VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Katedra informatiky

Vizualizace volumetrických dat

Visualization of Volumetric Data

2013 Vojtěch Uher

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Vojtěch Uher**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Vizualizace volumetrických dat
Visualization of Volumetric Data

Zásady pro vypracování:

Cílem práce je navrhnout a implementovat knihovnu pro vizualizaci volumetrických dat. Výsledná implementace by měla být schopna zobrazovat volumetrická data pomocí netriviálního volumetrického renderingu, generovat libovolné řezy a pro výpočty využívat GPU.

1. Seznamte se s metodami zobrazování volumetrických dat (vyjděte z [1]).
2. Převeďte dodaný datový soubor CT řezů do vhodné reprezentace.
3. Naimplementujte vizualizaci pomocí OpenGL a CUDA, popřípadě využijte OptiX.
4. Řešení doplňte o uživatelské řezy a jejich vhodné zobrazení.
5. Funkčnost výsledné aplikace demonstrujte na dodaném datovém souboru.

Seznam doporučené odborné literatury:

[1] KROES, Thomas, POST, Frits H., BOTHA, Charl P. Exposure Render: An interactive photo-realistic volume rendering framework. PLoS. 2012.
[2] FERNANDO, Randima. GPU Gems. 2004. 816 s. ISBN 978-0321228321.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Tomáš Fabián**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2013

.....
Vojtěch Uher

Poděkování

Děkuji svému vedoucímu diplomové práce Ing. Tomáši Fabiánovi za odborné vedení a podněty ke zlepšení závěrečné práce.

Abstrakt

Práce shrnuje současné techniky zobrazování volumetrických dat a popisuje implementaci jedné vlastní metody. Cílem práce bylo vytvoření netriviálního renderovacího algoritmu pro vizualizaci medicínských řezů získaných z lékařských přístrojů. Text hodnotí v několika kapitolách aktuální problematiku zpracovávání objemových dat, v další části pak komentuje vývoj metody vlastní vycházející z představené teorie. Implementovaný renderer vychází z principu přímého volumetrického zobrazování (DVR) a využívá moderních stochastických postupů (Monte Carlo) k efektivnějšímu běhu. Poslední částí práce je pak akcelerace algoritmu na GPU s využitím architektury CUDA a představení měření a dosažených výsledků.

Klíčová slova

Direct volume rendering, Volumetrický Ray Casting, C++, CUDA, OpenGL, Ray Marching, Woodcock tracking, Globální osvětlovací model

Abstract

This thesis summarizes present techniques for rendering volumetric data and describes implementation of our own method. The aim of thesis was to create nontrivial rendering algorithm for visualization of volumetric slices obtained from medical devices. This text is commenting actual issues of processing volumetric data in few chapters. The next part is about developing of our method which is based on the discussed theory. Render follows the principles of DVR and uses modern Monte Carlo algorithms to make it more efficient. The last part of the thesis is about accelerating of the render on a GPU via CUDA architecture and summarizing measured values and obtained results.

Key words

Direct volume rendering, Volume Ray Casting, C++, CUDA, OpenGL, Ray Marching, Woodcock tracking, Global illumination model

Seznam použitých symbolů a zkratek

BRDF – Bidirectional Reflection Distribution Function (obousměrná distribuční funkce)
CPU – Central Processing Unit (procesor)
CT – Computed Tomography (počítačová tomografie)
CUDA – Compute Unified Device Architecture (výpočetní architektura pro GPU)
DVR – Direct Volume Rendering (přímé volumetrické zobrazování)
GPU – Graphic Processing Unit (grafická karta)
IS – Importace Sampling (stochastické vzorkování osvětlení)
MC – Monte Carlo (stochastický algoritmus)
MIP – Maximum Intensity Projection (metoda promítající maximální intenzitu)
RC – Ray Casting (vrhání paprsků)
RM – Ray Marching (vzorkovací alg. s konstantním krokem)
RT – Ray Traycing (metoda zpětného sledování paprsků)
SS – Super Sampling
VRC – Volume Ray Casting (volumetrický RC)

Obsah

1 Úvod

Zpracování medicínských dat je problematika řešená minimálně posledních 30 let. Od vynalezení elektronických lékařských přístrojů se vědci po celém světě zabývají otázkou rychlého strojového zpracování takto pořízených dat, aby se lékařská praxe přesunula od fotografií ke grafickým vizualizacím. Podobné nástroje se již dnes v lékařství používají především v radiologii a diagnostice, ale vzhledem ke stále se zpřesňujícím přístrojům je proces vizualizace takovýchto dat nadále předmětem výzkumu z pohledu efektivního a rychlého zpracování.

Cílem této práce je vizualizace volumetrických dat. Volumetrická data jsou data popisující objekty nejen povrchově, ale také objemově. Víme tedy, jak jsou dané objekty popsané v jakýchkoliv úrovních "zanoření do hloubky" z hlediska vnitřní struktury. Přístroje používané v lékařství, jako jsou rentgen, magnetická rezonance, ultrazvuk nebo CT, produkují právě taková data. Můžeme tedy graficky znázornit útroby pacienta bez nutnosti jej operovat, či ručně zkoumat fotografie.

Objemová data však nesouvisí výhradně s lékařstvím. Můžeme se s nimi setkat také ve strojírenství u analýzy části strojů, vizualizací teplot či tlaku, v metalurgii při zkoumání vlastností materiálů či vnitřních vad odlitků. V počítačové grafice a simulacích mohou být zdrojem volumetrických dat částicové generátory pro zobrazení kouře nebo mračna. V herním a filmovém průmyslu se využívají také volumetrické editory pro modelování objektů popsaných objemem, což se hodí například pro simulace destrukcí předmětů, prostředí a exploze, neboť pak víme, co daná věc "skrývá" vevnitř a na jaké části se může rozpadnout. Zde však bude prezentována vizualizace lékařských snímků především proto, že taková data jsme dostali k dispozici.

Výsledkem práce je knihovna pro zobrazování volumetrických dat pomocí moderních pokročilých metod zobrazování a předvedení funkčnosti na konkrétních datech. Algoritmy by měly pro své fungování využívat grafickou kartu, a to zejména technologii CUDA společnosti NVIDIA, resp. OpenGL. Pro snazší implementaci může být použit framework NVIDIA OptiX, který byl vytvořen jako jádro pro akceleraci Ray Tracingu grafickými kartami s technologií CUDA. Jedná se tedy o renderer promítající skalární objemová data zarovnaná na ekvidistantní 3D mřížce, která je daná sadou medicínských řezů, do 2D barevného obrazu. Aplikace umožňuje vytváření volitelných řezů volumetrickými daty. Volumetrické renderování přináší množství výhod i komplikací, které vycházejí z formy reprezentace objektů a o kterých budou následující kapitoly.

• Struktura práce

Práce je rozdělena na několik části. První kapitola následující za úvodem popisuje související práce. Další dvě jsou zaměřeny na teoretické modely reprezentace a vizualizace volumetrických dat, poslední pak shrnuje popis realizace praktické části vycházející z popsané teorie.

- V 3. kapitole je rozepsána teorie volumetrických dat, jejich reprezentace, souhrn pojmů a značení představující základní terminologii k jasné orientaci v textu. Na dvou stranách jsou tam pak rozebrána zpracovávaná data a jejich vlastnosti.
- Ve 4. kapitole jsou zmíněny metody vizualizace volumetrických dat a jejich srovnání. Budou popsány metody zobrazující povrch i integrační algoritmy. Praktická část práce vychází z článku [1], ve kterém se oba tyto přístupy kombinují.
- 5. kapitola je praktickou částí textu popisující postup realizace diplomové práce a implementace vybraného algoritmu včetně převedení algoritmu na GPU. Závěr této kapitoly se věnuje shrnutí výsledků a měření.

2 Související práce

Zobrazování medicínských dat obvykle vede k dvěma základním úlohám. První je isosurfacing, kde se snažíme najít způsobem podobným hranovým obrazovým detektorů hraniční plochy definující přechod mezi prostředími. Druhá je přímé renderování, kdy pomocí metody vrhání paprsků načítáme barvu napříč objemem. Isosurfacing se nejčastěji řeší některou triangulační metodou jako je Marching Cubes [19] nebo Marching Tetrahedra (popsáno např. v práci [4]). Tyto metody samy o sobě trpí značnými nedostatky, kterými jsou především díry v trojúhelníkové síti nebo množství neoptimálních trojúhelníků. V práci [21] je popsaný algoritmu Regularised Marching Tetrahedra rozšířený o optimalizaci trojúhelníkové sítě pomocí Vertex Clusteringu. V článku [10] byla vysvětlena metoda Stretching Cubes, která se soustředí na jemnější prohledávání objemu a detekci nevýrazných přechodů orgánů s nižší hustotou (např. plíce).

V přímém volumetrickém zobrazování se pak nejčastěji využívají postupy založené na Levoyovém [20] nebo Drebinovém (viz např. [2]) integračním algoritmu, které akumulují barvu nasbíraných vzorků podél paprsku s průhledností. Vzhledem k náročnosti výpočtu a potřebám komplexních řešení se v oblasti volumetrického zobrazování často využívají stochastické algoritmy. Ukázkou Monte Carlo Ray Castingu je renderer popsaný v článku [1], kde Kroes a spol. aplikovali stochastický algoritmus trasování objemu Woodcock tracking popsaný např. v článcích [35, 36], který vzorkuje prostředí podél paprsku na základě vypočtené distribuční funkce. Alternativou k němu je starší Ray Marching (viz [35]), který však prochází objem konstantními skoky. Problém méně výrazných částí v objemu, které jsou Woodcockem kvůli nízké důležitosti vynechávané, řeší v [35] pomocí virtuálních částic, jež objem „zahustí“ tak, aby i méně viditelné části objemu byly ve výsledku zastoupeny. Pro rychlejší průchody rozsáhlých prázdných nebo homogenních prostor se často využívají stromové struktury podprostorů jako Oct-tree, BPS-tree, Kd-tree popsané např. v knize [37] a článku [9], nebo technika popsaná Levoyem v [42].

V článku [1] využili stochastický osvětlovací model a MC výpočet fázové funkce. Pro dosažení realističtějších nestranných modelů osvětlování se využívají tzv. Globální osvětlovací modely (viz [38]). Tradiční Phongův osvětlovací model [15] nerespektuje fyzikální podstatu světla, je spíše empirický. V článku [40] je rozšíření Phonga do podoby nestranného osvětlování. Ambientní složka je v globálních modelech nahrazena více světelnými zdroji různých druhů a tvarů (bodový, rovinný, plošný). Globální modely jsou často realizovány technikou zvanou Importance Sampling uvedenou v článcích [39, 40]. Jedná se o stochastický přístup vzorkující na základě pravděpodobnostních funkcí např. světelné zdroje, BRDF a body na plošných zářičích. Veach v práci [41] popsal rozšiřující techniku Multiple Importance Sampling, která v jednom kroku kombinuje několik vypočtených vzorků osvětlování. V [39] je pak popsaný způsob, jak svítit na scénu mapou prostředí. Při výpočtu osvětlovacího modelu nám pak situaci komplikují homogenní oblasti, ve kterých je nejednoznačná normála. Pro výpočet normály v diskrétním prostoru se obvykle využívá gradientní metoda popsaná např. v [2], u které se parciální derivace aproximují symetrickou diferencí. V homogenních oblastech je však tento odhad nedostatečný a normály vycházejí téměř nulové. Jedním z řešení takovéto situace je vyjít

z distanční funkce popsané Gibsonem v [3], která definuje vzdálenost aktuálního voxelu od okraje. Tuto funkci však nemáme vždy k dispozici. Jinou variantou je potlačení homogenních oblastí vynásobením neprůhlednosti vzorku velikostí gradientu, což je standardní rozšíření Levoyova integrálu (viz [2]). Kroes a kol. v článku [1] pak popisují Monte Carlo techniku zvanou Hybrid Scattering, která se stochasticky na základě velikosti gradientu v bodě rozhoduje, zda se pro výpočet osvětlení použije BRDF, nebo isotropní fázová funkce, která na normále závislá není.

Tato práce realizuje přímý volumetrický rendering, který vychází z Levoyova integrálu, využívající stochastické trasování a IS pro vzorkování světel.

Obrázek 1: Ukázka výsledku z našeho Monte Carlo Ray Castingu

3 Volumetrická data

První úlohou při zpracování volumetrického renderingu je stanovení reprezentace dat, která se mají vizualizovat. Jak bylo řečeno v úvodu, volumetrická data definují model celého objemu objektu. Těleso je tedy popsáno řezy, které představují jakési vrstvy daného objektu skládající se z elementárních jednotek (voxelů). V tomto případě budeme chápat datovou strukturu jako 3D ekvidistantní mřížku voxelů zarovnaných v jejích vrcholech. Každý řez je pak matice diskrétních skalárních dat.

V následujících podkapitolách jsou shrnuty základní termíny, metody reprezentace dat a jejich srovnání, popis a charakter zpracovávaných dat a varianty různých druhů mřížek.

3.1 Souhrn pojmů

Definujme tedy pojmy, jež se budou dále v textu objevovat**:**

- **Voxel** je základní stavební jednotkou volumetrického modelu. Nejčastěji má tvar krychle, ale můžou být použity i např. kvádry, pokud je to vhodné (třeba v případě, že řezy mají v jednom směru vyšší rozlišení než v druhém). Voxel je analogií pixelu v trojrozměrném prostoru.

- **Objem** (volumetrická data) je zde struktura voxelů na 3D mřížce popisující celý objekt (scénu) vycházející z hodnot hustot zaznamenaných na snímcích CT. Objem tedy představuje diskrétní prostor značený $v = \rho(x, y, z)$, kde v je hodnota voxelu (hustoty) pro dané x, y, z.

