je situace, které se budeme nyní věnovat. Následující typy interakcí paprsku světla s hmotou jsou brány v úvahu:

Vyzařování (angl. Emission)
Některé materiály (například plyny) vyzařují světlo, převodem tepla, či jiných vlastností, na energii.

Absorpce (angl. Absorpion)
Různé meteriály mohou naopak pohlcovat (absorbovat) světlo, které je následně převedeno v teplo. Energie generující světlo je v tomto případě redukována.

Rozptyl (angl. Scattering)
Světlo se přirozeně odráží a rozptyluje od ostatních objektů. V této situaci se mění směr šíření světla, jak můžeme vidět na Obrázku 2.1.



Obrázek 2.1: Typy vzájemného působení světla s objektem. Zdroj [27].

Zpracováním výše uvedených situací získáme kompletní rovnici pro výpočet šíření světla

$$\omega \cdot \nabla_{\mathbf{x}} I(\mathbf{x},\omega) = -(\kappa(\mathbf{x},\omega) + \sigma(\mathbf{x},\omega))I(\mathbf{x},\omega) + q(\mathbf{x},\omega) + \int \sigma(\mathbf{x},\omega')p(\mathbf{x},\omega',\omega)I(\mathbf{x},\omega')\,\mathrm{d}\Omega', \tag{2.2}$$

kde proměnné mají následující význam: $\mathbf{x}$ - pozice a ω' - směr původního paprsku, ω - směr paprsku odraženého, dále κ - koeficient absorpce, I - hodnota dopadajícího světla, σ - koeficient rozptylu, p - fázová funkce, q - koeficient vyzařování.

Jelikož výsledek této komplikované rovnice je příliš výpočetně složitý, často se využívá zjednodušených modelů. U těchto modelů vynecháme některé prvky z množiny interakcí vyjmenovaných výše. Tímto přístupem získáme rovnice, které jsou lépe uchopitelné. Výčet modelů vypadá následovně: Absorption Only Model, Emission Only Model, Emission-Absorption Model, Single Scattering and Shadowing Model, Multiple Scattering Model.

Z názvů modelů je patrné, které složky interakcí obsahují. Nyní se blíže podíváme na jeden z těchto modelů - Emission-Absoprtion Model.

Emission-Absoprtion Model je nejvíce používným modelem pro vykreslování volumetrických dat, jelikož se jedná o kompromis mezi efektivitou výpočtu a faktu, že je rovnice dostatečně obecná. Tento model, počítající s vyzařováním a absorcí světla, avšak ignorující rozptylování a nepřímé osvětlení, vede na následující rovnici

$$\omega \cdot \nabla_{\mathbf{x}} I(\mathbf{x},\omega) = -\kappa(\mathbf{x},\omega)I(\mathbf{x},\omega) + q(\mathbf{x},\omega), \tag{2.3}$$

která je označována jako *vykreslovací rovnice* (angl. volume-rendering equation). Přesněji se jedná o vykreslovací rovnici v diferenciální formě, jelikož popisuje šíření světla změnou jeho vyzařování. Pokud uvažujeme šíření jediného paprsku světla, může být rovnice 2.3 přepsána jako

$$\frac{\mathrm{d}I(s)}{\mathrm{d}s} = -\kappa(s)I(s) + q(s), \tag{2.4}$$

kde jednotlivé pozice jsou popsány délkou s měřenou od počátku vyslaného paprsku.

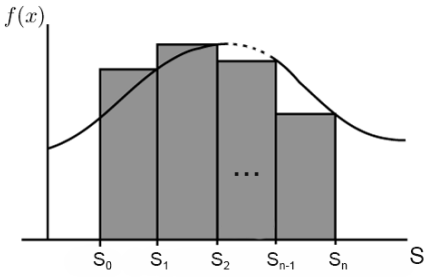
2.2.1 Integrál pro vykreslení volumetrických dat

Vykreslovací rovnice (Rovnice 2.4) může být řešena pro vyzařování integrováním podle směru světelného paprsku od začátku $s = s_0$ až po konec paprsku $s = D$, což vede na *integrál pro vykreslení volumetrických dat* (angl. volume-rendering integral)

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)\,\mathrm{d}t} + \int_{s_0}^{D} q(s) e^{-\int_{s_0}^{D} \kappa(t)\,\mathrm{d}t}\,\mathrm{d}s. \tag{2.5}$$

2.2.2 Diskretizace

Hlavním cílem vykreslování volumetrických dat je výpočet integrálu 2.5. Jelikož tento výpočet ve věšině případů nelze vyřešit analyticky, na řadu přicházejí numerické metody, které se snaží najít aproximace původního integrálu, známé pod pojmem diskretizace[1].



Obrázek 2.2: Aproximace integrálu - Riemannův přístup. Zdroj [6].

Běžný způsob je rozdělení integrálu na malé intervaly, které jsou popsány souřadnicemi $s_0 < s_1 < ... < S_{n-1} < s_n$, se začátkem v s_0 a koncem v $s_n = D$ (viz 2.2).

S tímto faktem můžeme získat hodnotu vyzářeného světla na pozici s_i

$$I(s_i) = I(s_{i-1})T(s_{i-1}, s_i) + \int_{s_{i-1}}^{s_i} q(s)T(s, s_i)\,\mathrm{d}s. \tag{2.6}$$

[1] diskretizace, fyz. náhrada spojitého prostředí (kontinua) systémem diskrétních bodů

Zde zavádíme dvě značení T - průhlednost, c_i - barva (neboli vyzářené světlo) i-tého intervalu.

$$T_i = T(s_{i-1}, s_i), \quad c_i = \int_{s_{i-1}}^{s_i} q(s)T(s, s_i)\,\mathrm{d}s. \tag{2.7}$$

Hodnota vyzářeného světla v místě kde paprsek opouští objekt je

$$I(D) = I(s_n) = I(s_{n-1})T_n + c_n = (I(s_{n-2})T_{n-1} + c_{n-1})T_n + c_n = ...,$$

což lze přepsat jako

$$I(D) = \sum_{i=0}^{n} c_i \prod_{j-i+1}^{n} T_j, \quad \text{kde } c_0 = I(s_0). \tag{2.8}$$

2.3 Kompoziční schémata

Kompoziční schémata (angl. Compositing schemes) jsou základem pro iterativní přístup výpočtu diskretizovaného integrálu pro vykreslení volumetrických dat (Rovnice 2.8).

Rozlišujeme dva základní přístupy - vykreslování zepředu dozadu (dále jen angl. front-to-back) a vykreslování zezadu dopředu (dále angl. back-to-front). Myšlenkou těchto přístupů je sekvenční zpracování výše zmíněné rovnice.

2.3.1 Front-to-back

Front-to-back schéma použito, putují-li zobrazovací paprsky od kamery k objektu. Iterativní formulace rovnice vyjadřující tuto metodu začíná na pozici nejblíže kameře a končí na zadní straně našeho objektu. Někdy je tato metoda nazývána *Under operator* - převzato z [8]. V následujících rovnicích bude proměnná C reprezentovat barvu (typicky RGB hodnotu).

$$\hat{C}_i = \hat{C}_{i+1} + \hat{T}_{i+1}\hat{C}_i,$$
$$\hat{T}_i = \hat{T}_{i+1}(1 - \alpha_i),$$

s počátečními hodnotami[2]

$$\hat{C_n} = C_n,$$
$$\hat{T_n} = 1 - \alpha_n.$$

[2] v tomto případě je první pozice i = n nejblíže kameře

Přepsáním proměnných $C_{dst} = \hat{C}_j$ $(j = i,i+1)$, $C_{src} = C_i$, $\alpha_{dst} = 1 - \hat{T}_j(j = i, i+1)$, $\alpha_{src} = \alpha_i$ získáme častější zápis rovnice:

$$\begin{aligned} C_{dst} &\leftarrow C_{dst} + (1 - \alpha_{dst})C_{src}, \\ \alpha_{dst} &\leftarrow \alpha_{dst} + (1 - \alpha_{dst})\alpha_{src}. \end{aligned} \tag{2.9}$$

Zde C reprezentuje barvu a α neprůhlednost. Proměnné s dolním indexem $_{src}$ popisují vstupní hodnoty. Naopak proměnné s dolním indexem $_{dst}$ popisují výstupní hodnoty, které akumulují hodnoty barvy a neprůhlednosti.

2.3.2 Back-to-front

Otočením směru procházení scény dle paprsku získáme back-to-front schéma, dle [8] označované jako *Over operator*:

$$\begin{aligned} \hat{C}_i &= \hat{C}_{i-1}(1 - \alpha_i) + C_i, \\ \hat{T}_i &= \hat{T}_{i-1}(1 - \alpha_i), \end{aligned}$$

s počátečními hodnotami[3]

$$\begin{aligned} \hat{C}_0 &= C_0, \\ \hat{T}_0 &= 1 - \alpha_0. \end{aligned}$$

Poznamenejme, že u této metody nepotřebujeme znát akumulovanou hodnotu průhlednosti, tím pádem můžeme druhou rovnici vynechat. Iterativní rovnice pak vypadá následovně

$$C_{dst} \leftarrow (1 - \alpha_{src})C_{dst} + C_{src}. \tag{2.10}$$

2.3.3 Maximum intensity projection

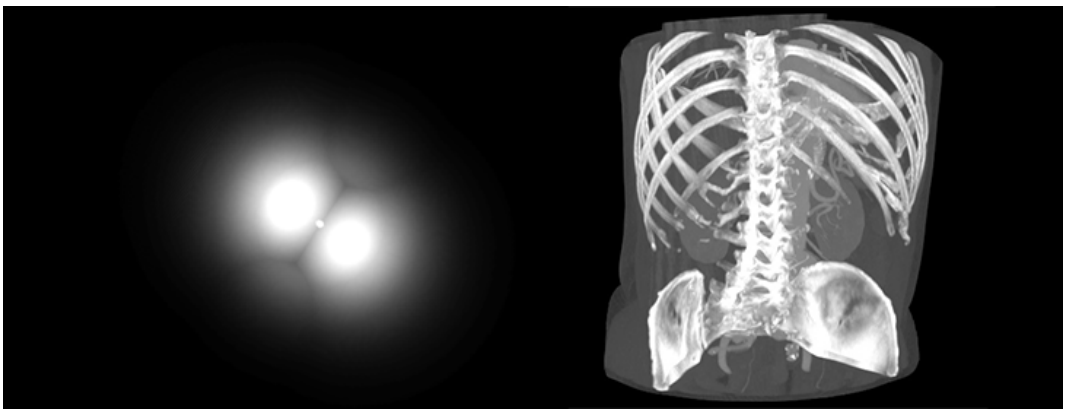
Maximum intensity projection je kompoziční schéma, které je hojně využíváno v medicínském prostředí. Je použitelné zejména na tomografická data po aplikování kontrastních látek (založených například na bázi jódu). Metoda je postavena na výpočtu barvy pixelu,

[3] iterace začíná v bodě $i = 0$ a končí v $i = n$

jehož hodnota je určena jako maximum ze všech intenzit měřených podél paprsku:

$$I = \max_{k=0..N} (s_k), \tag{2.11}$$

kde s_k je skalární hodnota meřená podél paprsku, N je počet vzorků a I výsledná intenzita. Velmi názorný je Obrázek 2.3, kde maximální intenzitu v těle představují kosti žeber, páteře a pánve. Všimněte si, že ztrácíme informaci o objemu s nižší intenzitou za těmito vykreslenými částmi.



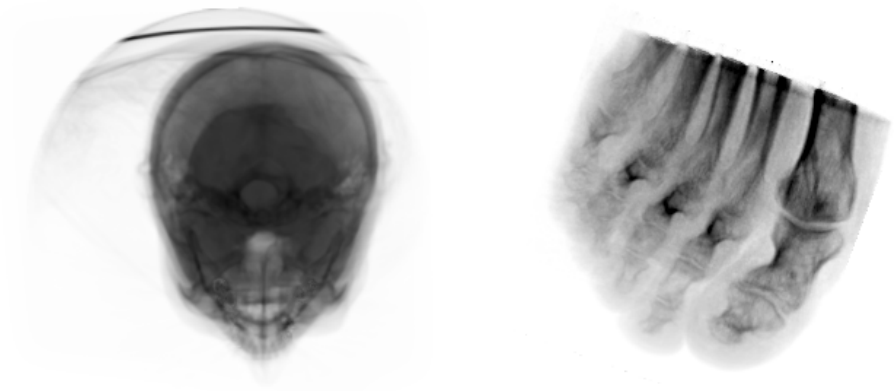
Obrázek 2.3: Maximum Intensity Projection, atom vodíku (vlevo) a torzo těla (vpravo). Zdroj (pouze torzo těla) [3].

2.3.4 Average intensity projection

Average intensity projection (viz Obrázek 2.4) je kompoziční schéma podobné MIP. Rozdíl nastane ve výpočtu barvy pixelu, kde výsledná barva bude vykreslena jako hodnota průměru intenzit podél paprsku:

$$I = \frac{1}{N} \sum_{k=0}^{N} S_k \tag{2.12}$$

Zde je s_k skalární hodnota meřená podél paprsku, N počet vzorků a I výsledná intenzita.



Obrázek 2.4: Average Intensity Projection s použitím převrácených hodnot barev pro lepší přehlednost, lidská lebka (vlevo) a část chodidla (vpravo)

2.4 Transfer funkce

Dle autora Kniss et al. [15] se přímé vykreslování volumetrických dat prokázalo jako efektivní a flexibilní metoda pro vykreslování 3D skalárních souborů dat. Transfer funkce jsou fundamentální složkou přímého vykreslování dat, jelikož jejich role je v podstatě vykreslování dat na základě optických vlastností (jako jsou barva a průhlednost). Tyto hodnoty poté transfer funkce přiřazuje volumetrickým datům, jenž pak mohou být běžným způsobem vykresleny na grafické kartě.

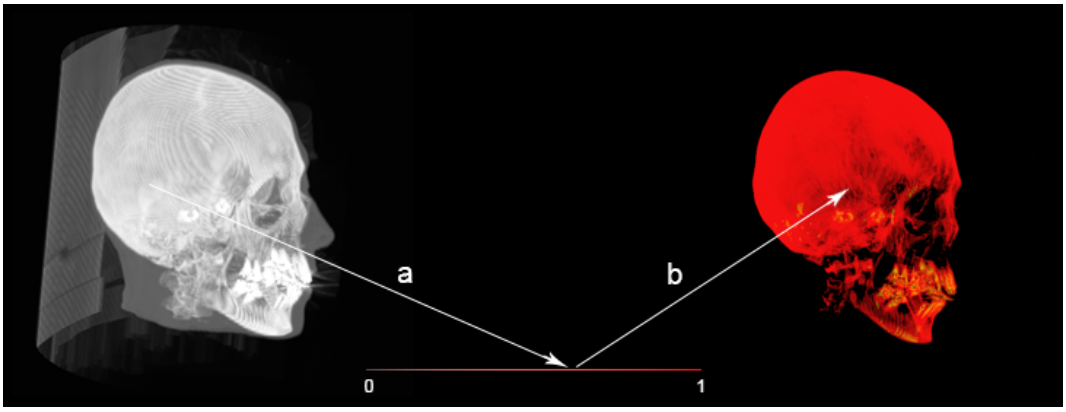
Transfer funkce byly dlouhou dobu omezeny pouze na funkce jednodimenzionální, které dané vlastnosti přiřazují volumetrickým datům jen na základě hodnot těchto dat. V mnohých případech se s tímto typem transfer funkcí nedá správně zachytit a vizualizovat míněnou oblast. Poté je třeba použít vícedimenzionálních transfer funkcí. Zdroj [15]. V této práci se zaměříme pouze na 1D a 2D transfer funkce, které jsem měl za úkol analyzovat a následně použít.

2.4.1 1D transfer funkce

V nejjednodušším případě se transfer funkce mapují na základě hodnoty skalárních dat. Obrázek 2.5 ukazuje jak funguje mapování barvy na náš objekt. Každá skalární hodnota volumetrických dat je převedena do intervalu $\langle 0, 1 \rangle$, kde 0 odpovídá nejmenší hodnotě skalárních dat objektu a 1 naopak hodnotě největší. Každé hodnotě tedy přísluší určitý vektor kanálů RGBA, který je poté zapsán do fragmentu jako výsledná barva.

2.4.2 2D transfer funkce

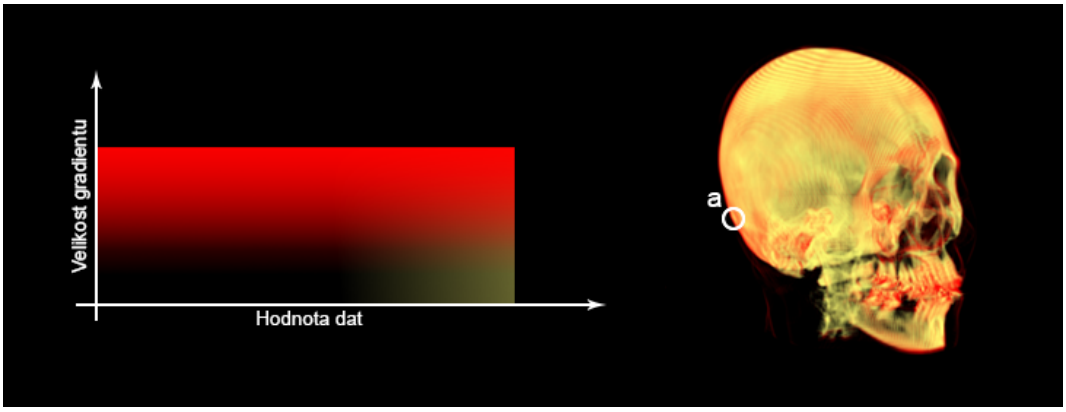
Přidáním druhého rozměru transfer funkci dostáváme možnost mapovat hodnoty na základě více parametrů. Vhodnou proměnnou se ukázala být hodnota velikosti *gradientu* [15]. Jelikož nám tato veličina udává jak rychle se hodnoty mění v přilehlém okolí, je ideální pro



Obrázek 2.5: Použití 1D transfer funkce, převedení hodnot do intervalu $\langle 0, 1 \rangle$ (a), zapsání barvy do fragmentu (b)

zvýraznění přechodu mezi objekty s rozdílnou hustotou. Toho se využívá zejména v lékařství pro rozlišení různých druhů tkání. Zdroj [14].

Na Obrázku 2.6 vidíme transfer funkci na jejíž svislé poloose určuje hodnotu barvy právě velikost gradientu. Na výsledném objektu se to poté projeví červeným zbarvením (zvýrazněno v obrázku u bodu a).



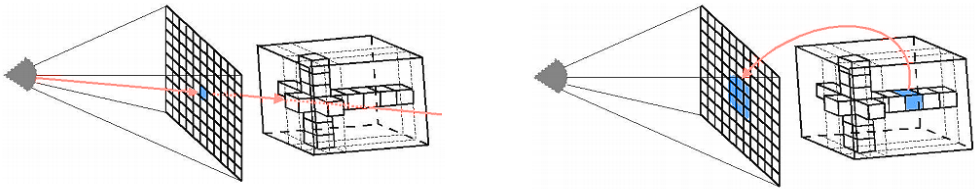
Obrázek 2.6: Použití 2D transfer funkce, 2D transfer funkce (vlevo), výsledek použití 2D transfer funkce s veličinou velikost gradientu (vpravo)

2.5 Vykreslovací techniky

Pro výše zmíněná schémata existují různé techniky vykreslování dat. Tyto techniky můžeme dělit dle způsobu zpracování dat na takzvané techniky Image-order a Object-order (viz Obrázek 2.7) [1]. Vykreslovací techniky jsou typicky používány k výpočtům diskrétní apro-

ximace integrálu pro vykreslení volumetrických dat.

Dle [10], přístup Image-order pracuje ve dvourozměrném prostoru zobrazovací plochy, kdy data zpracováváme po jednotlivých pixelech. Naproti tomu Object-order přístup zpracovává data ve 3D prostoru. Zpracovaná objemová data jsou poté promítána do zobrazovací plochy.



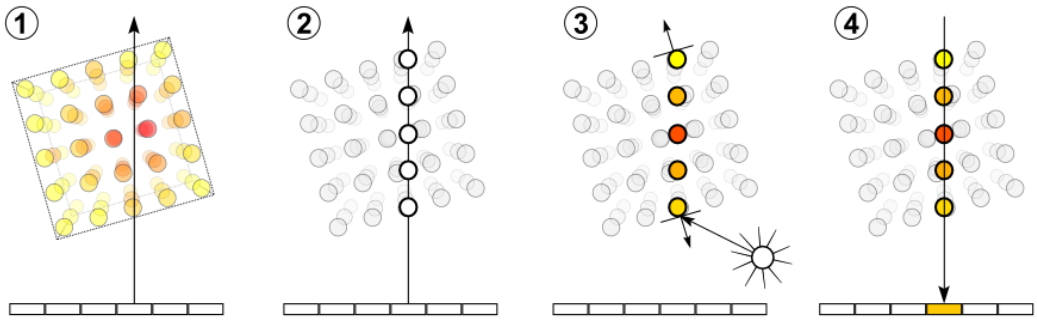
Obrázek 2.7: Vykreslení dat pomocí Image-order (vlevo) a Object-order techniky (vpravo). Zdroj [1].

Základní klasifikace vykreslovacích technik vypadá následovně:

2.5.1 Ray casting

Dle [10], ray casting je nejvíce používaná Image-order technika pro vykreslování volumetrických dat. Základní myšlenkou této metody je přímo vypočítávat hodnoty podél všech paprsků. Pro každý pixel ve scéně je určen právě jeden paprsek. Poté vzorkujeme v určitých intervalech objemová data našeho objektu (viz Obrázek 2.8).

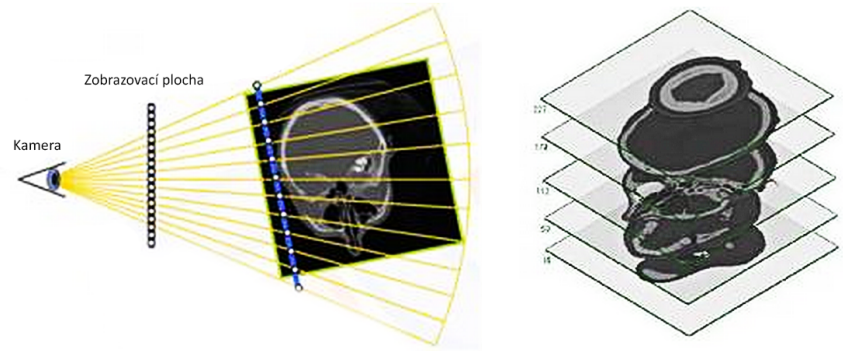
Pro tuto techniku bylo vyvynuto několik způsobů, jak zrychlit vykreslovací proces. Jedná se například o Early Ray Termination či Space Leaping. Více se lze dočíst v publikaci [24].