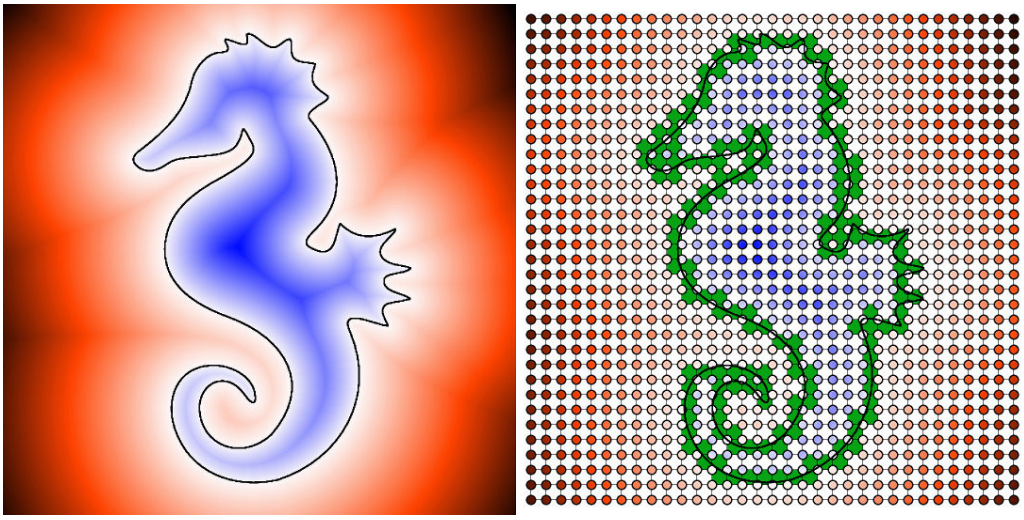
- **Řez** je podmnožina objemu představující jednu vrstvu ρ pro konkrétní z, tedy množina voxelů mající stejnou hodnotu souřadnice z. Obecně by se řez dal definovat i pro zbývající dvě osy.

- **Buňka** (cell) je podmnožina objemu definovaná osmi okolními voxely 3D mřížky. Má tvar krychle nebo kvádru, v jehož vrcholech jsou umístěny hodnoty voxelů sousedních dvou řezů (čtyři z prvního a čtyři z druhého).

3.2 Zdroj dat

Vzhledem k široké oblasti využití volumetrického renderingu mohou mít data různý význam. Voxely obvykle uchovávají hodnotu nějaké fyzikální veličiny analyzovaného objektu, jako například hustotu, teplotu, tlak apod. Kromě toho můžeme mít přiřazená také vektorová data jako barvu, normály, jednotlivé složky materiálů pro výpočet stínování (difúzní, spekulární, ambientní atd.), koordináty textur, pakliže je máme k dispozici.

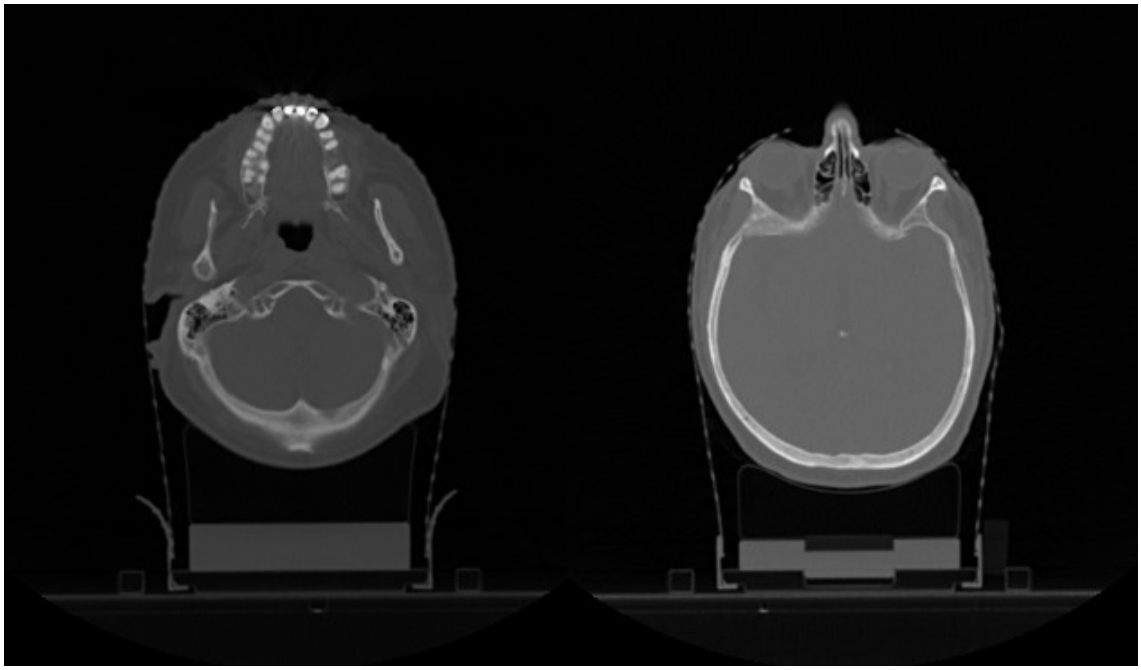
Kromě hodnot vyjadřujících vlastnosti voxelů nás může zajímat také pohled na jejich pozice a zařazení do logických celků. Nejjednodušším případem je bitová maska řezu, která pouze říká, kde objekt je (hodnota 1) a kde je pozadí (hodnota 0). Vhodné je pak mít k prostorovým datům tzv. distanční mapu představenou Gibsonem v článku [3]. Distanční mapa (resp. distanční funkce) je matice, která k odpovídajícím voxelům přiřazuje hodnotu vzdálenosti od povrchu tělesa. Na hranici objektu je 0 (hraniční iso-plocha), s rostoucím zanořením dovnitř se hodnota vzdálenosti zvyšuje. Stejně tak je tomu vně objektu. Aby se dal snadno rozlišit vnitřek od vnějšku, nabývají vesměs voxely uvnitř tělesa záporné hodnoty. Takto tedy můžeme zobrazovat jen voxely s určitou vzdáleností od povrchu, nebo využijeme prahování, tedy zobrazíme tu část objektu, která má vzdálenost vyšší nebo nižší, než je nějaká prahová. Zmíněné datové sety se pak při výpočtu kombinují pro získání co nejvěrohodnějších obrázků. Výhoda znalosti vzdálenosti od povrchu je obecně v možnosti přesnějšího výpočtu normálových vektorů ve voxelech. Normály se nad diskrétními daty obvykle odhadují gradientní metodou (viz kapitola 4.1 a [11, 2]). Gradient v bodě distanční funkce odpovídá vektoru směřujícímu k nejbližšímu povrchu, což je velmi přesný odhad.



Obrázek 2: Vizualizace spojité (vlevo) a diskrétní (vpravo) distanční funkce. Zdroj [6]

Testovací data

Dodaným datovým souborem k této práci jsou PNG obrázky ve stupních šedi o rozlišení 256×256px a 512×512px reprezentující řezy z CT uchovávající v sobě hustoty (Obrázek 3). Nejhustší místa nabývají v obraze bílé barvy, zatímco nejméně hustá mají barvu šedou nebo černou.



Obrázek 3: Ukázka řezů CT

Takovému druhu dat se říká matice hustot, resp. matice intenzit (viz [11] nebo [18]). Rozdílné hustoty vyjadřují různé materiály, případně objekty. Na lékařských snímcích tak můžeme rozeznat kosti od svaloviny nebo tkáně, různé orgány mezi sebou. Spojení voxelů s podobnou hustotou představuje jeden objekt. Datový soubor bohužel neobsahuje žádné rozšiřující informace. Odpovídající matice barev a materiálů tak bude dána přechodovou funkcí (vysvětleno např. v článku [5]), což je funkce přiřazující voxelům vlastnosti na základě jeho hustoty. Pro účely vizualizací diskutovaných později se přechodová funkce využívá také na určení průhlednosti, resp. hodnoty útlumu jednotlivých materiálů (hustot). To nám umožňuje samostatně si stanovit priority jednotlivých částí modelu, aby se neodvíjely pouze od fyzikálních vlastností zkoumaných objektů. Lékařské přístroje taktéž negenerují distanční mapy. Řešením by mohl být samostatný výpočet těchto vzdáleností. Problém je, že data ze své organické či biologické podstaty nemají nějakou standardizovanou strukturu, což by muselo vést k segmentaci CT snímků, jelikož potřebujeme znát distanční funkci pro každý objekt zvlášť, a k poměrně náročným úlohám z oblasti zpracování obrazu. Výpočet distanční mapy nad vysegmentovanými objekty by pak byl pravděpodobně řešen formou rekurze, a to s ohledem na implementaci na GPU není příznivé řešení. Normály se tedy budou muset vypočítat klasickou gradientní metodou řešenou na základě matice hustot. Když vezmeme v úvahu poměrně vysoké

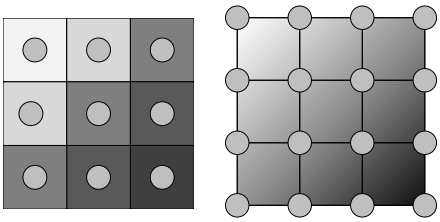
rozlišení obrázků (512×512px), může být i takový odhad naprosto vyhovující. Podrobněji bude výpočet normál rozebrán v kapitole 4.1.

3.3 Varianty mřížky

Na objem se dá nahlížet dvojím způsobem (viz [2]).

Zaprvé ho můžeme brát jednoduše jako 3D mřížku voxelů se stanoveným rozměrem, kdy celý objem voxelu má konstantní hodnotu. Když budeme chtít získat hodnotu vzorku umístěnou někde v objemu, může se často stát, že jeho pozice nebude přímo odpovídat souřadnici voxelu (resp. jeho středu), ale bude ležet mezi dvěma sousedícími voxely. V takovém případě máme dvě možnosti. Buďto použijeme hodnotu nejbližšího voxelu, což není příliš elegantní řešení, neboť výsledný obraz pak trpí schodovými artefakty, anebo hodnotu získáme trilineární interpolací okolních osmi voxelů, čímž zajistíme plynulý přechod mezi voxely.

Zadruhé se pak můžeme na objem dívat jako na strukturu buněk. Tato varianta z principu netrpí výraznými schodovými efekty, neboť objem buňky je opět definován trilineární interpolací. Voxely v tomto případě budeme chápat jako hodnoty ve vrcholech buňky mající nulový rozměr. Buňka má pak velikost danou vzdálenostmi mezi řezy v jednotlivých osách (W_x, W_y, W_z). Celý prostor je tak popsán spojitě po částech. Druhá varianta je daleko intuitivnější než ta první. Buňky jsou logicky opět uspořádány do ekvidistantní mřížky a dají se velmi jednoduše indexovat. Oba případy jsou ilustrovány na následujících obrázcích.



Obrázek 4: Voxel (vlevo), Cell (vpravo)

Rozsah indexů je ve všech osách o 1 menší než je nejvyšší index voxelu v dané ose, neboť každá buňka obsahuje v sobě reference na dva sousední řezy. Řekněme tedy, že objem má rozměry D_x, D_y, D_z voxelů, počty buněk budeme značit:

$$DC_x = D_x - 1,\ DC_y = D_y - 1,\ DC_z = D_z - 1,$$

indexy voxelů pak jsou v intervalech:

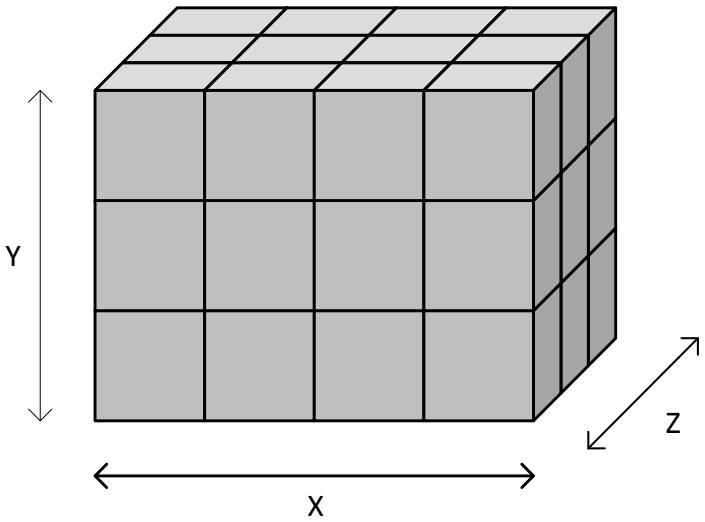
$$x \in \langle 0; D_x - 1 \rangle,\ \ y \in \langle 0; D_y - 1 \rangle,\ \ z \in \langle 0; D_z - 1 \rangle,$$

maximální hodnoty indexů buněk tedy jsou:

$$C_x^{\max} = D_x - 2,\ C_y^{\max} = D_y - 2,\ C_z^{\max} = D_z - 2.$$

Tento přístup se pak později vyplatí při průchodu datovou strukturou a při samotném renderování. Reprezentace buňkami má tu výhodu, že se k ní dá přistupovat univerzálně a výpočty s ní se provádějí jako s kvádrem či krychlí. Není potřeba dohledávat okolní voxely a řešit míru jejich vlivu na aktuální vzorek. Vše je otázkou interpolace hodnot odpovídajících vrcholů buňky.

Kromě ekvidistantní kolmé mřížky se využívají i jiné datové struktury popsané např. v [11]. Existují mřížky, které jsou sice pravoúhlé, ale nemají mezi sebou konstantní krok, jiné, nestrukturované mohou být definované nad neuspořádanými daty. Nestrukturovaná mřížka ve 3D pak může vypadat tak, že jednotlivé buňky nejsou kvádry nebo krychle, ale čtyřstěny. Zde se však vzhledem k charakteru dat získaných z CT můžeme spolehnout na uspořádaná data zarovnaná na pravoúhlou mřížku s konstantním krokem. Kvalita výsledných obrázků pak závisí na vzdálenosti jednotlivých řezů, což se odvíjí od přesnosti přístrojů, které je pořizují, resp. od uživatelsky nastavených rozměrů buňky W_x, W_y, W_z. Pokud například tomograf nedokáže produkovat snímky hustěji než několik jednotek milimetrů, hodnota mezi těmito řezy je neznámá a musí se interpolovat, čímž dochází ke zkreslení. Podobný problém nastává v případě, že snímky mají nízké rozlišení. Takové komplikace se objevují obecně u jakýchkoliv diskrétních dat a řešením je snažit se dosáhnout co nejvyšší vzorkovací frekvence. Proces získávání dat, porovnání jejich přesnosti a reprezentace dat je podrobněji rozepsán např. zde [18].



Obrázek 5: Ilustrace mřížky objemu

4 Volumetrický rendering

Následující kapitola pojednává o možnostech zobrazování volumetrických dat. V dnešní době existuje celá řada algoritmů, které se liší především v tom, co přesně zobrazují. Metody můžeme rozdělit na dvě základní kategorie (viz např. [2]):

- Algoritmy zobrazující povrch
- Algoritmy, které povrch nehledají

První metoda se anglicky nazývá Isosurfacing a jejím úkolem je vyjádřit z objemu hledanou iso-plochu odpovídající dané hodnotě hustoty. S tou se pak dále pracuje. Druhá varianta vesměs odpovídá technice zvané "přímé objemové zobrazování" (Direct volume rendering - DVR), která se snaží zobrazovat data celého objemu na základě průhlednosti jednotlivých materiálů dané přechodovou funkcí. Nejznámějším algoritmem z rodiny DVR je tzv. Ray Casting.

Zadáním diplomové práce je vytvoření netriviálního volumetrického rendereru s průhledností, bylo tedy třeba vybrat algoritmus, který tuto podmínku bude splňovat. V podstatě se dá řešení postavit na obou zmiňovaných metodách, které budou v odstavcích níže rozepsány a vyhodnoceny.

4.1 Isosurfacing

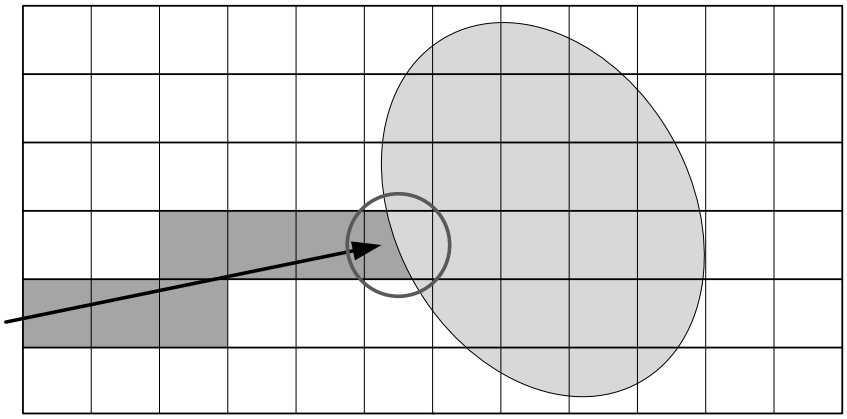
Algoritmy zobrazující pouze iso-plochy odpovídající stanovené hodnotě spadají do skupiny isosurfacing. Těchto ploch může být promítnuto do obrazu více s nějakou mírou průhlednosti, základní myšlenka je však ve vykreslování jedné vrstvy dle zadané hodnoty hustoty objemu, či vzdálenosti od okraje (dle distanční mapy). Isosurfacing se dá pojmout dvojím způsobem. Buď nám jde jen o vykreslení iso-plochy (správná volba přechodové funkce), tudíž není třeba ji přímo izolovat od zbytku dat, nebo chceme iso-plochu nějak dále zpracovávat, tak ji vyjádříme formou trojúhelníkové sítě (např. Marching Cubes).

Nalezení iso-plochy:

Stěžejní úlohou těchto metod je vyhledání všech buněk, kterými iso-plocha prochází. Postup se pak dá využít i pro DVR algoritmy při určování homogenity prostředí. U triangulačních algoritmů, kde chceme popsat povrch sítí trojúhelníků, procházíme všechny buňky objemu a vybereme ty, které obsahují hledanou hodnotu (obvykle hustotu). To se určuje na základě porovnávání hledané hodnoty s hodnotami voxelů ve vrcholech buňky. Mohou nastat 3 případy:

- hodnoty všech vrcholů buňky jsou menší než práh,
- nebo jsou všechny větší než práh,
- nebo jsou některé větší a některé menší

U prvních dvou je jasné, že těmito buňkami iso-plocha neprochází, protože jsou celé vně, nebo vevnitř hledaného tělesa. V opačném případě, kdy některé vrcholy jsou větší a některé menší než prahová hodnota, se jedná o buňku, kterou protíná iso-plocha. Situaci ilustruje Obrázek 6.



Obrázek 6: Hledání iso-plochy

Výpočet normály gradientní metodou:

Pro pozdější stínování je potřeba dohledat normálu iso-plochy v bodě, kde ji paprsek protíná. K tomu je nutné nejdříve zjistit normály jednotlivých voxelů ve vrcholech buňky, pomocí kterých se pak interpolací dobereme k normále plochy. Vzhledem k tomu, že hledáme jakousi hraniční plochu (přechod hustoty) v trojrozměrném prostoru, odpovídá orientace normály směru gradientu v bodě (směr největší změny). Jelikož neznáme spojitou funkci hustoty v prostoru, vycházíme z diskrétní reprezentace $\rho(x, y, z)$, kde derivace spojité funkce aproximujeme symetrickými diferencemi hodnot v prostorové mřížce (viz [11, 2]).

Výpočty souřadnic gradientu $\nabla(\rho) = (x_g, y_g, z_g)$ pak jsou:

$$x_g = \rho(x+1, y, z) - \rho(x-1, y, z),$$
$$y_g = \rho(x, y+1, z) - \rho(x, y-1, z),$$
$$z_g = \rho(x, y, z+1) - \rho(x, y, z-1).$$

Alternativami jsou 3D masky, kterými se provádí konvoluce v prostoru s diskrétními daty, čímž můžeme vzít v potaz větší počet voxelů nebo určit různé míry vlivu okolních voxelů na výslednou diferenci. V článku [24] je srovnání gradientních operátorů (Prewitt, Sobel, Scharr) testovaných na diskrétním obraze.

4.1.1 Marching Cubes, Marching Tetrahedrons

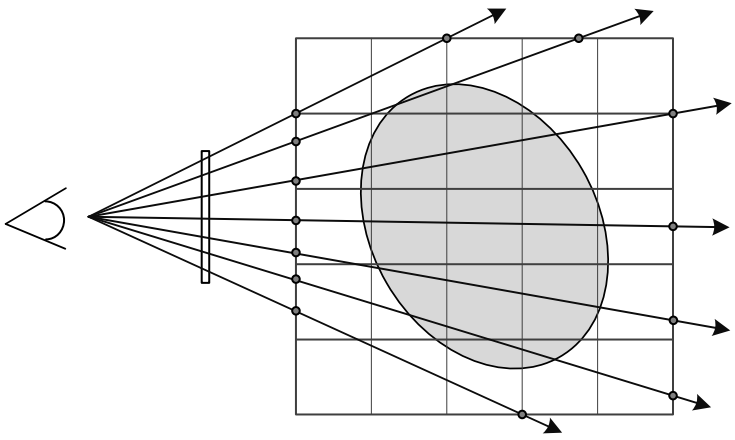
Klasickým algoritmem používaným k nalezení iso-ploch je zmiňovaný Marching Cubes představený Lorensenem a Clinem (čl. [19]), který se snaží popsat iso-plochu sítí trojúhelníků, jenž mají vrcholy umístěné na hranách volumetrických buněk. Jedná se o jednoduchou metodu triangulace volumetrických dat, která je výhodná především v případech, kdy potřebujeme z datasetu vytvořit povrchovou reprezentaci jeho určité části, a tu nějak dále zpracovávat. Algoritmus se obvykle implementuje metodou "rozděl a panuj" (zpětné spojování trojúhelníků do větších celků sítě). Postupem popsaným dříve se dohledají všechny zasažené buňky. Metoda využívá znalost celkem 256 konfigurací krychle protnuté trojúhelníky, ze kterých se postupně skládá výsledná síť. Většina konfigurací se však dá získat jednoduše rotací nebo symetrií základních 15 konfigurací. Rozšířením této metody je pak algoritmus Marching Tetrahedrons, která zamezuje vzniku děr v trojúhelníkové síti, což je největší slabina klasického Marching Cubes. Algoritmus spočívá v dělení krychle na menší čtyřstěny, které vyplňují celý prostor krychle. Teprve v těch se pak hledají finální trojúhelníky.

Výsledkem obou metod je síť trojúhelníků, která se pak dodatečně vizualizuje pomocí Ray Tracingu, nebo např. OpenGL či DirectX. Trojúhelníkové povrchy reprezentují jednotlivé hladiny objemu, které se pak dají zobrazovat přes sebe s určitou barvou a průhledností. Výsledný efekt však asi nebude odpovídat tomu, co by mělo být cílem této práce. Trojúhelníky budou v paměti zabírat hodně místa a práce s nimi je neobratná. Když si představíme, že budeme chtít zobrazit třeba 20 vrstev objemu, znamená to, že musíme vypočítat triangulaci 20ti iso-ploch, kde každá konfigurace může obsahovat až 4 trojúhelníky, tj. 12 bodů. Asi nebude problém, aby se při zadaném datovém setu iso-plocha skládala řádově i ze statisíců drobných trojúhelníčků. V paměti to pak může zabrat i stovky megabytů, na což se zvláště při implementaci na GPU musí brát ohled. Existují i novější vylepšené verze triangulačních algoritmů. Například topologické problémy s dírami a neoptimálními trojúhelníky řeší metoda Regularised Marching Tetrahedra [21], kde autoři rozšířili Marching Tetrahedrons o dodatečné optimalizace trojúhelníkové sítě metodou Vertex clustering, která spojuje malé trojúhelníky do větších celků (např. článek [22]). Další technikou je Stretching Cubes publikovaná v článku [10]. Ta se zaměřuje na jemnější prahování při hledání iso-plochy, aby bylo možné vizualizovat i méně výrazné orgány, jako jsou plíce, které jsou na řezech vzhledem k malé hustotě špatně rozpoznatelné. Ani tyto metody však neřeší všechny problémy, jsou rozsáhlé a výpočetně náročné a pro rychlé vykreslování, nebo dokonce real-time rendering se nehodí.

4.2 Direct Volume Rendering (DVR)

Algoritmy z rodiny DVR přistupují k vykreslování z pohledu globálních zobrazovacích metod. Nejčastěji používaným takovým algoritmem je Ray Tracing (RT) popsaný například v [7, 8, 23] vynalezený Turnerem Whitedem roku 1979. Jedná se tedy o metodu přímého zobrazování toho, co reálně vidíme okem kamery, která se snaží o simulaci fyzikálních vlastností a chování světla. Ray Tracing se také nazývá metodou "zpětného sledování paprsku", neboť sledujeme jeho trasu ve směru od kamery skrz scénu po světelný zdroj. Takový přístup se pro rychlejší a realistickou vizualizaci objemu hodí lépe než metody generující povrch, neboť nemusíme počítat to, co na výsledný obraz nemá vůbec vliv. Kromě toho každý paprsek můžeme chápat jako jeden nezávislý proces. Podél paprsku se akumuluje barva daná materiály objektů ve scéně a barvou pozadí, proces si tedy udržuje v paměti hodnotu z minulého kroku trasování, ke které nějakým způsobem přičítá hodnoty aktuálních vzorků. Z toho vyplývá, že není nutné provádět preprocessing triangulace a uchovávat jednotlivé iso-plochy v paměti. Stačí nám pamatovat si hodnotu v akumulátoru a hodnoty lokálních proměnných souvisejících s výpočtem aktuálního kroku. Ray Tracing se obecně používá pro vykreslování téměř jakýchkoliv grafických dat, vždy se jedná o to, co přesně chceme zobrazovat. Vedle volumetrické struktury jsou to také trojúhelníkové sítě, částicové systémy apod. Při vykreslování objemových dat však obvykle nepotřebujeme přímo kompletní Ray Tracing. Místo něj se používá tzv. Ray Casting (RC, metoda "vrhání paprsků"), což je přímá metoda založená na sledování paprsku prostředím. V tomto případě se však nepočítá odraz a lom, jako tomu je u RT. Neboť známe objem celého objektu a zajímá nás jeho zobrazení s průhledností, do akumulátoru se přičítají hodnoty podél paprsku napříč celým objemem, tedy ne hodnoty získané odraženým či lomeným paprskem. Reflexivní efekt v případě medicínských snímků by pravděpodobně byl spíše na škodu.

V kapitole 4.2.1 jsou popsány triviální metody přímého volumetrického zobrazování, které jsou základem pro podrobněji rozepsané Ray Castingové metody v kapitole 4.2.2.