Obrázek 2.8: Ray casting - vyslán paprsek pro každý pixel ve scéně (1), Vzorkování (2), Interpolování hodnot a aplikace transfer funkce (3), Iterativní výpočet diskretizovaného integrálu pro vykreslení volumetrických dat (4). Zdroj [4].

2.5.2 Texture slicing

Dle [1] se jedná o velice rozšířenou techniku uplatňovanou převážně pro vykreslování na grafických procesorech. Tento přístup generuje výsledný obraz s použitím setu obrázků - textur. Tyto 2D řezy nacházející se ve 3D scéně vzorkují náš objekt a následně jsou na základě použitého kompozičního schématu promítnuty do zobrazovací plochy (viz Obrázek 2.9). Metoda vykreslení Texture slicing spadá pod Object-order přístup.



Obrázek 2.9: Texture slicing (vlevo), Set 2D řezů (vpravo). Zdroj [7].

Výhodou je široká podpora ze strany hardwaru díky nenáročným požadavkům (vyžaduje pouze podporu texture a blending). Díky tomuto faktu byla donedávna technika Texture slicing dominantní metodou ve vykreslovaní volumetrických dat na GPU. Častý problém, se kterým se můžeme setkat, je vznik artefaktů pod úhlem 45 stupňů.

2.5.3 Shear-warp volume rendering

Podobně jako v předchozí technice jsou v této Object-order metodě objemová data procházena po jednotlivých řezech. Myšlenka této techniky spočívá v poloze paprsků, které musí být vzájemně rovnoběžné a zároveň kolmé na 2D řezy našeho objektu. Toho docílíme zkosením (střihem - shear) daného objektu.

Silnou stránkou Shear-warp volume rendering techniky je široká škála optimalizačních metod, díky čemuž je jednou z nejrychlejších metod vykreslování dat na CPU. Více o této technice lze vyčíst v v publikaci [16].

2.5.4 Splatting

V metodě Splatting promítáme 3D objekty nazvané jádra (angl. kernel) na zobrazovací plochu. Těmto 2D projekcím se říká stopy (angl. footprint).

Object-order metoda Splatting byla vyvinuta za účelem zrychlení výpočtů vykreslení dat na úkor jiných atributů, například méně přesného vykreslování. Aproximováním projekce a

následným složením všech stop získáme výsledný obraz. Výhodou této techniky je její malá časová náročnost, což se ovšem projeví například v přiblížených úsecích, kde může být obraz rozmazaný. Zdroj [10].

2.5.5 Cell projection

Object-order technika využívaná pro tetrahedrické a více komplexní (nerovnoměrné) mřížky. Známý je například PT algoritmus [Shirley and Tuchman].

2.6 Analýza technologií pro implementaci

V této části se budu věnovat technologiím použitých v implementační části. Ve výběru této práce přispěl fakt, že s Javou mám největší zkušenosti a díky knihovně JOGL mohu přistupovat k funkcím OpenGL relativně snadno.

2.6.1 OpenGL

První z technologií, se kterou jsem se setkal, je průmyslový standard specifikující multiplatformní rozhraní OpenGL. Jeho základní funkcí je vykreslování do frame bufferu. Činnost OpenGL se řídí vydáváním příkazů pomocí volání funkcí. Umožňuje nám vykreslit základní primitiva, nahrávat textury na grafickou kartu a má velkou výhodu, že implementace OpenGL existují pro prakticky všechny počítačové platformy. Jako velké plus bych zmínil rozsáhlou dokumentaci, která se nachází na oficiálních stránkách. Zdroj odstavce [19].

Na OpenGL lze nahlížet jako na automat - v každém okamžiku má definován svůj vnitřní stav, jenž je určen hodnotami vnitřních proměnných. Zdroj - Petr Felkel [21]. Tyto proměnné lze nastavovat a ptát se na jejich hodnotu.

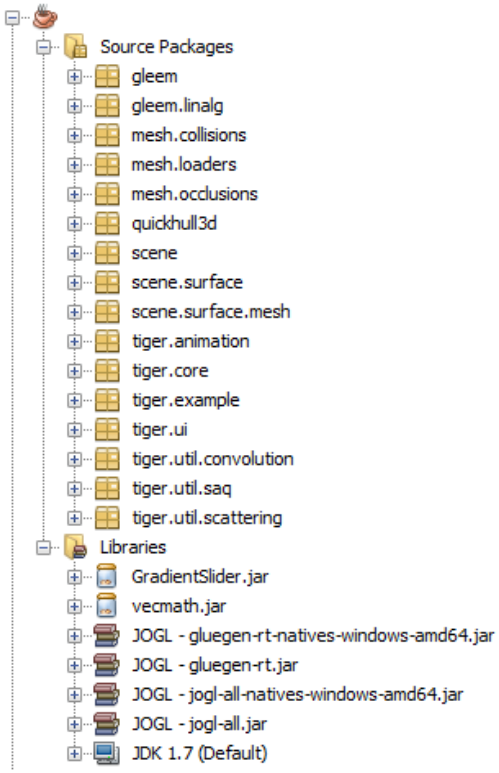
2.6.2 JOGL

JOGL, díky níž přistupuje také Tiger k OpenGL API, je knihovna v Javě, která umožňuje přístup k celému API OpenGL. JOGL pracuje s AWT, Swing a SWT knihovnami. JOGL knihovnu vyvinul Kenneth Bradley a Christopher John Kline, a dále vývoj pokračoval pod záštitou Sun Microsystems Game Technology Group. Zdroj [12].

Velký problém mi dělalo pochopení základů a používání této knihovny. Většina návodů se věnovala přípravě a instalaci a navíc neexistoval jeden ucelený návod. Kód je ovšem velmi dobře čitelný hlavně díky tomu, že využíváme funkcí OpenGL, a tak nebylo těžké pochopit, co jaký úsek kódu znamená.

2.6.3 Tiger (Toolkit for Interactive Graphics rendERing)

Tiger je knihovna v Javě pro zjednodušení vývoje víceprůchodového vykreslování v OpenGL. Tiger využívá JOGL k přístupu k OpenGL API (viz Obrázek 2.10). [11]



Obrázek 2.10: Struktura projektu - balíčky knihovny Tiger, využívající JOGL

Důležitou částí pro moji práci je balíček tiger.core, který obsahuje třídy pro práci s texturami. V sekci 3.1 se blíže podíváme na realizaci třídy pro 3D textury, kterou jsem měl za úkol implementovat a následně použít k vykreslení dat.

2.6.4 GLSL (OpenGL Shading Language)

S GLSL[4] se konkrétně setkáme v sekci 2.7.2 a 2.7.3, kdy si ukážeme jednoduché programy operující na grafické kartě (shadery).

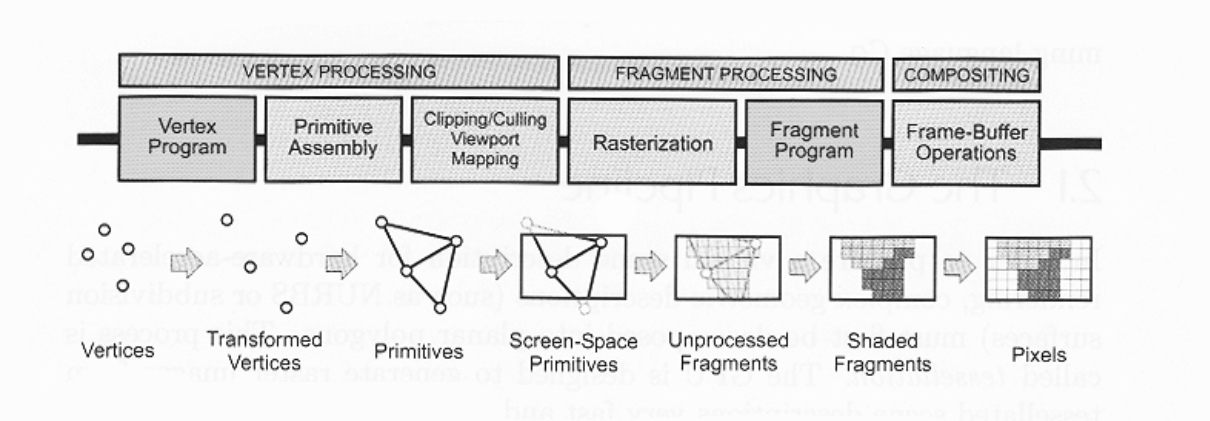
OpenGL Shading Language vychází ze syntaxe C, kdy tento jazyk vznikl v rámci procesu postupné trasformace fixního vykreslovacího řetězce na řetězec programovatelný (programovatelná pipeline). [20] U tohoto jazyka bych zmínil takzvané direktivy, což jsou příkazy pro preprocesor, který zpracovává kód před jeho spuštěním. Dvě direktivy jsem použil v mých shaderech, a to #version a #extension, což jsou unikátní direktivy pro tento jazyk nutné pro správný chod programu. Dále můžeme použít běžných direktiv z jazyka C.

[4]Další alternativou pro GLSL je například jazyk Cg (C for Graphics), který se mírně liší v syntaxi.

2.7 Vykreslení geometrie na GPU

Hlavní novinkou ve funkcionalitě grafických čipů je nahrazení tradičního fixního vykreslovacího procesu programovatelným řetězcem [10]. Díky možnosti nahrání mikroprogramů na grafickou kartu, dostává programátor kontrolu nad výpočty, které probíhají na grafickém procesoru. Tyto takzvané shadery jsou na grafické kartě prováděny velmi efektivně a rychle. Programátor má dále možnost si definovat vlastní osvětlovací systém, texturování, stínování apod. Posloupnost kroků užitých k vytvoření 2D reprezentace z výchozích dat označujeme pojmem *zobrazovací řetězec*. [21]

2.7.1 Zobrazovací řetězec



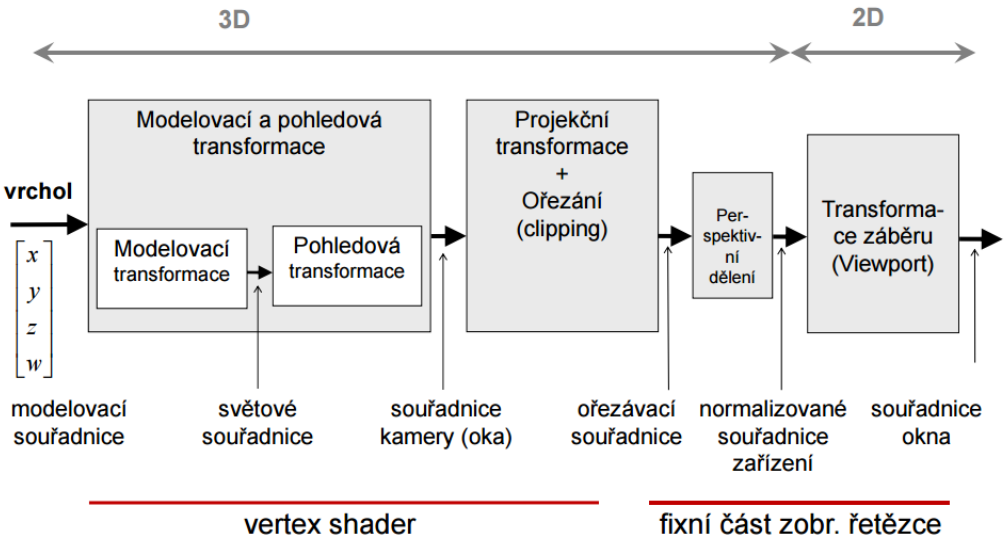
Obrázek 2.11: Programovatelný zobrazovací řetězec (angl. The programmable graphics pipeline). Zdroj [10].