Obrázek 7: Průchod paprsků objemem

4.2.1 Triviální metody DVR

Triviální metody přímého renderování (popsané např. zde [2]) představují základní modely vizualizace volumetrických dat, které sice nevypadají realisticky a mají nižší vypovídací hodnotu, ale jsou rychlé a jejich implementace jednoduchá. Výsledky je nutné normalizovat do rozsahu hodnot výstupního obrazu (0-1, resp. 0-255).

- **Maximum-intensity projection -** Metoda založená na vykreslování maximální intensity. Paprsek prochází volumetrickou strukturou a do obrazu se promítá pouze hodnota, které je přiřazena nejvyšší intenzita (priorita) ze všech průchozích buněk. Tato metoda je velmi rychlá a používá se při renderování animací (např. rotace) tělesa, neboť aplikací MIP ztrácí obraz informace o prostoru, tudíž bez záběrů z různých úhlů není možné se v obraze dobře zorientovat. Obrázky pořízené stejnými paprsky sledovanými z opačných směrů jsou vzájemně symetrické. Do hloubky je tento přístup rozepsán v článcích [11, 16, 17, 18].
- **Součtová metoda** - výsledný pixel obrazu je tvořen součtem jasů vzorků podél paprsku
- **Průměrová metoda** - výsledný pixel obrazu je tvořen průměrem jasů vzorků podél paprsku
- **Povrchová metoda** - Přímá metoda zobrazující iso-plochu objemu (viz [11]). Na rozdíl od Marching Cubes tato metoda neprování triangulaci povrchu, ale hledá podél paprsku nejbližší buňku obsahující hledanou hustotu (intenzitu). Po nalezení požadované buňky se dosadí rovnice paprsku (resp. přímky) do rovnice objemu, který je popsán trilineární interpolací diskrétních hodnot na uniformní mřížce. Získáme tedy:

$$\rho(x_a + t \cdot x_b, y_a + t \cdot y_b, z_a + t \cdot z_b) - \rho_{\text{iso}} = 0,$$

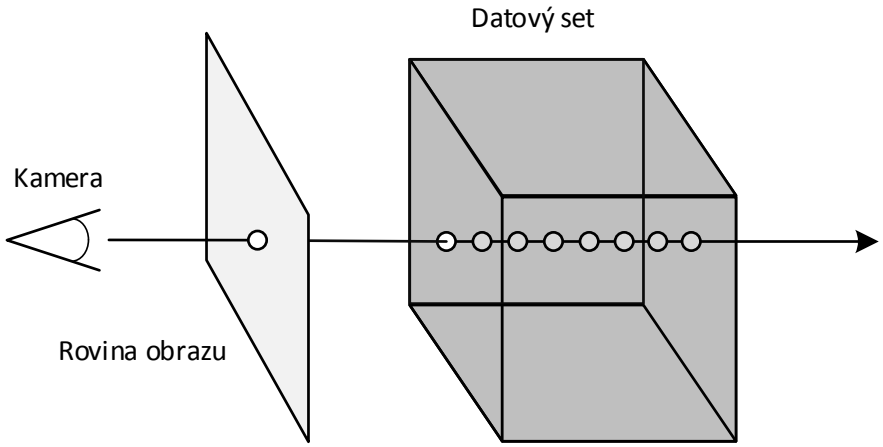
  kde ρ je funkce definující objem a ρ_{iso} je hodnota hledané hustoty. Složky x_a, y_a, z_a jsou jednotlivé souřadnice počátečního bodu přímky, x_b, y_b, z_b pak jsou složky jejího směrového vektoru. Pokud má rovnice řešení, minimalizací tohoto rozdílu získáme nejpřesnější odhad výskytu iso-plochy. Je zapotřebí si ale pohlídat, že se vypočtený průsečík s iso-plochou nachází uvnitř dané buňky, průsečíky paprsku s kvádrem buňky nám tedy zde určují interval parametrů, které nás zajímají. Z rovnice pak dostaneme parametr t, jehož dosazením do rovnice přímky získáme konkrétní bod iso-plochy.

Všechny popsané algoritmy mohou být rozšířeny o stínování na základě normály v bodě např. Phongovým stínováním ([13, 15]). Předpokládáme, že normály ve vrcholech mřížky máme vypočteny gradientní metodou, normálu v bodě uvnitř buňky pak dopočítáme trilineární interpolací vrcholových hodnot, tedy dostaneme vektorovou rovnici, která se bude řešit pro každou složku normálového vektoru zvlášť. Rozšířením součtové nebo průměrové metody získáme Volumetrický Ray Casting diskutovaný v následující kapitole.

4.2.2 Volumetrický Ray Casting (VRC)

Metoda využívá základního předpokladu, že částem modelu lišícím se svou hustotou jsou přiřazeny různé hodnoty průhlednosti a barev (tzv. přechodová funkce). Vykreslování volumetrických dat pak není omezeno jen na konkrétní iso-plochy nebo intenzity, ale bere v potaz všechny buňky „zasažené“ průchodem paprsku od kamery do scény (viz např. [5, 16, 17, 2]).

Původní myšlenka Ray Castingu vynalezeného Arthurem Appelem v roce 1968 pochází ještě z dob před rozvojem Ray Tracingu. Nevolumetrický RC spočíval ve vyslání paprsků z kamery obrazem (jeden pro každý pixel) a hledání nejbližšího zasaženého objektu. Další trasování odražených a lomených paprsků bylo aplikováno až s vývojem technologií RT. Volumetrický Ray Casting využívá podobného postupu s tím rozdílem, že nehledá pouze první zasažený voxel (resp. buňku), ale prochází celým objektem napříč ve směru vyslaného paprsku (Obrázek 8).



Obrázek 8: Volumetrický Ray Casting

Ray Casting naproti iso-surfacingu dokáže zachytit větší míru detailů v obraze, jemnější přechody a materiály, které mají být poloprůhledné, jsou také takto zobrazeny. Stejná data mohou být zobrazena různým způsobem s ohledem na nastavení průhledností jednotlivým hustotám.

V podkapitole 4.2.2.1 je rozebrán teoretický model Blinn/Kajiya, ze kterého vychází algoritmy Levoye a Drebina popsané v podkapitole 4.2.2.2 pro diskrétní prostor.

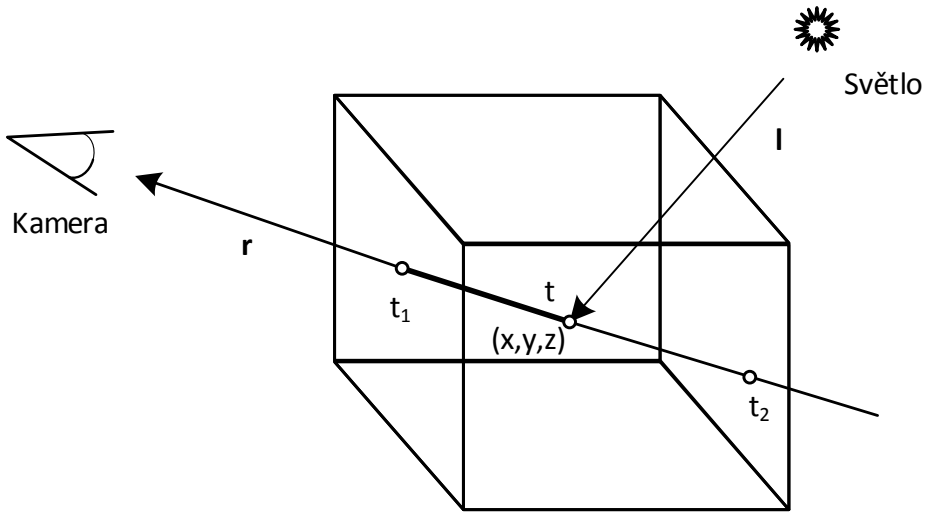
4.2.2.1 Blinn/Kajiya model

Dnešní techniky VRC staví na Blinn/Kajiya modelu [5]. Mějme spojitý volumetrický prostor, kterým prochází paprsek do oka kamery (v tomto případě se uvažuje přímé sledování paprsku).

Model je popsán funkcí hustoty:

$$\rho(\mathbf{r}(t)) = \rho(x_a + t \cdot x_b, y_a + t \cdot y_b, z_a + t \cdot z_b),$$

kde $\rho(\mathbf{r}(t))$ vyjadřuje hustotu v bodě přímky paprsku závislé na parametru t (v textu ji budeme zkráceně označovat $\rho(t)$), složky x_a, y_a, z_a jsou jednotlivé souřadnice počátečního bodu přímky, x_b, y_b, z_b pak složky směrového vektoru. Dále je definován funkcí osvětlení $I(\mathbf{r}(t))$ (označovaná jako $I(t)$), která představuje osvětlení ve zkoumaném bodě daném parametrem t, a fázovou funkcí P vyjadřující vliv úhlu mezi paprskem a spojnicí zkoumaného bodu se světlem na šíření světla objemem. Ilustrace na obrázku Obrázek 9.



Obrázek 9: Šíření světla prostorem

Hodnota intenzity světla vyzářeného ze zkoumaného bodu daného souřadnicemi (x, y, z) získanými pro parametr t z rovnice paprsku je:

$$C(t) = \rho(t) \cdot I(t) \cdot P(\cos\varphi), \qquad (1)$$

kde φ je zmiňovaný úhel mezi **r** a **l** (**r** je směrový vektor paprsku a **l** je směrový vektor od světelného zdroje k aktuálnímu bodu, oba normalizované).

Na obrázku vidíme, že paprsek protíná prostor v bodech označených jako t_1 a t_2, tzn. tyto body odpovídají hodnotám získaným z rovnice přímky (paprsku) po zadání parametru $t = t_1$, resp. $t = t_2$. Abychom dostali celkový součet světla, které prošlo skrz volumetrický prostor podél paprsku, musíme provést integraci rovnice šíření světla v mezích t_1 až t_2. Vhledem k tomu, že míra osvětlení nemusí záležet pouze na úhlu $\mathbf{r}$ a $\mathbf{l}$, ale také na útlumu světla v prostředí, pakliže není osvětlení konstantní, vypočteme hodnotu útlumu způsobeného průchodem objemu od t_1 po t takto:

$$A(t, t_1) = \exp\left(-\tau \int_{t_1}^{t} \rho(s)\mathrm{d}s \right), \tag{2}$$

kde τ je konstanta převádějící hodnotu hustoty na útlum.

Výslednou rovnicí šíření světla celou oblastí podél paprsku je tedy:

$$B(t_1, t_2) = \int_{t_1}^{t_2} \left(\exp\left(-\tau \int_{t_1}^{t} \rho(s)\mathrm{d}s \right) \right) \left(I(t)\rho(t)P(\cos\varphi) \right) \mathrm{d}t, \tag{3}$$

$B(t_1, t_2)$ je pak hodnota pixelu obrazu, kterým prochází paprsek z kamery.

Popsaný postup neodpovídá zcela představě nastíněné v úvodu (princip hustoty a průhlednosti). Model Blinn/Kajiya popisuje celou úlohu jako fyzikální simulaci průchodu světla objemem (např. kouřem nebo mrakem) a místo přiřazené průhlednosti zde veškerou práci zastane spojitá funkce hustoty, která definuje průchodnost jednotlivými částmi objektu (vyšší hustoty propouští skrz méně světla, resp. mají na výsledek největší vliv) a taktéž útlum prostředí, čímž se nahrazuje akumulační efekt sčítání barev násobených průhledností. Výsledek zde také neodpovídá vektoru barvy, nýbrž skaláru jasu. Rozepsaný model tedy je základní matematickou teorií volumetrického Ray Castingu a staví na něm většina ostatních přístupů (teorie i obrázky vychází z článku [5]).

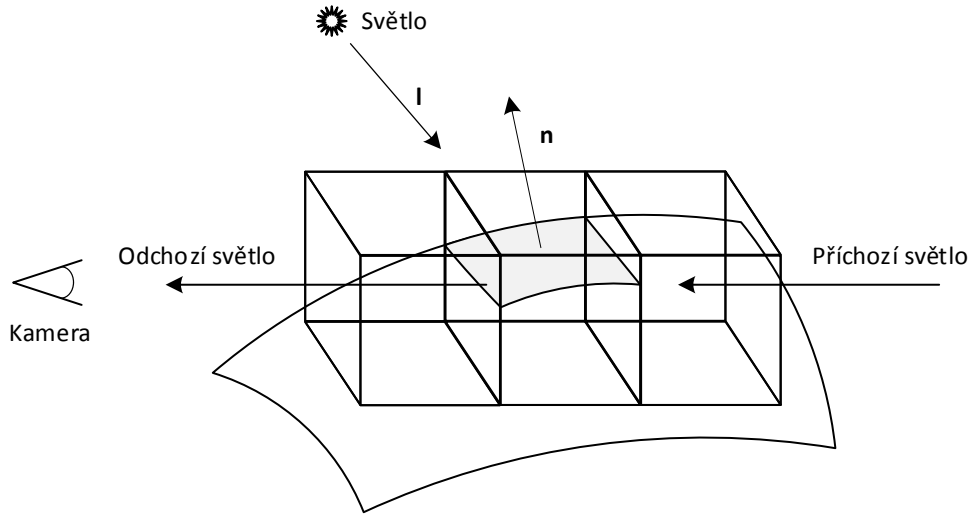
4.2.2.2 Algoritmus VRC v diskrétním prostoru

Speciálními případy aplikace Blinn/Kajiya modelu šíření světla objemem pro diskrétní prostor jsou dva velmi podobné algoritmy. Jsou jimi Levoyova a Drebinova integrační metoda. Z těchto postupů vychází praktická část diplomové práce.

Levoyův algoritmus

Algoritmu se také říká Aditivní reprojekce (popsáno v článku [5]).

Aditivní reprojekce popisuje výstupní hodnotu vrženého paprsku jako kombinaci ozařujícího světla (pocházejícího ze zdroje) v daném bodě a příchozího světla procházejícího mřížkou voxelů (Obrázek 10).