S nástupem grafických karet počítače prodělaly výraznou změnu, která uvedla v pohyb vývoj programovatelného zobrazovacího řetězce. Programovatelný řetězec nahradil mnohé zastaralé funkce fixního řetězce, jako například okamžitého režimu (angl. immediate mode) v OpenGL. [26] Pro názornost lze princip vykreslení dat rozdělit na tři základní úrovně (viz Obrázek 2.11). Nutno dodat, že proces vykreslení dat na novějších kartách je složitější, nicméně v mé práci jsem použil metody následující.

Zpracování vrcholů (angl. Vertex processing)
V této části se spočítají lineární transformace příchozích vrcholů, jako například rotace, posunutí, zvětšení, případně zmenšení.

Jak můžeme vidět na obrázku 2.12, vrcholy jsou transformovány z modelovacích souřadnic do světových souřadnic (modelovací transformace), kdy umístíme a natočíme objekty ve scéně. Dále jsou vrcholy převedeny do souřadnic kamery (pohledová transformace), kdy získáme polohu a natočení kamery ve scéně a následně je aplikována projekční transformace, kde

se určí typ projekce, ořezání. [22] Díky tomu se někdy zpracování vrcholů označuje obecněji jako zpracování geometrie (angl. geometry processing). Vrcholy jsou sestaveny v geometrická primitiva, na které můžeme aplikovat další funkce. Využíváme zde *vertex shaderu*.



Obrázek 2.12: Logické kroky při transformaci vrcholů. Zdroj [22].

Zpracování fragmentů (angl. Fragment processing)
Geometrická primitiva jsou rasterizována v takzvané fragmenty. Každý fragment připadá jednomu pixelu výsledné plochy. Rasterizační jednotka pro každý fragment vypočítá sadu atributů, získanou interpolací příchozích atributů vrcholů. Každý fragment tedy nese sadu atributů (např. barvy, souřadnice textury atd.). Fragment shader vypočítá výslednou barvu (případně hloubku), která je poté zapsána do registrů. Zdroj [10].

Výsledná kompozice (angl. Compositing)
Zde se vyhodnocuje výsledná barva jednotlivých fragmentů pomocí *frame bufferu*, do kterého jsou poté fragmenty zapsány. Na fragmenty jsou aplikovány podmínky, které rozhodnou, zda-li příchozí fragment má být vyřazen či vykreslen na obrazovce. Více o operacích a testech spouštěných na frame bufferu se můžete dozvědět v sekci 2.7.4. Zdroj [10].

2.7.2 Vertex shader

Vertex shader je program, který se provede na každém vrcholu (neboli vertexu) vstupní geometrie scény. Mezi nejčastější operace ve vertex shaderu patří transformace vrcholů, které byli vysvětleny v sekci 2.7.1.

Existují určitá pravidla a specifikace, které je třeba dodržovat pro správný chod vertex shaderu. Každy vstupní vrchol se musí mapovat na určitý výstupní vrchol, a zároveň tyto

vrcholy mezi sebou nesdílejí informace o svém stavu. Mapování mezi vstupními a výstupními vrcholy je tedy 1:1. Jinými slovy do programu vstoupí jeden vrchol, je upraven a zase vystoupí, nelze tedy vrcholy přidávat či odebírat. [5]

Příklad jednoduchého vertex shaderu vypadá následovně:

```
#version 330
in vec4 vert;

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;

void main()
{
    gl_Position = projection * view * model * vert;
}
```

Listing 2.1: Jednoduchý vertex shader v jayzce GLSL

2.7.3 Fragment shader

Fragment shader je využíván k výpočtu výsledné barvy a v některých případech hloubky každého z fragmentů. Fragment shader je vykonán pro jednotlivé fragmenty zvlášť. Každý fragment získaný z rasterizace nese sadu atributů (např. barvy, souřadnice textury atd.). Fragment procesor pro jednotlivé fragmenty, a tedy i sady atributů, spustí fragment shader a zapíše vypočítanou barvu (případně hloubku) do registrů.[10]

Příklad jednoduchého fragment shaderu vypadá následovně:

```
#version 330

void main()
{
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Listing 2.2: Jednoduchý fragment shader v jayzce GLSL

2.7.4 Frame buffer

Frame buffer je kolekce bufferů, které mohou být použity jako výstup pro vykreslování. OpenGL má dva druhy frame bufferů - defaultní, ke kterému přistupujeme pomocí OpenGL Context a uživatelem vytvořené Framebuffer objekty - ty odkazují převážně na textury. [9]

Dle Hadwiger et al. [10] jsou do frame bufferu zapsány také fragmenty získané rasterizací. Ty jsou uloženy jako 2D pole atributů budoucích pixelů (barva, hloubka, neprůhlednost). Při zapsání dalšího z fragmentů upravíme dosavadní hodnoty na frame bufferu dle určitých kritérií a parametrů.

Dále je možné na frame bufferu spouštět testy, které rozhodují o zapsání fragmentu. Krátký přehled vypadá následovně:

Alpha Test
Tento test zamítne fragmenty na základě porovnání hodnot průhlednosti fragmentu s určitou referenční hodnotou. Alpha Test se dá využít v mnoha směrech, avšak původní účel této operace bylo vyřazení zcela průhledných fragmentů.

Šablona (angl. Stencil)
Šablona nám umožňuje označit pixely, na které aplikujeme masku uloženou v takzvaném *stencil bufferu*. Pomocí této masky se zvolí bitová rovina. U této operace používáme víceprůchodové algoritmy pro speciální efekty (obtisky, obrysy, odrazy atd.). [22]

Hloubkový test (angl. Depth Test)
Jelikož geometrická primitiva nemají pevně danou sekvenci, ve které se budou vykreslovat, je třeba hloubkového testu, který zaručí správné vykreslení částečně zakrytých objektů ve scéně. Hloubka fragmentu je tedy uložena v *depth bufferu*. Tato operace testuje příchozí hloubku fragmentu a porovnává ji s hodnotou již uloženou. Zakryté fragmenty mohou být ihned vyřazeny. Tento test se často označuje jako *z-test*, jelikož rozhodnutí o vyřazení fragmentu z frame bufferu udává zetová souřadnice.

2.8 Návrh řešení

V této sekci si popíšeme návrh řešení bez zatížení implementací. Veškeré závěry učiníme na základě analýzy.

Samotný algoritmus vykreslení dat bude využívat metodu Ray casting (viz 2.5.1), která přímo vypočítává hodnoty podél všech paprsků. Pro každý pixel ve scéně je určen právě jeden paprsek a následně vzorkujeme v určitých intervalech objemová data našeho objektu. Implementace bude provedena na GPU - každé metodě vykreslení bude náležet fragment shader.

2.8.1 Postup

Po nahrání dat na grafickou kartu začneme tím, že vykreslíme přivrácené (přední) plošky kvádru dat, díky čemuž získáme texturovací souřadnice začátku paprsku. Analogicky dostaneme texturovací souřadnice konce paprsku vykreslením odvrácených plošek. Ve fragment

shaderech si poté odečtením těchto souřadnic vypočítáme směr paprsku. Budeme tedy paralelně trasovat pro každý fragment paprsek, podél něhož budeme vypočítávat výslednou barvu. Způsoby jak vypočítat barvu výsledného pixelu se liší podle toho čeho chceme dosáhnout a proto je nyní popíšeme níže.

2.8.2 Front-to-back

Algorithm 1 Front-to-back metoda

```
1:  procedure FrontToBack
2:  loop:
3:      data ← texFront
4:      data ← data ∗ fraction
5:      color ← get value from tf
6:
7:      cout ← cout + (1 − alpha) ∗ color ∗ color.alpha
8:      alpha ← alpha + (1 − alpha) ∗ color.alpha
9:
10:     if color.alpha > 0.99 then
11:         goto loop.
12:     else
13:         texFront ← texFront + direction
14:     end if
15: end procedure
```

Pseudokód 1 reprezentující front-to-back schéma prochází v cyklu data podél paprsku. Na řádku 3 nahrajeme do proměnné data z 3D textury. Ta vynásobíme zlomkem, který nám převede rozsah hodnot na interval $\langle 0, 1\rangle$. Na základě převedené hodnoty se určí hodnota barvy vstupující do vykreslovací rovnice. Takzvaný *Under operator* (viz 2.9) aplikujeme na řádcích 7 - 8. Na základě hodnoty alfa kanálu jsem poté přidal ořezávání dat - řádek 10.

2.8.3 Back-to-front

Schéma back-to-front se od předchozího schématu liší pouze vykreslovací rovnicí. Na data, která procházíme zezadu, aplikujeme *Over operator* (viz Pseudokód 2).

2.8.4 MIP

Maximum intensity projection vykreslí hodnotu s největší intenzitou. Tu si ukládáme v proměnné *output*.

Algorithm 2 Back-to-front metoda

```
1: procedure BackToFront
2: loop:
3:     data ← texBack
4:     data ← data ∗ fraction
5:     color ← get value from tf
6:
7:     cout ← cout ∗ (1 − color.alpha) + color ∗ color.alpha
8:     texBack ← texBack − direction
9: end procedure
```

Algorithm 3 Maximum intensity projection metoda

```
1: procedure MaximumIntensityProjection
2: loop:
3:     alpha ← texFront
4:     if alpha > output then
5:         output ← alpha
6:     end if
7:     texFront ← texFront + (stepSize ∗ direction)
8: color ← alpha
9: end procedure
```

2.8.5 AIP

Z části analýzy víme, že shader reprezentující schéma Average intensity projection musí vykreslovat průměrnou hodnotu dle vrženého paprsku. Tu si ukládáme v proměnné *output*, kterou na konci cyklu vydělíme počtem *count*.

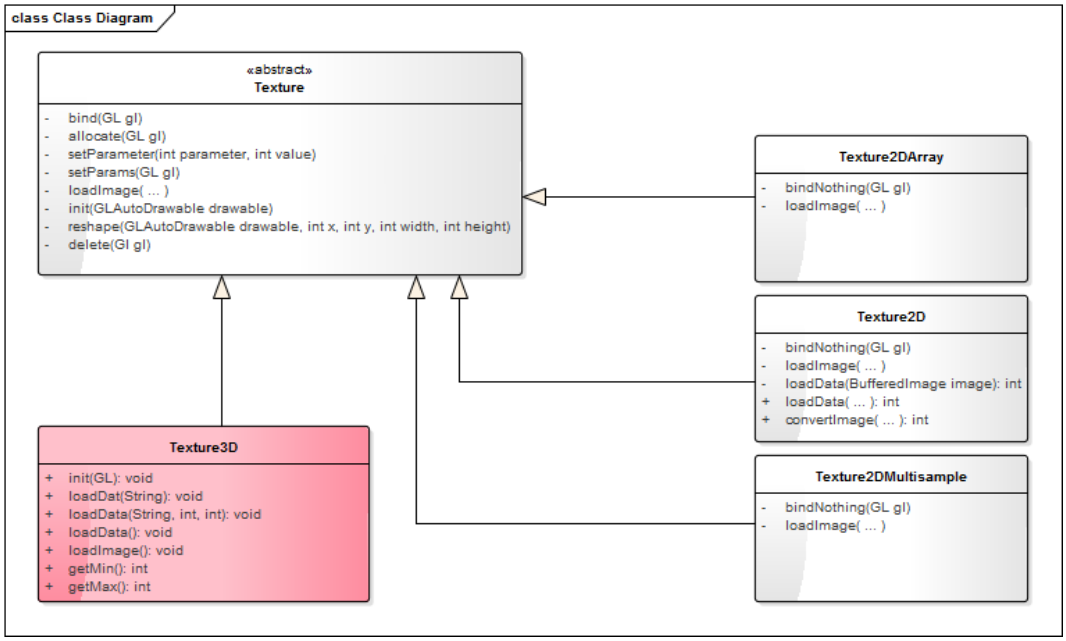
Algorithm 4 Average intensity projection metoda

```
1: procedure AverageIntensityProjection
2: loop:
3:     alpha ← texFront
4:     if alpha > output then
5:         output ← alpha
6:     end if
7:     texFront ← texFront + (stepSize ∗ direction)
8:     count ← count + 1
9: color ← alpha/count
10: end procedure
```

Kapitola 3

Implementace

V této kapitole se budu věnovat implementační části práce, jejíž kód naleznete na přiloženém CD. Před tím, než je možné volumetrická data zobrazovat a obecně s nimi pracovat, je zapotřebí implementovat třídu pracující s 3D texturou. Knihovna Tiger dosud obsahovala třídy pro práci s 2D texturami. Mým úkolem bylo vytvořit třídu pracující s 3D texturami, která obsahuje metody pracující s volumetrickými daty - načtení, zpracování a nahrávání těchto dat na grafickou kartu. Na následujícím Obrázku 3.1 můžete vidět diagram analytických tříd, jenž vystihuje danou situaci. Implementované či mnou upravované metody jsou znázorněny znakem +, stejně tak jsem použil barevné rozlišení pro navrženou třídu.



Obrázek 3.1: Class diagram tříd pracujících s texturami

3.1 Implementace třídy Texture3D

Před tím, než budeme rozebírat jednotlivé metody, specifikujme si formát dat, s kterým budeme pracovat. Jako testovací data, která bych měl vykreslit, jsem obdržel 16bitová data ve stupních šedi s rozlišením 256 x 256 x 113 ze Standfordského data archivu[1]. Dále jsem dle zadání měl využít celkem čtyř volumetrických modelů. Ty jsem čerpal z knihoven volumetrických[2] a medicínských[3] dat.

Implementaci třídy Texture3D začnu popisovat od inicializace textur, která probíhá v metodě init(). Jedná se o překrytí metody ze třídy Texture. V této metodě alokujeme a vygenerujeme názvy (unikátní integer hodnoty) textur, které potom navážeme (angl. bind) s existujícími texturami, specifikujeme parametry a po této činnosti jsou textury připravené k nahrání na grafickou kartu. Jedním z nastavených parametrů je buffer *imageData*, který by měl obsahovat data, která chceme vykreslit. Ta musíme nejdříve dle typu dat načíst metodou loadData(String path, String name, int start, int end) či loadDat(String path).

Metody na základě formátu načtou data, která se nacházejí na disku (cesta k souboru se nachází v parametru path). Další dva parametry u první z metod udávají číslo prvního a posledního snímku, který se bude zpracovávat. Z dat získáme parametry nutné pro vykreslení (typ, formát atd.) a následně data uložíme do instanční proměnné imageData a můžeme je nahrát na grafickou kartu. O to se stará metoda loadImage(...)[4]. Její parametry obsahují informace o nahrávaných datech. Metodu opět překrýváme z abstraktní třídy Texture.

Přetížená metoda loadData() bez parametrů nahraje do instanční proměnné prázdnou texturu - nastavíme imageData jako null. Tuto metodu lze použít pokud bychom chtěli data na grafickou kartu nahrávat po jednotlivých řezech.

3.2 Načítání volumetrických modelů

Nyní již naimplementovanou třídu budeme využívat u praktických ukázek - příkladů. Následující podsekce popisují tyto příklady a ukazují jak se volumetrická data vykreslí pro jednotlivé případy. Abychom využili všech volumetrických modelů a nemuseli zbytečně zasahovat do kódu, můžeme využít třídy TextureLoader v balíčku tiger.core. Tato třída nám pomocí několika metod vykreslí vybraný volumetrický model do 3D prostředí (viz 3.5).

Začněme statickou metodou create(). Jedná se o statickou metodu jejíž návratová hodnota je právě 3D textura (Texture3D). Jedno *vlákno* nám zde nabídne kontextovou nabídku

[1] <http://graphics.stanford.edu/data/voldata/>

[2] <http://www.volvis.org/>

[3] <http://lgdv.cs.fau.de/External/vollib/>

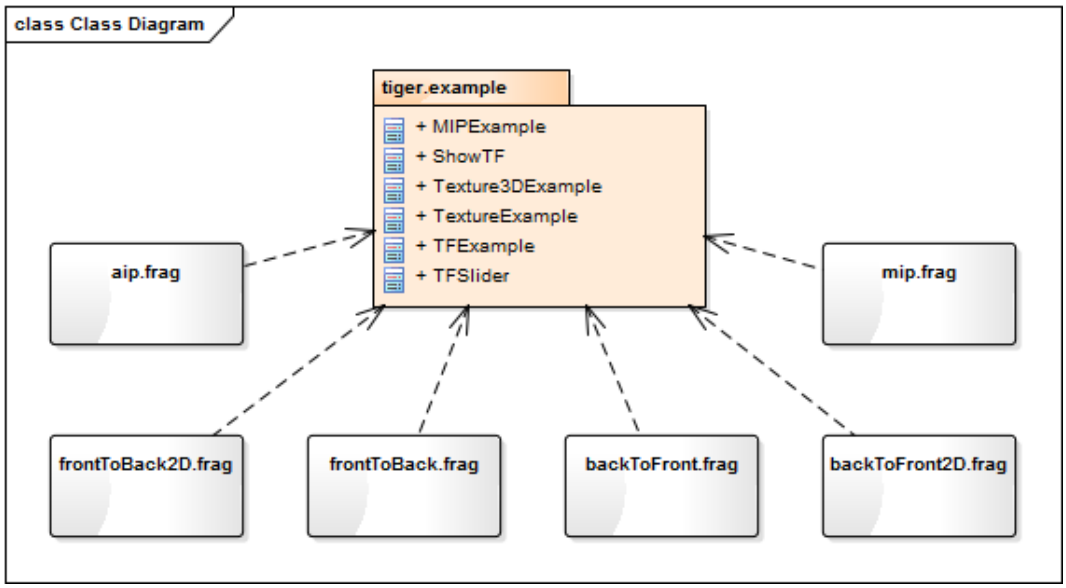
[4] formální parametry metody zde z úsporných důvodů neuvádíme

výběru souborů s volumetrickými daty. Vlákno je uspáno až do doby, kdy nám akce vybrání některého ze souborů zmíněné vlákno probudí. Nyní již nahrajeme data na grafickou kartu pomocí privátní metody handleFile(File file)[5], jejíž logika určí parametry textury, které jsou potřeba pro správné vykreslení.

Posloupnost činností potřebných k vykreslení volumetrického modelu končí zavoláním statické metody create(...). Její parametry vám umožní specifikovat fragment shader a dále *transfer funkci*, pomocí které lze data ovlivnit. Více na toto téma se dozvíte v sekci 3.6.

3.3 Implementace kompozičních schémat

Na základě analýzy (viz 2.3) byly navrženy shadery reprezentující příslušná kompoziční schémata.

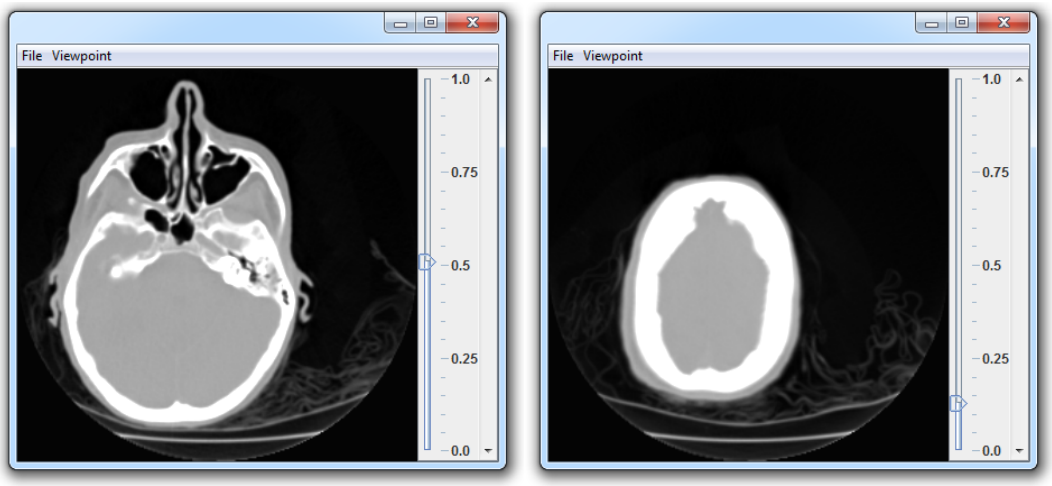


Obrázek 3.2: Diagram tříd a balíčků figurujících při použití kompozičních schémat

3.4 Vykreslení dat po jednotlivých řezech

V prvním příkladu vykreslení volumetrických dat se objekt lebky bude vykreslovat jako 2D průřez.

[5] u volumetrických modelů reprezentovanými sérií souborů načteme data výběrem pouze souboru prvního



Obrázek 3.3: Jednotlivé řezy lebkou

Pro realizaci tohoto úkolu je nejprve nezbytné inicializovat novou 3D texturu, kterou pomocí již implementované třídy Texture3D nahrajeme na grafickou kartu. Hloubku zobrazovaného řezu, budeme určovat pomocí Sliderů, konkrétně FloatSlideru, jehož hodnotu předáme fragment shaderu jako *uniform* proměnnou.

Jedním průchodem (angl. pass) vykreslíme data tak, že instanci třídy Pass předáme nezbytné objekty (texturu, fragmentStream, proměnné atd.). Tuto instanci přidáme do efektu (angl. Effect), který je vykreslen v okně (viz Obrázek 3.3). Proces bude názornější po prohlédnutí kódu Texture3DExample.java v balíčku tiger.examples, kde názvy použitých metod jsou více než samovysvětlující.

3.5 Vizualizace dat ve 3D prostoru

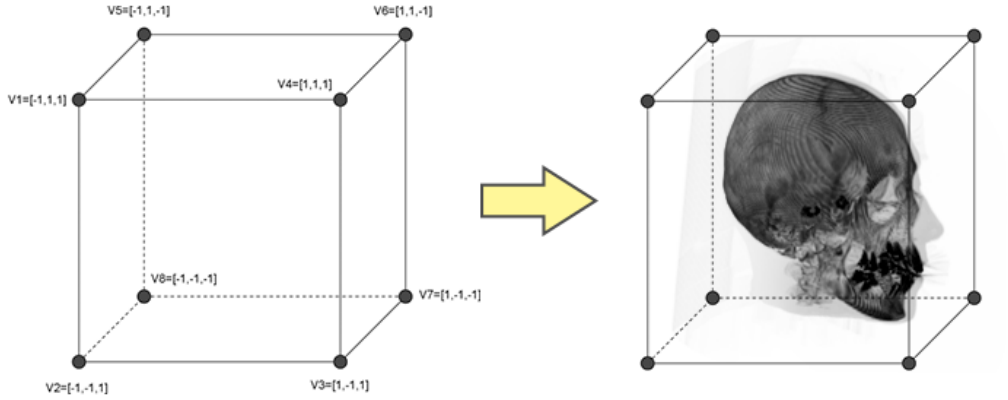
Pro vykreslení dat ve 3D prostoru použijeme scénu krychle. Vytvoříme novou geometrii, do které přidáme nejprve vrcholy a následně plošky krychle. Plošky jsou tvořeny vrcholy, a co je podstatné, při definování musí být dodrženo správné pořadí vrcholů, které je dáno proti směru hodinových ručiček (při pohledu proti normálovému vektoru plošky). Formát volumetrických dat není vždy stejný, například nerovnoměrný poměr stran zapříčiní deformaci dat v jednotkové krychli. Tomu lze zabránit nastavením texturovacích souřadnic u této geometrie. Informaci texturovacích souřadnic nesou jednotlivé vrcholy krychle, jak můžeme vidět na následující ukázce kódu.