Obrázek 10: Průchod světla mřížkou

Metoda, kterou popsal Marc Levoy (článek [20]), se dělí na dvě dílčí úlohy:

- vizualizační část
- klasifikační část

Vizualizační část řeší výpočet stínování v jednotlivých voxelech, klasifikační pak přiřazení neprůhledností (alfa kanálu) jednotlivým voxelům. Pro každý pixel obrazu tedy vysíláme jeden paprsek voxelovou mřížkou. Podél paprsku se pro každý protnutý voxel počítá jeho barva a neprůhlednost. Výsledné hodnoty voxelů se pak sčítají mezi sebou, a k nim se ještě přičte barva pozadí scény.

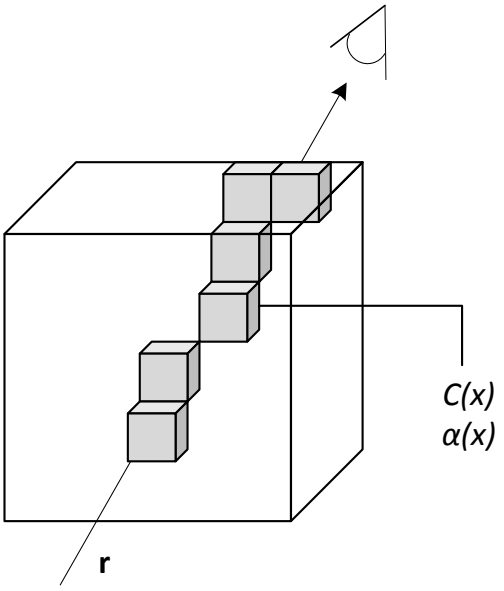
V prvním kroku tedy počítáme pro každý zasažený voxel jeho normálu pomocí gradientní metody popsané v kapitole 4.1. Voxelům je pak přiřazena určitá barva na základě jejich hustoty. S využitím těchto dvou hodnot můžeme provést stínování voxelu (např. Phongovým stínováním). Pro přesnější hodnoty použijeme trilineární interpolaci okolních voxelů, v případě

hranové reprezentace bude výsledná hodnota určená interpolací vrcholů protnuté buňky. Takto tedy získáme výsledky (barvy, resp. intenzity) vizualizačního kroku pro každý voxel (buňku) podél paprsku.

Klasifikační krok pak spočívá ve stanovení funkce, která bude jednotlivým voxelům přiřazovat míru neprůhlednosti. To závisí čistě na požadavcích řešení. Funkce se může odvíjet od funkce hustoty nebo intenzit z předešlého kroku. Abychom zohlednili fakt, že paprsek může každým voxelem procházet jinak, bere se v potaz délka úsečky, kterou paprsek voxel protíná. Klasifikace by také měla řešit problém stínování homogenních oblastí v mřížce. Pakliže má více sousedních voxelů stejnou hodnotu hustoty (barvy), je nežádoucí aby se stínovaly i voxely vevnitř takové oblasti (potřebujeme vystínovat jen hraniční iso-plochu). Voxelům, které se nemají stínovat, klasifikační krok přiřadí průhlednost 1 (resp. alfa kanál 0). To, jestli je voxel uvnitř homogenní oblasti, se pozná tak, že jeho gradient je nulový. Obecně se situace řeší tak, že se neprůhlednost $\alpha(x)$ vynásobí velikostí gradientu voxelu, výsledná hodnota neprůhlednosti tedy je: $\alpha'(x) = |\nabla\rho(x)| \cdot \alpha(x)$.

Pro každý voxel nyní známe dvě hodnoty:

- $C(x)$ – intenzita získaná stínováním (barva)
- $\alpha(x)$ – neprůhlednost (alfa kanál)



Obrázek 11: Průchod objemem

Na uvedeném obrázku (Obrázek 11) je znázorněn algoritmus postupného načítání barev podél paprsku, který je popsán následující rekurentní rovnicí:

$$C_{i+1} = C_i\left(1-\alpha(x_i)\right)+c(x_i)\alpha(x_i),\ C_0 = \text{pozadí} \qquad (4)$$

C_{i+1} je výstupní intenzita naakumulovaná podél paprsku včetně i-tého voxelu. C_i je pak vstupní intenzita akumulovaná podél paprsku v předešlých krocích. Funkce $c(x_i)$ je intenzita aktuálního voxelu a $\alpha(x_i)$ jemu přiřazená neprůhlednost. Jakmile se takto projdou veškeré voxely protnuté paprskem, získáme výslednou intenzitu (barvu) pixelu obrazu. Celý proces se opakuje pro všechny pixely obrazu.

Drebinův algoritmus

Algoritmus popsaný např. v knize [2] vynalezený R. A. Drebinem pracuje také dvoufázově. V prvním kroku se připraví pole hodnot, ze kterých se pak v 2. kroku vypočítává výsledná barva. Těmito poli jsou:

- procentní zastoupení materiálů
- barvy
- hustoty
- velikosti gradientu

Metoda vychází z toho, že voxel může nabývat více barev, resp. materiálů. Pole procentního zastoupení pak vyjadřuje, jak velký vliv který materiál na něj má. Neprůhlednost je zde zahrnuta v poli barev. Vektory barev jsou vyjádřeny v projektivním prostoru, kde vahou homogenní souřadnice je právě alfa kanál. Zbylý postup je téměř shodný s Levoyovou metodou. Dá se říct, že každá z uvedených metod se dá jednoduše upravit na druhou. Levoyova metoda vycházela hlavně z voxelové reprezentace a jejich poměrového zastoupení na výsledku, Drebinova metoda se blíží myšlenkově spíše reprezentaci buňkami, které se zde simulují procentním zastoupením materiálů.

5 Implementace

Zadáním diplomové práce bylo naprogramovat netriviální volumetrický renderer využívající k výpočtům GPU. Úkolem bylo nastudovat článek [1], ze kterého se mělo vycházet. Článek popisuje volumetrický Monte Carlo Ray Tracing algoritmus, který využívá stochastické postupy pro stínování a trasování paprsků objemem. Z toho tedy vyplývá, že výsledný program je volumetrický Ray Casting. Náš postup vychází z Levoyova integračního algoritmu diskutovaného v předcházející kapitole a využívá hranovou reprezentaci dat pomocí buněk. Průchod objemu se děje pomocí stochastického vzorkování podél paprsku technikou zvanou Woodcock tracking [35, 36]. Osvětlovací model se pak odvíjí od Phongova modelu (viz např. [15]) rozšířeného o stochastické vzorkování světel, což vede k realističtějšímu vzhledu a nahrazení ambientní složky sférickým zářičem.

Tato kapitola popisuje průběh implementace. Podkapitola 5.1 řeší dílčí úlohy spojené s vytvořením přímého volumetrického rendereru a teoretický popis použitých algoritmů. Druhá část 5.2 je pak o implementaci pomocí CUDA na GPU.

5.1 Realizace

Cílem práce bylo vyzkoušení určitého postupu, který bude realizovat přímý volumetrický rendering. Nejprve tedy bylo nutné nastudovat metody řešící jednotlivé fáze vykreslování a některé z nich vybrat. Kroky vedoucí napříč implementací výsledné aplikace jsou shrnuty do následujících bodů:

1) Načtení dat a vytvoření datové reprezentace
2) Vytvoření kamery
3) Trasování paprsku objemem a dohledání protnutých buněk
4) Přechodová funkce
5) Výpočet stínování a osvětlovací model
6) Integrace hodnot podél paprsku

Následující kapitoly hovoří o metodách testovaných v průběhu práce. Obsahují popis a hodnocení jednotlivých algoritmů a zdůvodnění, proč byly či nebyly výsledně vybrány. Závěrem je sestaveno shrnutí celého rendereru a jeho pipeline včetně doprovodných obrázků.

5.1.1 Načtení dat a vytvoření datové reprezentace

Jak již bylo několikrát zmíněno, datová reprezentace vychází z hranového modelu popsaného v 3. kapitole. Voxely tedy představují hodnoty ve vrcholech mřížky o nulovém rozměru a celý objem je popsán buňkami tvořenými osmi sousedními voxely. Taková datová struktura vyjadřuje prostor spojitě po částech, tedy není třeba řešit příspěvky jednotlivých voxelů na výsledek. Vše se vyřeší integrací trilineární interpolace. Datový set je zadán sérií řezů v podobě PNG obrázků. V aplikaci je pro načítání obrázků použita OpenSource C++ knihovna FreeImage [25]. Knihovna adresuje řádky obrazu odspodu nahoru, tudíž je bylo nutné číst

v opačném pořadí. Řezy se načítají do 3D statického pole v podobě floatů. Každý voxel je zde reprezentován jedním desetinným číslem vyjadřujícím hustotu materiálu v normovaném tvaru (od 0 do 1). V průběhu načítání dat se také provádí jejich analýza. Řezy obvykle mají čtvercovou velikost (např. 512×512px). Sledovaný objem však může například zabírat pouze čtvrtinu tohoto prostoru. Je tedy nežádoucí, aby se trasovaly rozsáhlé nulové prostory nabývající pouze černé barvy. Výsledkem této analýzy je tedy stanovení obalového boxu, který vytyčuje nejmenší prostor, ve kterém se nacházejí všechny buňky, jež mají nenulovou viditelnost. Dále je také třeba pro budoucí stochastické trasování zjistit maximální hodnotu útlumu (resp. neprůhlednosti) v datovém setu a průměrnou i maximální velikost gradientu v datasetu. I toto je výsledkem datové analýzy. Pro opakované testování nad stejnými daty je pak možné zadat si tyto hodnoty do aplikace ručně a jejich výpočet zakázat, čímž se ušetří čas.

Rozměry mřížky v tomto kroku není nutné řešit, neboť s těmi se počítá až při samotném trasování. Aplikace nepotřebuje mít polohy jednotlivých voxelů předpočítané, stejně tak není vhodné pamatovat si jejich gradienty, protože by to vedlo k dramatickému zvýšení nároků na paměť. Pro urychlení načítání obrázků byla využita paralelizace cyklů pomocí OpenMP.

5.1.2 Kamera

Základem každého Ray Tracingu (resp. Ray Castingu) je kamera, která vysílá paprsky výsledným obrazem do scény. Kamerou chápeme postup vedoucí k vygenerování paprsků, jejichž výsledná naakumulovaná hodnota utváří barvu pixelu. Tímto mechanismem dojde k projektivní transformaci bodů ve scéně na bod 2D obrazu. Výsledný obraz je umístěn v rovině kolmé na směrový vektor kamery. Kamera v aplikaci vychází z komůrkového (pinhole) modelu (popsaného např. v [29]) realizujícího středové promítání.

Vstupními hodnotami kamery jsou:

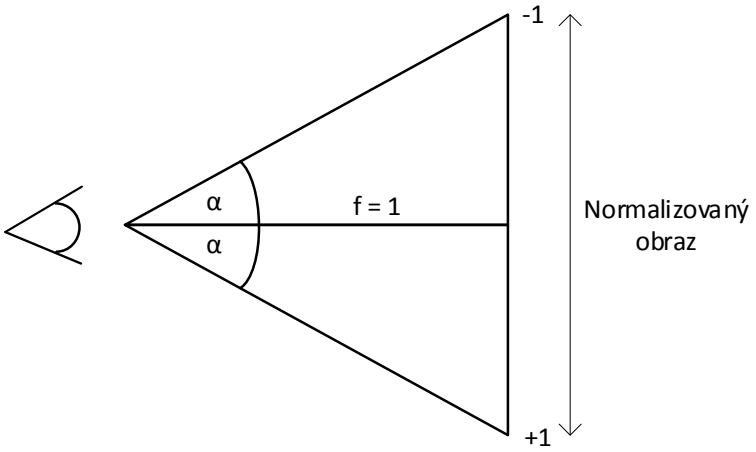
1. poloha kamery (střed promítání)
2. směrový vektor kamery
3. ohnisková vzdálenost f
4. souřadný systém obrazu (vektory u a v, definující osy)

Souřadný systém obrazu je umístěn do jeho středu. Stanovením rozsahu jeho hodnot definujeme zorný jehlan kamery. Obrazový systém se obecně definuje jako pravidelná kolmá mřížka odpovídající pixelům finálního obrázku. Krok v mřížce je určen konstantním krokem, nebo může být vyjádřen úhlem, který svírají roviny zorného jehlanu ve vodorovné a svislé ose kamery.

Implementovaná kamera má střed obrazové soustavy přesně v polovině výšky a šířky výsledného obrazu. Vzdálenost pixelů v mřížce se počítá na základě zadaného úhlu alfa, který zde představuje polovinu horizontálního zorného úhlu. Model pracuje s normalizovanými vzdálenostmi (viz Obrázek 12). Ohnisková vzdálenost je zde 1 a poloha bodu v souřadném systému obrazu je vyjádřena jako desetinné číslo od -1 do 1. Popsaný postup je realizován jednou funkcí, která pak na základě tangens úhlu vypočítá vzdálenost v mřížce. Vertikální úhel

se zde vypočítá z horizontálního na základě poměru výšky a šířky renderovaného obrázku. Funkce vrací normalizované souřadnice bodů v obrazovém prostoru, jejichž přenásobením bázovými funkcemi obrazového prostoru získáme odpovídající polohové vektory tzv. světového prostoru. Dále se provede rozdíl mezi takto získaným bodem a polohovým bodem kamery, čímž získáme směr paprsku vycházejícího z kamery. Výpočet bázových funkcí je popsán např. v knihách [2, 23]. Výhodou takto vyjádřené kamery je možnost jednoduché transformace bázové matice. S kamerou se pak dá nezávisle rotovat kolem osy, měnit její polohu nebo zoomovat. V práci je bázová matice vyjádřena jako rotace bází vzhledem k nárysu. Vstupními parametry kamery tedy jsou ještě tři úhly, pro každou osu jeden.