```
Vertex vertex = new Vertex(0.0, 0.0, 0.0);
vertex.setTexCoords(0.0, 0.0, 0.0);
```

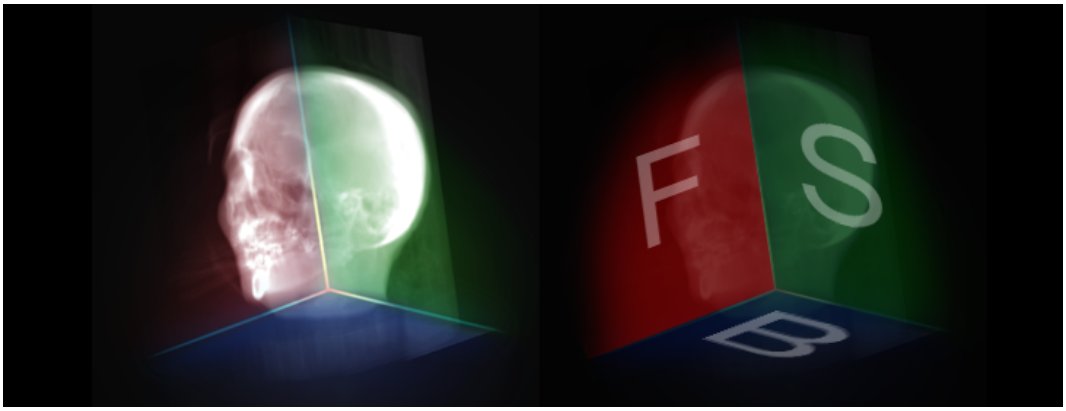
Listing 3.1: Základní nastavení texturovacích souřadnic u daného vrcholu

Po vytvoření scény s geometrií krychle (viz Obrázek 3.4) nahrajeme na grafickou kartu potřebné textury s volumetrickými daty. Pomocí frame bufferu vykreslíme krychli (přední a

zadní stěnu).



Obrázek 3.4: Princip zobrazení dat ve scéně, jednotková krychle s texturovacími souřadnicemi (vlevo), objekt lebky ve scéně (vpravo)



Obrázek 3.5: 3D zobrazení, jednotlivé stěny krychle jsou barevně nasvíceny pro lepší přehlednost, přední strana krychle (F), spodní strana krychle (B), boční strana krychle (S).

Poté postupujeme jako v předchozím případě. Na Obrázku 3.5 jsou barevně odlišeny plošky krychle. Pro vykreslení dat jsem použil Average Intensity Projection schéma (viz 2.3.4).

3.6 Transfer funkce

Z analýzy víme, že vykreslení dat lze ovlivnit pomocí transfer funkcí. Transfer funkce je mapování, kterým rozhodneme, jak rozlišit různé struktury v objektu na základě přiřazení těchto struktur určitým hodnotám v transfer funkci (např barvě). Účel této funkce je tedy vizualizovat a zdůraznit právě ty struktury, které uživatel chce zobrazit. Nejjednodušší a nejpoužívanější transfer funkce jsou funkce jednodimenzionální, označované jako 1D transfer

funkce. Ty mapují rozmezí hodnot dat na barvu a (ne)průhlednost.

Mým úkolem bylo implementovat a použít transfer funkci, a tedy ovlivnit vykreslení jednotlivých volumetrických modelů.

3.6.1 Příprava

Transfer funkce je reprezentována obrázkem podporující alfa kanál. Jako formát obrázků jsem si zvolil Portable Network Graphics (přípona PNG). Obrázek je třeba nahrát na grafickou kartu jako 2D texturu. Nahrávání tohoto typu jsem nejprve musel naimplementovat.

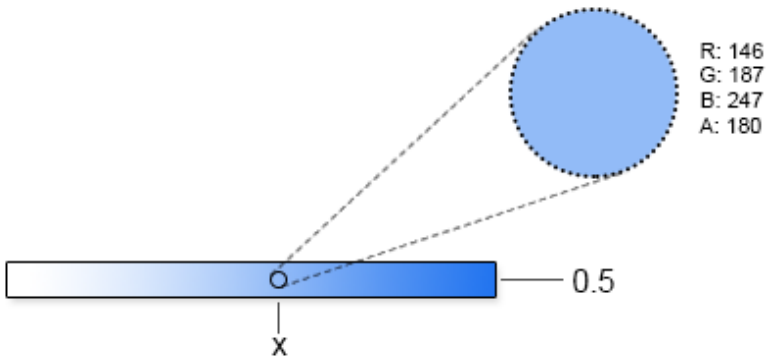
Metoda *loadData(BufferedImage image)* ve třídě Texture2D je právě metodou, která na základě formátu nahraje data do bufferu, který můžeme přenést na GPU. Tuto metodu jsem tedy upravil a společně s tím přidal konverzi, která převede formáty RGBA či ARGB na formát stejný.

3.6.2 1D transfer funkce

Kód 3.2 nám ukazuje příklad použití 1D transfer funkce. Vektor barvy se určí na základě hodnot skalárních dat, které nás odkazují na pozici x v Obrázku 3.6.

```
vec4 color = texture2D(tf, vec2(data, 0.5));
```

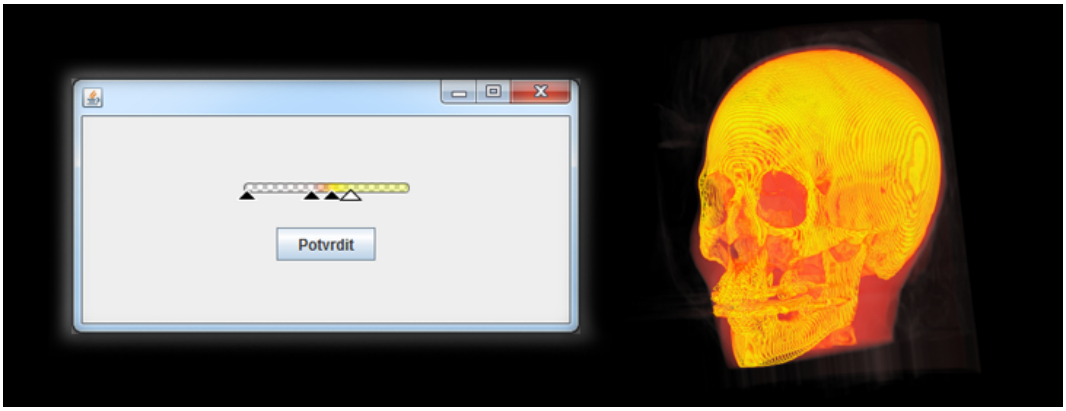
Listing 3.2: Získání vektoru barvy na základě hodnoty volumetrických dat (*data*) a transfer funkce (*tf*)



Obrázek 3.6: Význam transfer funkce

Dále jsem implementoval program, který pomocí komponenty *Gradient Slider* [13] umožní vytvoření jakékoliv 1D transfer funkce (viz Obrázek 3.7). Tato komponenta může mít několik odrážek (angl. thumb). Každá z odrážek nese informaci o barvě a průhlednosti v daném bodě, a také si pamatuje svou pozici. Ke komponentě jsem doimplementoval metodu, která

lineární interpolací dodefinuje hodnoty barev mezi těmito odrážkami. Tím vznikne nový obrázek reprezentující transfer funkci.



Obrázek 3.7: Komponenta Gradient Slider (vlevo), výsledek použití transfer funkce (vpravo)

3.6.3 2D transfer funkce

Ve 2D transfer funkci se data mapují také na základě velikosti gradientu. Gradient se vypočítá dle následujícího vzorce [25]:

$$g_x = \frac{f(x-1,y,z)-f(x+1,y,z)}{2}, \quad g_y = \frac{f(x,y-1,z)-f(x,y+1,z)}{2}, \quad g_z = \frac{f(x,y,z-1)-f(x+1,y,z+1)}{2}. \tag{3.1}$$

K tomu jsem přistoupil následujícím způsobem:

```
1  ...
2
3  float vValueXa = texture3D(vTexture, texFront+vec3(+stepSize,0,0)).x;
4  float vValueXb = texture3D(vTexture, texFront+vec3(-stepSize,0,0)).x;
5  float vValueYa = texture3D(vTexture, texFront+vec3(0,+stepSize,0)).x;
6  float vValueYb = texture3D(vTexture, texFront+vec3(0,-stepSize,0)).x;
7  float vValueZa = texture3D(vTexture, texFront+vec3(0,0,+stepSize)).x;
8  float vValueZb = texture3D(vTexture, texFront+vec3(0,0,-stepSize)).x;
9
10 gradient = vec3(
11                 (vValueXb-vValueXa)/2.0,
12                 (vValueYb-vValueYa)/2.0,
13                 (vValueZb-vValueZa)/2.0);
14
15 float gradientMag = length(gradient);
16
17 ...
```

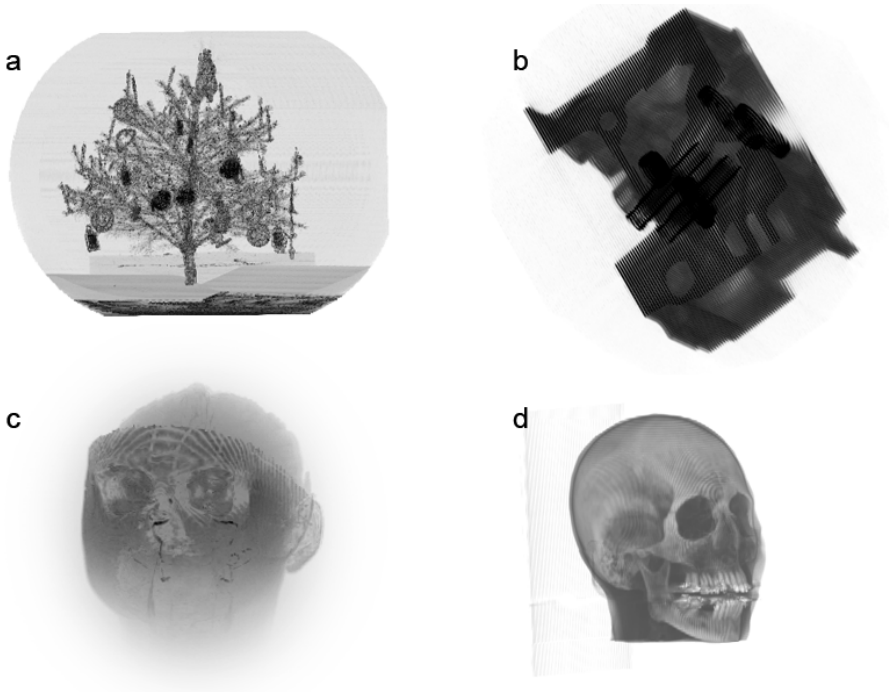
Listing 3.3: Výpočet velikosti gradientu

Na řádku 3 - 8 v Kódu 3.3 načteme hodnoty okolních voxelů (v každém směru), ty následně použijeme ve vzorci pro výpočet gradientu. Na řádku 15 poté zavoláme metodu *length(gradient)*, která nám vrátí velikost vektoru.

Kapitola 4

Testování

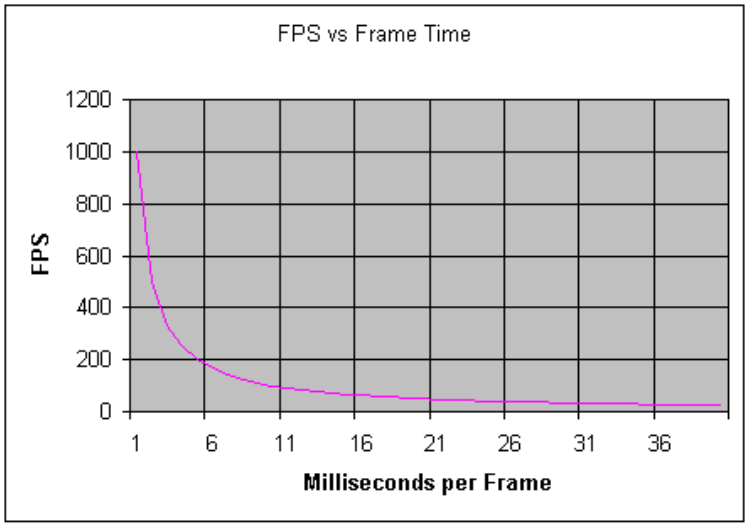
V této kapitole otestujeme výkon metod vykreslování analyzované a implementované v předchozích části. Testování proběhne pro čtyři volumetrické modely různé složitosti (viz Obrázek 4.1).



Obrázek 4.1: Volumetrické modely: vánoční stromek (a), motor (b), otevřená lebka (c), lebka (d), modely byly převedeny do inverzních barev kvůli lepší viditelnosti.

V knihovně Tiger je možnost pro vykreslení hodnot snímků za sekundu (dále jen FPS). Hodnota FPS ale není optimální jednotkou pro měření výkonu zpracování volumetrických dat. Jak můžeme vidět na Obrázku 4.2, změna FPS není lineární. Proto budeme v tomto testování pracovat s veličinou Frame Time - neboli počet milisekund potřebných pro vykreslení

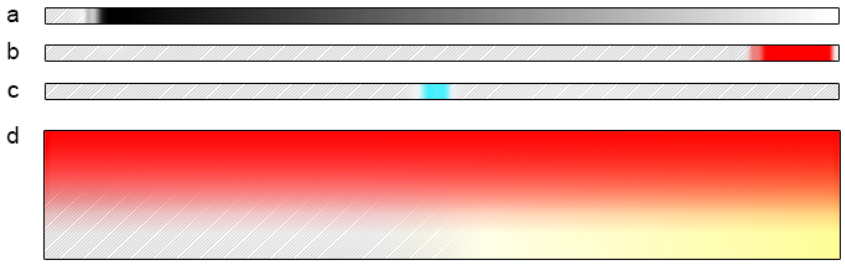
jednoho snímku. Více o tomto téma pojednává [23].



Obrázek 4.2: Obrázek reprezentující graf závisloti FPS na Frame Time. Zdroj [23].

4.1 Výkonostní testování

Výkon metod front-to-back a back-to-front budeme testovat pro jednotlivé volumetrické modely za použití transfer funkcí zobrazených v Obrázku 4.3. Šrafované části reprezentují průhlednou barvu. Pro lepší přehlednost byly rozměry funkcí v obrázku upraveny.



Obrázek 4.3: Transfer funkce: 1D transfer funkce a (a), 1D transfer funkce b (b), 1D transfer funkce c (c), 2D transfer funkce d (d). Průhledná barva šrafovaně.

Jednotlivé transfer funkce z Obrázku 4.3 se liší v rozmístění barev. Vliv tohoto rozmístění si vysvětlíme níže.

Transfer funkce *a*
První z našich funkcí soustřeďuje nejvyšší hodnoty černé barvy u levého okraje a poté postupným přechodem přechází v barvu bílou.

U tohoto rozvržení předpokládáme, že front-to-back metoda bude mít lepší výkon, díky faktu, že hodnota barvy se pro velký rozptyl nižších hodnot akumuluje rychle. Tím pádem se výpočet ve front-to-back metodě může ukončit dříve. V back-to-front metodě musíme výpočet provést pro všechny vzorky podél paprsku.

Obrázek 4.4: Výsledky testování výkonu pro transfer funkci *a*. Použité volumetrické modely: lebka (vlevo nahoře), motor (vpravo nahoře), vánoční stromek (vlevo dole) a otevřená lebka (vpravo dole)

Z grafů je patrné, že lepších výkonů dosahuje front-to-back metoda, která pro každé měření vyprodukovala lepší výsledek.

Transfer funkce *b*
Pro druhé měření vezmeme opačnou transfer funkci. Ta má malý rozptyl hodnot, které se soustřeďují v oblasti vysokých hodnot skalárních dat. To by mělo zapříčinit vykreslení pouze dat s nejvyšší prioritou. Očekáváme tedy podobný výkon vykreslení.

Podle předpokladu jsme dosáhli podobných měření (viz 4.5). Jednotlivé výsledky velmi kolísají, což může být zapříčiněno úhlem pohledu, kdy se data s vysokou prioritou mohou vzájemně zakrývat či nikoliv.

Obrázek 4.5: Výsledky testování výkonu pro transfer funkci *b*. Použité volumetrické modely: lebka (vlevo nahoře), motor (vpravo nahoře), vánoční stromek (vlevo dole) a otevřená lebka (vpravo dole)

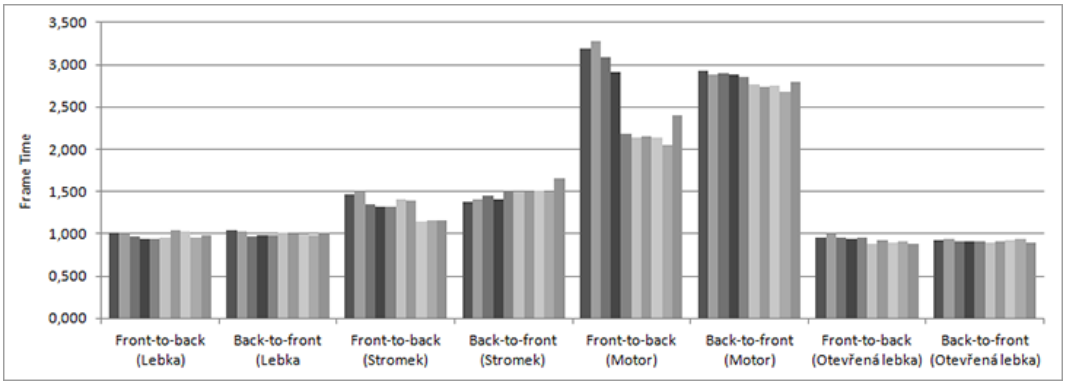
Transfer funkce *c*

Výsledky pro třetí transfer funkci přinesly velmi podobné hodnoty. Pokusíme se tedy ověřit, zda-li rozdíl středních hodnot rozdělení pro metodu front-to-back a rozdělení pro metodu back-to-front je roven 0. K ověření našeho přepokladu neboli hypotézy použijeme párový t-test, který vypočítáme pomocí programu R s hladinou významnosti 0.05 neboli s 95% jistotou výsledku. Testované hodnoty náleží volumetrickému modelu otevřené lebky.