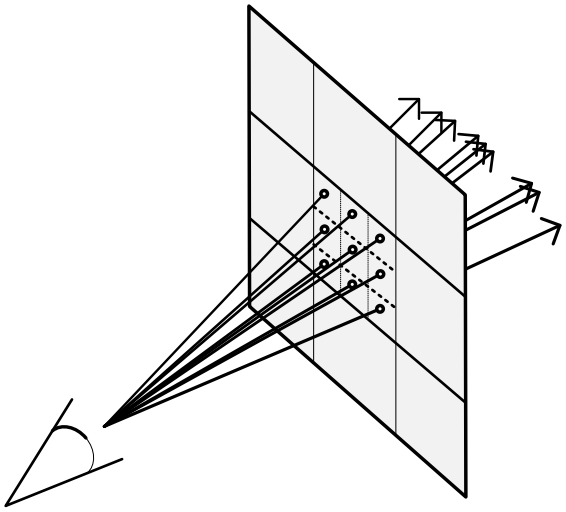
Blízká a vzdálená ořezová rovina je zde částečně nahrazena obalovým boxem objemu, který je vždy konečný, a pak nastavením počáteční hodnoty parametru *t* paprsku, což se později využívá k realizaci uživatelských řezů.



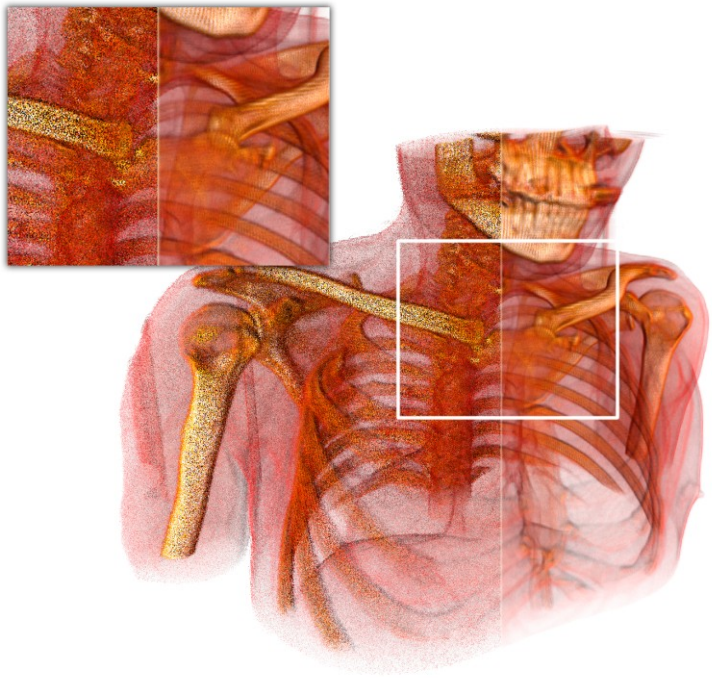
Obrázek 12: Ilustrace kamery

Supersampling

Popisovaný renderer je stochastický DVR, to znamená, že v důsledku aplikace různých Monte Carlo technik dojde k zašumění obrazu. Klasickým problémem je pak aliasing vedoucí k „zubatým“ hranám a falešným obrazcům. Z těchto důvodů je zde implementovaný standardní algoritmus zvaný Super sampling (viz např. [2, 34]). Super sampling vysílá obrazem několikanásobně vyšší počet paprsků. Prakticky je každý pixel obrazu rozdělen na nějakou jemnější mřížku a každým subpixelem této mřížky se vyšle jeden paprsek, jak je ukázáno na obrázku Obrázek 13. Zprůměrováním získaných hodnot nasbíraných paprsky mřížky jednoho pixelu dojde ke zjemnění obrazu a vyhlazení šumu (Obrázek 14). U volumetrických dat se vzhledem k interpolování diskrétních hodnot v obraze projevuje více či méně výrazné pruhování. Zvýšením Super samplingu dojde k protnutí většího počtu buněk, čímž se ve výsledku tento negativní efekt redukuje. Hodnota supersamplingu se v aplikaci nastavuje jako počet subpixelů jedné strany mřížky. Výsledný počet subpixelů je tedy kvadrát této hodnoty.



Obrázek 13: Super sampling 3×3



Obrázek 14: Levá část obrázku je vykreslená bez SS, na pravou je pak aplikovaný SS 8×8

5.1.3 Trasování paprsku objemem a dohledání protnutých buněk

V kapitole o zobrazovacích metodách jsme shrnuli základní přístupy k renderování volumetrických dat. Komentovaná Levoyova technika byla prováděna ve 2 fázích: vizualizační a klasifikační. Předpokladem pro obě byl seznam protnutých voxelů (buněk), ke kterým se

dopočítávaly normály, stínování, neprůhlednost atp. Tato kapitola se tedy zabývá „nultým“ krokem Levoyova integračního algoritmu, jehož výsledkem by měly být hodnoty nasbírané podél paprsku, které se ve finále použijí pro výpočet akumulované barvy. Procházení objemu je z hlediska složitosti nejnáročnější operací celého rendereru. Jednak každý přístup do paměti stojí nějaký čas (obzvlášť na GPU), ale především nám jde o to, abychom zbytečně neprocházeli oblasti, které paprsek neprotíná. Dalším problémem pak jsou rozsáhlé prázdné prostory, které na výsledek nemají prakticky žádný vliv, proto je dobré je z výpočtu vynechat. V teoretickém úvodu bylo zmíněno, že objem můžeme procházet buďto vyhledáním všech protnutých buněk a jejich integrací získat výslednou barvu, nebo hodnoty získat vzorkováním parametru paprsku a akumulací těchto vzorků nahradit integraci buněk. Oba tyto přístupy jsme vyzkoušeli a došli k několika variantám řešení, ze kterých jsme vybrali jednu konečnou.

Podkapitoly jsou rozděleny do tří částí, první komentuje související výpočty, které jsou zapotřebí ve zbylých dvou podkapitolách, kde je rozepsáno několik zvažovaných způsobů traverzace.

5.1.3.1 Související algoritmy

Během trasování paprsků objemem se řeší 2 základní úlohy. Zaprvé je to výpočet průsečíků přímky a kvádru, což se používá pro zjištění, zda paprsek protíná buňku. Zadruhé je to zjištění indexu buňky, ve které se nachází aktuální vzorek. To je potřeba řešit u metody vzorkování podél paprsku. Ještě předtím v první podkapitole je uveden výpočet obalového boxu objemu.

Výpočet obalového kvádru

Z první fáze výpočtu (analýzy dat v průběhu načtení) známe indexy buněk definující minimální a maximální hranici obalového boxu. Na začátku traverzace se vždy počítají průsečíky vrženého paprsku právě s tímto boxem, aby se ověřilo, že paprsek opravdu objem protíná, což znamená značnou úsporu času a dokonce nutný krok, který zamezuje přístupu mimo alokovanou paměť. Vzhledem k tomu, že indexy buněk jsou číslované od nuly, je nutné dataset dodatečně vycentrovat, aby se celá scéna nacházela uprostřed souřadnicového systému. Výsledný obalový box pak bude dán:

$$\begin{aligned} \mathrm{boxMinFinal}_i &= (\mathrm{boxMin}_i - D_i/2)\cdot W_i \\ \mathrm{boxMaxFinal}_i &= (\mathrm{boxMax}_i - D_i/2)\cdot W_i \ \text{ pro } i \in \{x, y, z\}, \end{aligned}$$

kde boxMin_i a boxMax_i jsou indexy buněk obalového boxu, D_i je rozměr objemu ve voxelech a W_i vzdálenost dvou voxelů v mřížce, tedy rozměr buňky.

Obalový box je velmi důležitým prvkem finálního DVR. Kromě urychlení výpočtu je výhodou především adresace buněk, popř. voxelů. Vzhledem k velikosti objemových dat by bylo nevhodné vypočítávat polohu každého voxelu ve scéně nebo si ji dokonce ukládat do paměti. Průsečíky s obalovým boxem vyjadřují interval parametru, v němž se paprsek sleduje. Na

základě znalosti konkrétního bodu na přímce paprsku dokážeme dohledat index související buňky bez výpočtu její reálné polohy v prostoru. Veškerý výpočet se překlopí na práci s jedním bodem a jeho lokalizaci v mřížce. Když chceme datový set nějakým způsobem posunout, nemusíme přepočítávat polohu všech voxelů, stačí nám posunout dva ohraničující body boxu. Ručním omezením boxu můžeme provádět řezy datovou strukturou v rovinách souřadných os.

Výpočet průsečíků přímky a kvádru

Výpočet průsečíků přímky s kvádrem se bude dělat v případě, kdy chceme zjistit, kde paprsek protíná buňku, resp. obalový box objemu. Vzhledem k tomu, že vycházíme z reprezentace volumetrických dat pomocí uniformní kolmé mřížky, roviny této mřížky jsou všechny rovnoběžné s osami souřadného systému, a nepotřebujeme tedy řešit případ obecné polohy kvádru v prostoru. Z toho důvodu můžeme kvádr zadat pomocí minimálního a maximálního hraničního bodu (**boxMin** a **boxMax**). Tyto dva body definují roviny kvádru zadané jednotlivými souřadnicemi x, y, z. Složky souřadnice pak reálně odpovídají polohám voxelů v datové reprezentaci.

Nejčastěji používaný algoritmus pro výpočet průsečíků paprsku s kvádrem navrhli Kay a Kajiya (viz např. [31] i s obrázky). Výsledkem této metody jsou parametry t_{near} a t_{far} představující blízký a vzdálený průsečík s kvádrem, pokud paprsek kvádr protíná.

Paprsek je definován parametrickou rovnicí:

$$\mathbf{r} = \mathbf{o} + \mathbf{d} \cdot t,$$

kde $\mathbf{o}$ je počáteční bod přímky a $\mathbf{d}$ její směrový vektor. Nejprve se vypočítají průsečíky paprsku se všemi šesti rovinami kvádru, čímž dostaneme:

$$t_{\text{near}} = \frac{\text{boxMin}_i - o_i}{d_i},\ t_{\text{far}} = \frac{\text{boxMax}_i - o_i}{d_i},\ \text{pro} i \in \{x, y, z\}.$$

Problém spočívá v tom, že rovina stěny kvádru je obecně nekonečná, tedy průsečíky s rovinami získáme i v případě, že paprsek objem kvádru míjí. Porovnáváním velikostí parametrů odpovídajícím jednotlivým rovinám zjistíme, zda paprsek protíná roviny vevnitř kvádru či vně. Pokud výsledné t_{near} vyjde větší než t_{far}, znamená to, že paprsek protíná roviny mimo kvádr, neboť došlo k jejich prostorovému překřížení, paprsek tedy kvádrem neprochází. V opačném případě kvádr zasažen byl.

Metoda trpí určitými numerickými nedostatky. Problémem je především dělení nulou, což může nastat, pokud některá souřadnice směrového vektoru vyjde rovna 0. Williams a spol. v článku [30] upozorňují na to, že dle standardu IEEE je výsledek dělení kladného čísla nulou $+\infty$ a dělení záporného čísla nulou $-\infty$. Pokud však dělíme zápornou nulou, je to právě naopak, čímž by nebyla dodržena podmínka, že t_{near} je menší než t_{far}. Zápornou nulu můžeme dostat vynásobením kladné nuly nějakým záporným číslem. V článku je prezentován vylepšený

algoritmus, který izoluje dělení směrovým vektorem. Podle znaménka podílu $1/d_i$ se pak rozhodne o pořadí obou parametrů. Metoda byla v aplikaci implementována a otestována. Nejevila žádné nedostatky.

Identifikace buňky vzorku (dělení rozměrem)

U metod traverzace, které nehledají všechny protnuté buňky, ale načítají hodnoty vzorkováním parametru t paprsku (Ray Marching, Woodcock tracking), je zapotřebí identifikovat buňku, ve které se aktuální vzorek nachází, aby se pak na základě jeho pozice v buňce dala pomocí trilineární interpolace dopočítat jeho hustota a normála. V aplikaci je použitý velmi přímočarý postup, který však funguje dobře. Výše byl vysvětlen postup výpočtu obalového boxu a centrování datového setu. Bylo řečeno, že poloha bodů se odvíjí od úsečky, kterou paprsek objem protíná. Identifikace buňky se tedy taktéž odvíjí od obalového boxu. Index buňky získáme dle následujícího vzorce:

$$Id_i = p_i / W_i + D_i / 2 \text{ pro } i \in \{x, y, z\},$$

kde p_i je hodnota aktuálního vzorku, W_i je velikost buňky a D_i je rozměr objemu v počtech voxelů. Pro případ zaokrouhlovací chyby v desetinách bylo nutné pohlídat si znaménko Id_i. Pokud vyšlo záporné, nastavila se hodnota na 0.

Výsledná hodnota indexu **Id** je však stále float. Jestliže vzorek nepadnul přímo na některý voxel, znamená to, že se nachází někde uvnitř buňky (drtivá většina případů) a desetinná část vyjadřuje normalizovanou polohu vzorku v rámci buňky. Desetinnou část **Id** tedy můžeme následně použít pro výpočet trilineární interpolace. Samotné desetiny se izolují odečtením celé části, čímž zbude pouze ta desetinná.

Tato metoda tedy vrací dvě hodnoty: celočíselný vektor indexů v objemu a desetinný vektor čísel normovaných dělením do rozsahu $\langle 0,1)$. Trilineární interpolace (popsaná např. v [11, 2]) je obecně definovaná pro libovolně umístěné vrcholy kolmé mřížky v prostoru, resp. mezi hodnotami osmi bodů zarovnaných ve vrcholech kvádru. Kdybychom tedy chtěli počítat hodnotu uprostřed buňky zadané hodnotami voxelů a jejich pozicemi, musely by se rozměry a polohy normalizovat, a teprve pomocí těchto údajů by se dala vypočítat hodnota vzorku uprostřed buňky. Desetinná část zde však již normovaná je, tedy trilineární interpolace se provádí nad jednotkovou krychlí, čímž odpadne několik nadbytečných výpočtů.

5.1.3.2 Dohledání protnutých buněk

Zezačátku jsme testovali přístup dohledáním všech protnutých buněk. Snaha byla vyjít z výhod uniformní kolmé mřížky a dohledat buňky bez nutnosti generování nějaké stromové struktury podprostorů a preprocessingu s ohledem na budoucí implementaci na GPU. Následující postupy jsme si sami odvodili a snažili se je naimplementovat a otestovat. Ačkoliv komentované algoritmy nejsou součástí finální aplikace, kde jsme dali přednost vzorkovacím

metodám, byly nezbytnou součástí studia a přispěly k lepšímu pochopení práce s volumetrickými daty a krizových stavů s tím souvisejících.

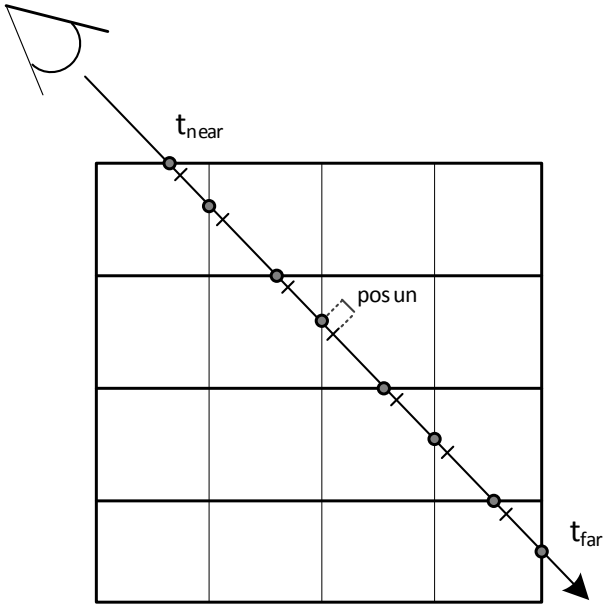
Brute force metoda

Nejjednodušší způsob, jak dohledat protnuté buňky objemu, je naivně otestovat průsečíky se všemi buňkami v objemu, čímž zaručeně najdeme ty protnuté. Algoritmus má však složitost $O(n^2)$, takže je prakticky nepoužitelný. Tento přístup jsme využili pouze na začátku pro otestování správného načtení dat z malého datasetu.

Dohledání buněk skokem parametru

Předtím, než byla v rendereru implementována metoda vzorkování parametru podél paprsku, snažili jsme se vymyslet způsob jak dohledat všechny zasažené buňky přímým postupem bez zbytečného testování neprotnutých buněk. První nápad byl určit nejbližší protnutou buňku, zjistit průsečíky s ní a do následující buňky se posunout skokem parametru.

Nejdříve tedy byly zjištěny průsečíky s obalovým boxem celého objemu a odpovídající parametry paprsku. Podle nejbližšího průsečíku ke kameře určíme první protnutou buňku (její index) dělením rozměrem. Spočteme průsečíky s touto buňkou a k tomu vzdálenějšímu připočteme nějakou malou hodnotu parametru (vesměs jen zlomek rozměru buňky). Tím bychom se měli dostat do sousední buňky podél paprsku, jejíž index opět zjistíme dělením rozměrem (viz Obrázek 15). Opakováním tohoto postupu teoreticky dohledáme všechny paprskem protnuté buňky včetně průsečíku s nimi, jež použijeme jako meze při integrování, resp. jejich odpovídající hodnoty útlumu v těchto bodech určené lineární interpolací. Když se vzorek dostane mimo mřížku, výpočet končí. Algoritmus při testování však jevil značné nedostatky. Hlavním problémem byly numerické chyby. Nejdříve se musí určit index buňky pomocí metody popsané v předcházející kapitole, a pak se provádí výpočet průsečíků paprsku s buňkou. V krizových případech se však stalo, že chybou přesnosti pohyblivé řádové čárky algoritmus počítající průsečíky došel k závěru, že buňka protnutá není. Dělením rozměrem však tato buňka byla vyhodnocena jako protnutá. Důvodem je pravděpodobně jiný počet dělení a násobení v obou algoritmech, který vedl k různým výsledkům. Tato situace by se pak musela řešit procházením okolních buněk a testováním, která z nich je ta pravá. Dalším závažným problémem byla situace, kdy přímka paprsku splývala s rovinou buňky. Vypočtené průsečíky s buňkou nebyly jednoznačné a posunutím parametru se stávalo, že vzorku byla stále přiřazována stejná buňka, čímž došlo k zacyklení. Parametr se totiž posouval neustále stěnou buňky. Vzhledem k rozsáhlým problémům jsme tento postup zavrhnuli.

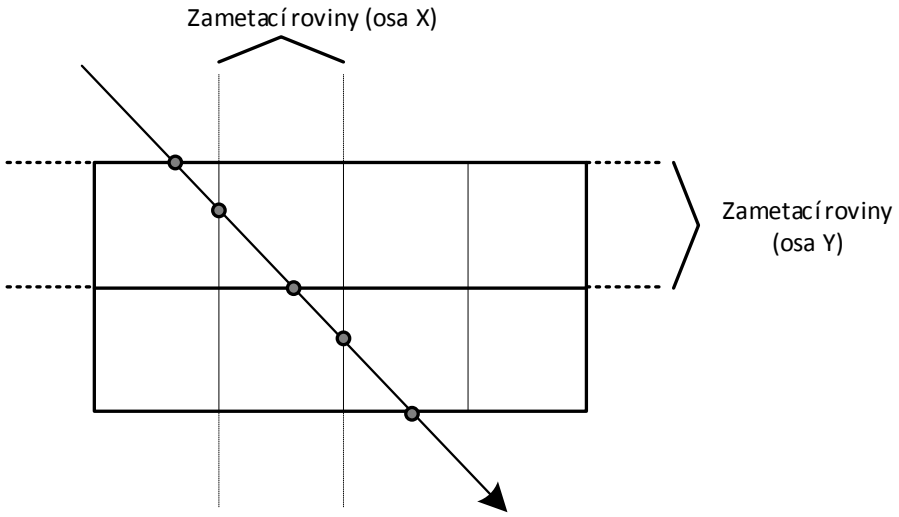


Obrázek 15: Průchod objemu posunutím parametru

Algoritmem řešícím průchod objemu podobným způsobem je 3D Digital Differential Analyzer popsaný Fujimotem [32], který rozvinul Amantides a Woo v článku [33]. Ten prochází buňky tak, že si hlídá hladinu buňky, ve které se aktuálně nachází. Při výpočtu průsečíku určí následující rovinu, kterou paprsek protne. Podle toho se rozhodne, kterým směrem leží další buňka. Index současné buňky a směr sledování je držen neustále v paměti. Na základě znalosti rozměrů objemu pak víme, zda vzorek opustil mřížku, nebo je stále uvnitř.Ve srovnání s výše popsanou metodou, je 3D Digital Differential Analyzer schopen vyhnout se chybám pohyblivé řádové čárky, neboť pracuje s celočíselnými inkrementacemi indexů.

Dohledání buněk zametáním rovinou

V druhém případě jsme vycházeli z myšlenky postupného zametání rovinou mřížkou objemu. Buňky by se pak daly zjistit na základě výpočtu průsečíku paprsku s rovinou ležící v jedné hladině voxelů mřížky (Obrázek 16). Podělením souřadnic průsečíků rozměrem buňky bychom pak dostali její index (jedna osa by byla dána hladinou, ve které se rovina vyskytuje, a ostatní souřadnicemi průsečíku s rovinou). Takto bychom však nezískali všechny protnuté buňky, neboť ty mohou být zasaženy z libovolné strany. Zametání by se pak muselo provádět pro každou osu zvlášť, aby se žádná nevynechala. Z toho plyne problém duplicit, protože ve většině případů paprsek protíná buňku ve dvou stěnách, tedy musela by být implementována nějaká fronta buněk, proti které by se ověřovalo, zda se již v ní ta aktuální nevyskytuje. Algoritmus má složitost $O(3n^2)$, neboť pro každý paprsek vyslaný prostorem testuje průsečík s n rovinami, což se děje pro každou ze tří os.



Obrázek 16: Průchod objemu zametáním rovinou

Tento postup má však zásadní nevýhodu v tom, že pořadí vkládání buněk do fronty nemusí odpovídat pořadí integrace podél paprsku. Když zametáním podél jedné osy získáme část buněk, zametáním podél ostatních os zařadíme nově nalezené buňky až na konec fronty. Pořadí je pak dosti podstatné pro realizaci útlumu podél paprsku, aby bylo patrné, která buňka je blíže kamery, a tedy má přednost. Fronta by se ve výsledku dala nahradit klasickým seznamem, který by se na závěr musel seřadit od nejvzdálenějšího po nejbližší element. Tento postup se tedy také příliš neosvědčil. Seřazení fronty pro buňky nasbírané podél každého paprsku by znamenalo navýšení složitosti algoritmu $n \cdot \log n$ krát, výsledná složitost by pak byla $O(3n^3 \cdot \log n)$, což je nakonec horší než brute force. Řešení postupným vkládáním do seřazeného binárního stromu představuje obdobný problém, kromě toho by se pak projevilo zpomalení integrace napříč objemem, neboť by se při ní musel procházet celý strom. K tomu se zde vyskytují opět stejné zaokrouhlovací chyby zmiňované v předchozím případě. Tato metoda tak nalézá využití snad jen u některé triviální zobrazovací techniky (např. součtové nebo průměrové), která neřeší pořadí buněk v akumulátoru. I tak je kvadratická složitost velmi vysoká, když vezmeme v úvahu, že vzorkovací metody jsou schopné objem procházet v lineárním čase. Reálná složitost je vzhledem k rozsáhlé režii a krizovým stavům ještě vyšší.

5.1.3.3 Vzorkovací metody

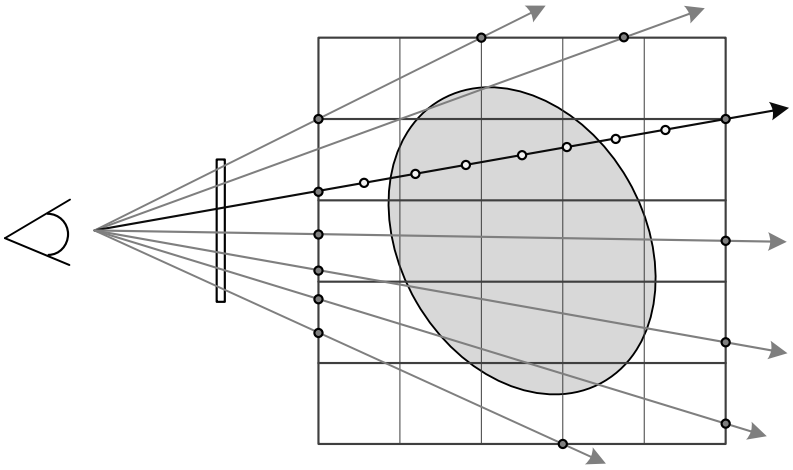
V předešlé kapitole bylo rozepsáno několik nápadů průchodu objemem a dohledání protnutých buněk. Podobné metody se v praxi používají, ale přístup k integraci nashromážděných hodnot se liší od těch vzorkovacích. Vyhledané buňky jsou popsané trilineární interpolací, ta se tedy dá spojitě integrovat. Do výsledné rovnice se pak dosadí hodnoty odpovídající průsečíkům paprsku s buňkou. Takto se projdou všechny buňky podél paprsku a získá se naakumulovaná hodnota pixelu obrazu. V případě voxelové reprezentace by pak nebylo nutné počítat integrál interpolace, ale dala by se přímo použít Levoyova metoda.

Vzorkovací algoritmy přistupují k dohledání hodnot diskrétními skoky podél paprsku. Prakticky se jedná o navyšování či snižování parametru t rovnice přímky v mezích hodnot t_{near} a t_{far} získaných průsečíky s globálním obalovým boxem. V aplikaci jsou implementovány 2 vzorkovací metody, kterými jsou Ray Marching a Woodcock tracking.

Ray Marching

Ray Marching (RM) popsaný např. v článku [35] trasuje objem jednoduše nějakým konstantním krokem (Obrázek 17). Předpokládá se, že hodnota mezi dvěma vzorky je stále stejná. Krok parametru je zadán jako vstupní hodnota algoritmu. V této práci je krok určen na základě diagonály obalového boxu. Vycházelo se z faktu, že diagonála je nejdelší možnou úsečkou, kterou může paprsek protínat objemem. Argumentem funkce tak nebyl přímo konstantní krok, nýbrž počet vzorků podél diagonály. Spočte se tedy přímka procházející body tělesové uhlopříčky, získá se hodnota parametru, který je potřeba k dosažení jednoho hraničního bodu diagonály od druhého, a tento interval se podělí počtem zadaných kroků. Rozsah parametru t zde nebude od 0 do 1, protože se při trasování vždy počítá s normalizovanými směrovými vektory. Celý postup se pak dá shrnout do následujících bodů:

1. Na základě zadaného počtu vzorků se dle diagonály vypočítá konstantní krok parametru.
2. Z kamery se vyšlou paprsky objemem.
3. Pro každý paprsek se vypočítá průsečík s obalovým kvádrem algoritmem popsaným v úvodu kapitoly (získáme t_{near} a t_{far}).
4. V zjištěném intervalu vzorkujeme parametr t konstantním krokem získaném v bodu 1.
5. Dosazením parametru t_0 aktuálního vzorku do rovnice paprsku dostaneme související bod.
6. Pomocí techniky dělení souřadnic rozměrem buňky aktuálního bodu získáme index buňky, ve které se nachází, a jeho normalizovanou polohu uvnitř ní.
7. Trilineární interpolací zjistíme hustotu odpovídající aktuálnímu vzorku a normálu v něm.
8. body 4 až 7 se opakují pro každý vzorek, dokud nedojde k opuštění mřížky objemu, čímž dostaneme požadované vstupní hodnoty pro Levoyův integrál.