```
Paired t-test

data: f-2-b and b-2-f
t = 1.1309, df = 9, p-value = 0.2873
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.01260431  0.03780431
sample estimates:
mean of the differences
                 0.0126
```

Listing 4.1: Výstup programu R při použití párového t-testu

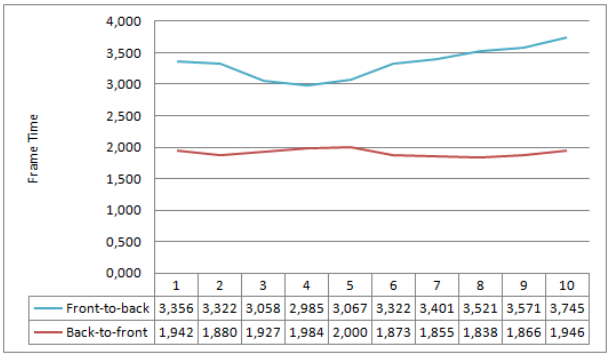


Obrázek 4.6: Graf výkonu jednotlivých měření (sloupce) v závislosti na Frame Time

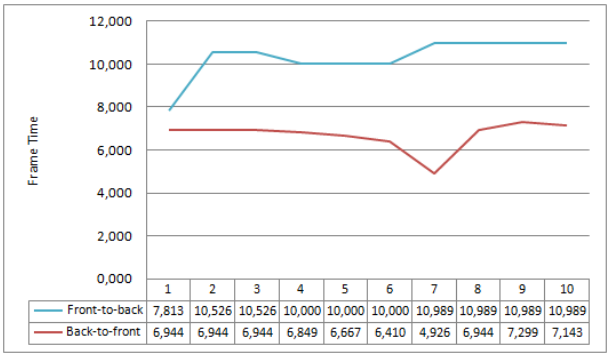
Data jsme zpracovali v párovém t-testu, $t(9) = 1.1309$, $p = 0.2873$. Jelikož pro hodnotu p platí $0.2873 > 0.05$, **hypotézu nezamítáme**.

Transfer funkce *d*

Poslední použitou transfer funkcí byla 2D funkce, kde pro data vypočítáváme velikost gradientu. Pro jednotlivé volumetrické modely byly výsledky testování výkonu následující:



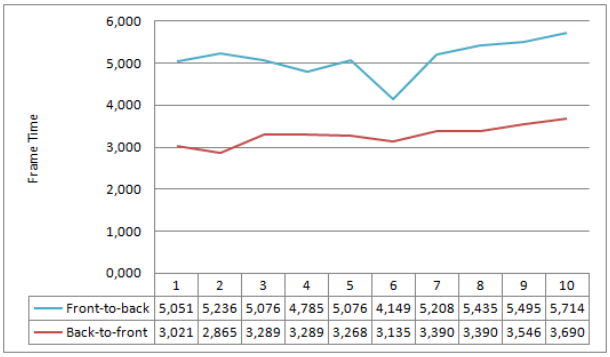
	1	2	3	4	5	6	7	8	9	10
Front-to-back	3,356	3,322	3,058	2,985	3,067	3,322	3,401	3,521	3,571	3,745
Back-to-front	1,942	1,880	1,927	1,984	2,000	1,873	1,855	1,838	1,866	1,946

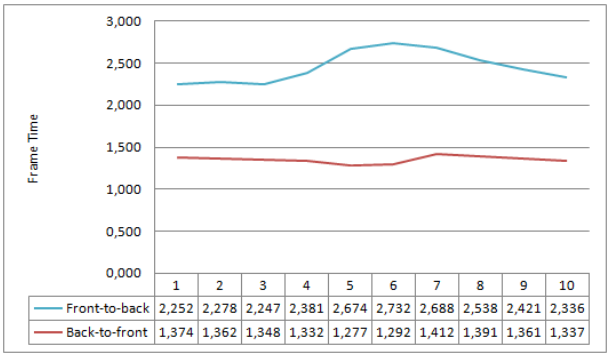


	1	2	3	4	5	6	7	8	9	10
Front-to-back	7,813	10,526	10,526	10,000	10,000	10,000	10,989	10,989	10,989	10,989
Back-to-front	6,944	6,944	6,944	6,849	6,667	6,410	4,926	6,944	7,299	7,143

Obrázek 4.7: Výsledky testování výkonu pro transfer funkci *d*. Použité volumetrické modely: lebka (vlevo), motor (vpravo), vánoční stromek (vlevo dole) a otevřená lebka (vpravo dole)

Z grafů na Obrázcích 4.7 a 4.8 můžeme zjistit, že metoda back-to-front je v tomto případě výkonnější. Důvodem tohoto výsledku je dle mého názoru počet operací v jednotlivých shaderech, kde pro front-to-back metodu musíme vypočítávat hodnoty barvy a průhlednosti. V kombinaci s výpočtem velikosti gradientu tato metoda nemusí být výhodná. Možným řešením je předpočítání hodnoty velikosti gradientu na CPU, kde tyto hodnoty nahrajeme na grafickou kartu jako samotnou texturu. V shaderu se pouze odkážeme na daný voxel a neztrácíme tak výpočetní výkon.

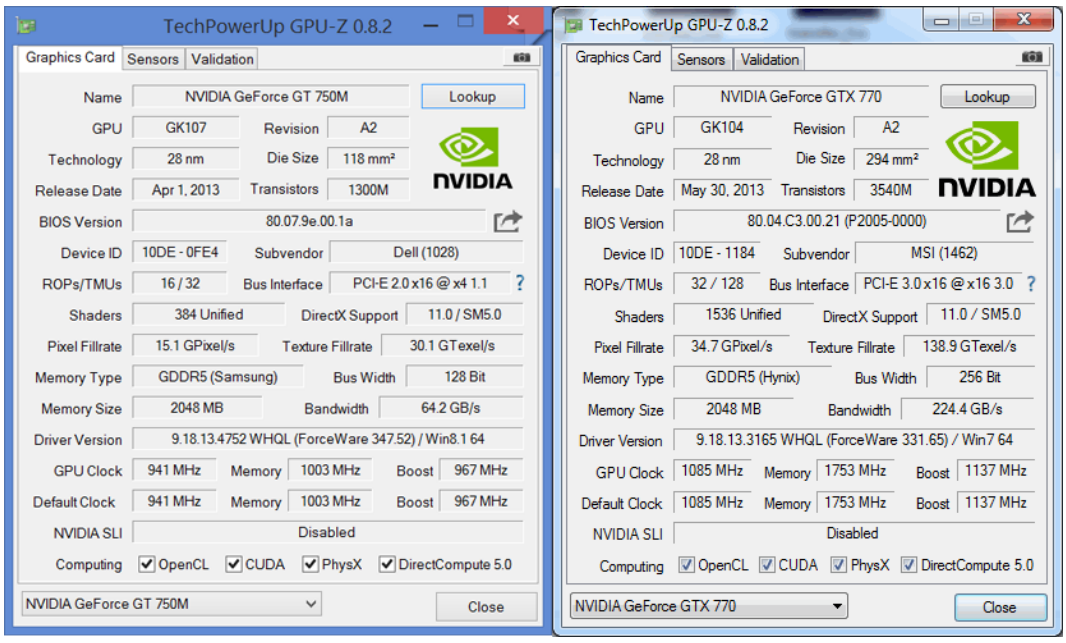




Obrázek 4.8: Výsledky testování výkonu pro transfer funkci *d*. Použité volumetrické modely: vánoční stromek (vlevo) a otevřená lebka (vpravo)

4.2 Testování funkčnosti

Všechny implementované části byly testovány na třech nezávislých počítačích s grafickým čipem nVidia. Jednalo se o můj osobní počítač, notebook Dell Inspiron a desktopový počítač v učebně *Triangle*.



Obrázek 4.9: Informace o GPU: osobní počítač (vlevo), notebook (vpravo)

Všechny implementované části proběhly bez problémů, což podle mého názoru bude zapřičiněno velmi rozšířeným standartem OpenGL, který je podporovaný velkým procentem grafických karet na trhu.

Kapitola 5

Závěr

Cíl této práce, seznámit se s přímým vykreslováním volumetrických dat a s knihovnou dodanou vedoucím práce, jsem splnil. Na základě analýzy, jsem navrhl a implementoval schémata front-to-back a bact-to-front (viz Pseudokódy 1 a 2), Average a Maximum intensity projection (viz Pseudokódy 3 a 4). Dále jsem realizoval vykreslování dat po jednotlivých řezech a také jsem volumetrická data vykreslil do scény krychle. Data je možno ovlivnit 1D a 2D transfer funkcí, jejichž princip jsme si vysvětlili v sekci 2.4 a realizaci v sekci 3.6. Dále je možné vytvořit pomocí komponenty *Gradient Slider* libovolnou 1D transfer funkci a následně jí použít k vykreslení.

Výsledkem této práce je tedy kód schopný vykreslovat data po jednotlivých řezech, vykreslovat data do vytvořené 3D scény, kde lze ovlivnit frekvenci vzorkování a dále jsou implementovány metody Maximum Intensity Projection a Average Intensity Projection, tedy různá schémata vykreslení dat. Nejdůležitější ze schémat, tedy front-to-back a back-to-front metody, jsem nejen realizoval, ale v kapitole 4 jsem je také otestoval. Výkony těchto metod se liší na základě použitých transfer funkcí. Při dostatečně malém rozptylu barev uprostřed transfer funkce mají obě metody stejný výkon. V případech, kdy má transfer funkce dostatečný rozptyl, který dovolí metodě front-to-back ukončit cyklus výpočtu hodnoty světla dle paprsku dříve, má tato metoda lepší výkon. Při použití 2D funkce toto tvrzení nelze dokázat.

Celkem je možné použít až sedm volumetrických modelů, z toho čtyři jsem využil při testování výkonu. U některých modelů je třeba manuálně změnit texturovací souřadnice kvůli neobvyklým rozměrům (konvice, otevřená lebka).

5.1 Možné pokračování práce

Knihovnu Tiger je možné rozšířit o další funkcionalitu. Zajímavá by mohla být implementace vícedimenzionálních transfer funkcí či předpočítávání gradientních textur. Také lze přidat další metody vykreslování, neboť v mé práci jsem použil pouze Raycasting.

Literatura

[1] Prezentace - Vizualizace, Direct Volume Rendering. <http://leyfi.felk.cvut.cz/courses/viz/lecture05/VIS-Modules-06-Direct_Volume_Rendering.pdf>. Datum: 2015-05-19.

[2] A New Rating Scale for Age-Related White Matter Changes Applicable to MRI and CT. <http://stroke.ahajournals.org/content/32/6/1318/F1.expansion.html>. Datum: 2015-05-19.

[3] Maximum Intensity Projection. <http://www.livevolume.com/?page_id=139&lang=en>. Datum: 2015-05-19.

[4] Ray Casting. <http://commons.wikimedia.org/wiki/File:Volume_ray_casting.png>. Datum: 2015-05-19.

[5] Reprezentace 3D teles. <https://leyfi.felk.cvut.cz/courses/mga/lectures/05_reprezentace_3d_teles.pdf>. Datum: 2015-05-19.

[6] Definition of the Definite Integral. <http://commons.wikimedia.org/wiki/File:Riemann-Sum-right-hand.png>. Datum: 2015-05-19.

[7] 3D MEDICAL DATA VISUALIZATION TOOLKIT. <http://www.dca.fee.unicamp.br/courses/IA369E/2s2010/projects/vidalon_watanabe/index.htm>. Datum: 2015-05-19.

[8] FERNANDO, R. – HAINES, E. – SWEENEY, T. GPU gems: programming techniques, tips, and tricks for real-time graphics. *Dimensions*. 2001, 7, 4, s. 816.

[9] Framebuffer. Framebuffer — Wikipedia, The Free Encyclopedia, 2015. Dostupné z: <http://en.wikipedia.org/wiki/Framebuffer>. [Online; Datum: 2015-05-19].

[10] HADWIGER, M. et al. *Real-time volume graphics*. 888 Worcester Street, Suite 230 : AK Peters, Ltd., 2006.

[11] Ing. Ladislav Cmolik, Ph.D. DCGI Ing. Ladislav Cmolik, Ph.D. Profile, 2015. Dostupné z: <http://dcgi.felk.cvut.cz/home/cmolikl/>. [Online; Datum: 2015-05-19].

[12] Java OpenGL. Java OpenGL — Wikipedia, The Free Encyclopedia, 2015. Dostupné z: <http://en.wikipedia.org/wiki/Java_OpenGL>. [Online; Datum: 2015-05-19].

[13] Java.net. GradientSlider, 2015. Dostupné z: <`https://javagraphics.java.net/doc/com/bric/swing/GradientSlider.html`>. [Online; Datum: 2015-05-19].

[14] KINDLMANN, G. Transfer functions in direct volume rendering: Design, interface, interaction. *Course notes of ACM SIGGRAPH*. 2002.

[15] KNISS, J. – KINDLMANN, G. – HANSEN, C. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of the conference on Visualization'01*, s. 255–262. IEEE Computer Society, 2001.

[16] LACROUTE, P. – LEVOY, M. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, s. 451–458. ACM, 1994.

[17] LEVOY, M. Display of surfaces from volume data. *Computer Graphics and Applications, IEEE*. 1988, 8, 3, s. 29–37.

[18] MAX, N. Optical models for direct volume rendering. *Visualization and Computer Graphics, IEEE Transactions on*. 1995, 1, 2.

[19] OpenGL. OpenGL — Wikipedia, The Free Encyclopedia, 2015. Dostupné z: <`http://cs.wikipedia.org/wiki/OpenGL`>. [Online; Datum: 2015-05-19].

[20] OpenGL Shading Language. OpenGL Shading Language Official Webpage, 2015. Dostupné z: <`https://www.opengl.org/documentation/glsl/`>. [Online; Datum: 2015-05-19].

[21] Petr Felkel. OpenGL jako automat - prezentace, 2015. Dostupné z: <`https://cent.felk.cvut.cz/courses/PGR/lectures/01-INTRO.pdf`>. [Online; Datum: 2015-05-19].

[22] Petr Felkel. PGR - prezentace, 2015. Dostupné z: <`https://cent.felk.cvut.cz/courses/PGR/lectures/08a-zobrRetezec.pdf`>. [Online; Datum: 2015-05-19].

[23] Robert Dunlop. FPS: A common yet flawed metric of game performance, 2015. Dostupné z: <`https://www.mvps.org/directx/articles/fps_versus_frame_time.htm`>. [Online; Datum: 2015-05-19].

[24] ROTH, S. D. Ray casting for modeling solids. *Computer graphics and image processing*. 1982, 18, 2, s. 109–144.

[25] Scientific Visualization. Volume Rendering Shading and Transfer Function, 2015. Dostupné z: <`http://www.cs.kent.edu/~zhao/vis11/lectures/ShadingAndTransfunction.pdf`>. [Online; Datum: 2015-05-19].

[26] The graphic pipeline. The graphic pipeline — Wikipedia, The Free Encyclopedia, 2015. Dostupné z: <`http://en.wikipedia.org/wiki/Graphics_pipeline`>. [Online; Datum: 2015-05-19].

[27] ZDROJEWSKA, D. Real time rendering of heterogeneous fog based on the graphics hardware acceleration. *Proceedings of CESCG*. 2004, 4.

Příloha A

Obsah přiloženého CD

- CD
  - `readme.txt` ..... Manuál
  - `repozitar` ..... Repozitář obsahující implementaci
  - `files` ..... Složka s volumetrickými modely a transfer funkcemi
  - `Java` ..... Java prostředí pro spustitelné soubory bat
  - `lib` ..... Složka s potřebnými knihovnami
  - `tiger` ..... Knihovna Tiger
  - `thesis` ..... Zdrojové kódy textu bakalářské práce
    - `figures` ..... Složka s obrázky použitými v textu bakalářské práce
    - `grolljak-thesis-2015.pdf` ..... PDF soubor bakalářské práce
  - `priklady.bat` ..... Spustitelné příklady