Obrázek 17: Konstantní krok Ray Marchingu

V důsledku zaokrouhlovacích chyb při dělením rozměrem se může stát, že se vzorek ocitne mimo definovaný objem. Aplikace pak hlídá nejenom hodnotu parametru t, ale také vypočtený index buňky, zda se nenachází mimo rozsah mřížky.

Ray Marching je velmi přímočará vzorkovací metoda. Její implementace je jednoduchá a funkční. Hodí se však především pro objemy vysoce heterogenní. V homogenních prostorách RM musí provádět velké množství kroků, než narazí na význačný vzorek, který přispěje do výsledného akumulátoru. Problémem jsou pak také rozsáhlé nulové prostory objemu, které na barvu pixelu nemají vůbec žádný vliv. Předpoklad konstantní hodnoty mezi vzorky se neslučuje s principy nestranných metod snažících se simulovat fyzikální podstatu šíření světla prostředím. Inkrementací parametru totiž může dojít k přeskočení nějakého objektu, tedy upřednostnění určité části objemu na úkor druhé. Řešením by mohlo být navýšení počtu vzorků, čas výpočtu příliš velkého počtu vzorků však dramaticky roste. Tyto problémy byly v aplikaci vyřešeny pomocí tzv. Woodcock trackingu.

Woodcock tracking

Algoritmus Woodcock tracking byl vynalezen Woodcockem a kol. v roce 1965 (popsáno v článcích [35, 36]) v souvislosti s jaderným výzkumem. Metoda je podobná algoritmu Ray Marching. Představuje nestranný stochastický vzorkovací algoritmus, který objem prochází náhodnými skoky parametru podél paprsku. Počet vzorků se pro různé části objemu liší v závislosti na hodnotě útlumu prostředí. Tím se šetří čas a upřednostňují materiály, které mají vysokou míru neprůhlednosti, zároveň jsou však ve výsledku obsaženy i méně výrazné materiály, čehož je docíleno citlivou Monte Carlo metodou. Trasování vychází z distribuční funkce:

$$F(t) = 1 - A(t,0), \tag{5}$$

kde $A(t,0)$ je integrál útlumu světla šířeného od předcházejícího po aktuální vzorek zmiňovaný v teoretické části u Blinn/Kajiya modelu. Po dosazení rovnice (2) do (5) dostaneme:

$$F(t) = 1 - \exp\left(-\tau \int_0^t \rho(s)\mathrm{d}s \right), \tag{6}$$

kde $\rho(s)$ je funkce hustoty objemu a τ je koeficient přepočítávající hustotu na útlum. Logaritmováním rovnice získáme:

$$-\ln(1 - F(t)) = -\tau \int_0^t \rho(s)\mathrm{d}s\,. \tag{7}$$

Woodcock tracking trasuje objem stochasticky dle odvozené distribuce. Když položíme $F(t) = r$, kde r je náhodné číslo v intervalu $\langle 0,1 \rangle$ vygenerované uniformní distribucí, jsme schopni ze vztahu odvodit parametr t, který představuje délku následujícího kroku podél paprsku. Reálná implementace pak počítá s maximální hodnotou útlumu, která se předpočítává při načítání dat z disku do paměti. Následující krok t je dán:

$$t = \frac{-\ln(1 - r)}{A_{\max}}, \tag{8}$$

kde $A_{\max}$ je hodnota maximálního útlumu datového setu. Kromě stochasticky určeného kroku je součástí algoritmu rozhodnutí, zda aktuální vzorek má smysl vůbec počítat. Vzorek je přijat s pravděpodobností:

$$\frac{A_t}{A_{\max}},$$

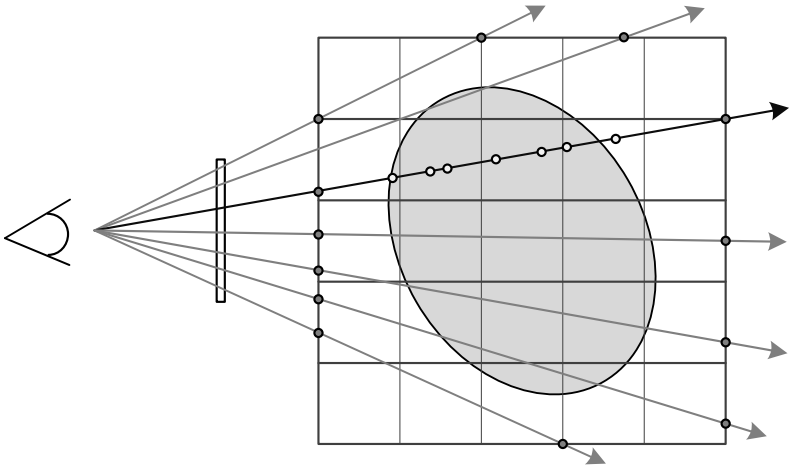
kde A_t je faktor útlumu aktuálního vzorku. Podílem tedy dojde k normalizaci.

Metoda se dá zapsat následujícím pseudokódem:

```
WoodcockTracking(t0, Ray, Amax)
  t = t0
  while( t > Ray_tnear )
    t = - ln(1 - rand()) / Amax
    if( rand() < At/Amax )
      break
    end
  end
  return t0 - t
end
```

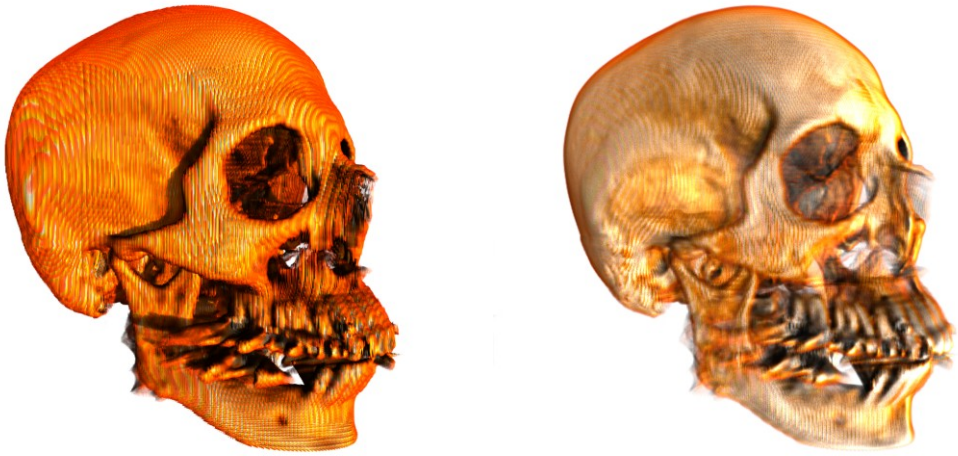
Objem se tedy trasuje od vzdáleného průsečíku s obalovým boxem k blízkému. Inicializační parametr t_0 je parametrem předcházejícího vzorku, od kterého se pokračuje v trasování postupným odečítáním. Cyklus běží, dokud není splněna podmínka přijetí, nebo dokud se vzorek nedostane na okraj boxu (dosáhle t_{near}). Funkce pak vrací rozdíl vstupního parametru t_0 a parametru přijatého vzorku. V implementaci je metoda rozšířena o ověření, že index buňky vybraného vzorku není již mimo rozsah mřížky (zaokrouhlovací chyba). Vzhledem k tomu, že skok daný rovnicí (8) se může blížit 0, je třeba určit minimální krok, se kterým se výsledný rozdíl porovnává. Ve vysoce nepravděpodobném případě by mohlo dojít k zacyklení. Když je rozdíl menší než stanovený minimální, vrací se ten minimální. Minimální krok se určuje jako polovina kroku spočteného na diagonále, jak bylo vysvětleno u metody Ray Marching. Uvedený pseudokód také nebere vůbec ohled na rozměry mřížky. Obecně Woodcock tracking počítá v průměru s jednotkovým krokem. Při dekrementaci se tedy musí skok daný rovnicí (8) vynásobit hodnotou uživatelsky stanoveného kroku. V aplikaci se používá krok vypočtený na diagonále jako v předchozím případě. Počet vzorků na diagonále je vhodné volit tak, aby základní krok byl blízký rozměrům buňky, čímž se minimalizuje efekt přeskakování některých částí objemu. Podstatné také je provádět stochastickou dekrementaci hned na začátku cyklu, nikoliv na konci. Výsledný obraz pak trpí artefakty. Pokud se dekrementace provádí až na konci cyklu, znamená to, že na začátku trasování je vždy přijat vzdálený průsečík s obalovým boxem. Všechny ostatní vzorky podléhají Monte Carlu, tudíž jejich hodnota má v obraze menší zastoupení než upřednostňovaný průsečík s boxem. V obraze je pak patrný obrys odpovídající hraničnímu řezu boxu.

Souhrnný postup je stejný jako v případě Ray Marchingu. Jediný rozdíl je v bodě 4., ve kterém se místo konstantního kroku použije výsledek stochastického trasování Woodcocka (viz Obrázek 18). Popsaný algoritmus je komplexním řešením procházení objemu. Je rychlejší než Ray Marching a na rozdíl od něj nestranný. Obrázky se pak jeví živější a díky stochastickému přístupu netrpí tak moc nežádoucím pruhováním způsobeným konstantním krokem u RM. Ve srovnání s metodami hledajícími všechny protnuté buňky, je Woodcock tracking implementačně jednoduchý a není potřeba ošetřovat nijak velké množství krizových stavů. Další výhodou je, že u všech ostatních metod by pro akceleraci průchodu prázdných prostor bylo nutné realizovat nějakou stromovou strukturu podprostorů, jako např. Oct-tree, Kd-tree nebo BSP-tree zmíněné v knize [37], nebo podobnou techniku popsanou Levoyem v [42], která dělí objem na části, kterým přiřazuje příznak, zda daný podprostor je nulový nebo nenulový. Woodcock tracking prochází prázdné prostory díky Monte Carlu řešícímu pravděpodobnost přijetí aktuálního vzorku. Metoda však není vhodná pro všechny typy dat. Vyplatí se především tam, kde se husté materiály vyskytují napříč objemem (např. kosti v lidském těle). V objemech obsahujících pouze malou hustou část dochází kvůli minimální pravděpodobnosti přijetí nevýrazných materiálů k jejich téměř absolutnímu eliminování. Pravděpodobně bychom přitom chtěli, aby byl vidět alespoň nějaký obrys takovýchto dat. Tato diplomová práce je ale zaměřená na vizualizaci medicínských snímků, k čemuž se Woodcock tracking výborně hodí.



Obrázek 18: Stochastický krok Woodcock trackingu

Kromě rozdílů popsaných výše je nutné upozornit, že Ray Marching a Woodcock tracking vyžadují různě nastavenou přechodovou funkci k tomu, aby bylo dosaženo přibližně stejných výsledků. Je to proto, že RM přijímá každý vzorek nacházející se v bodě odpovídajícímu násobku konstantního kroku, zatímco Woodcock přijímá stochasticky pouze vzorky s vyšší neprůhledností a vzorkuje s náhodným krokem. RM upřednostňuje určitou část objemu, zatímco Woodcock je nestranný. V důsledku toho pak u RM dochází k daleko větší míře „pruhování“. Vzorky sousedních paprsků jsou si vzhledem ke konstantnímu kroku velmi podobné, tudíž je z obrazu patrný vzor trasování. Stochastický Woodcock tyto artefakty lépe maskuje (Obrázek 19). Díky tomu, že Woodcock přeskakuje méně důležité oblasti, daří se mu lépe redukovat vady datového setu a nežádoucí elementy jako vlasy, přikrývky, polštáře, odrazy od zubních plomb atd. RM zobrazuje vše včetně chyb (Obrázek 20).



Obrázek 19: Srovnání Ray Marchingu (vlevo) a Woodcock trackingu (vpravo) pro stejné nastavení přechodové funkce. Levý obrázek trpí výrazným pruhováním. Kvůli konstantnímu kroku DVR také upřednostnil oranžovou barvu, která je u Woodcocka regulovaná stochastickým vzorkováním

Obrázek 20: Srovnání Ray Marchingu (vlevo) a Woodcock trackingu (vpravo) pro stejné nastavení přechodové funkce. Woodcock na základě Monte Carla zvýraznil hustší materiály (lebku), zatímco RM je zjevně překryl ke kameře bližšími tkáněmi a svalovinou. Levý obrázek je také daleko více zatížen nežádoucími artefakty plynoucími z nekvalitního zdroje dat

5.1.4 Přechodová funkce

Klasifikační krok Levoyovy integrační metody (viz [20]) přiřazuje jednotlivým materiálům neprůhlednost, vizualizační pak barvu a vlastnosti. K datasetům nebyly dodány žádné doplňující informace o materiálech. Bylo jim je tedy nutné ručně přiřadit. Přechodová funkce je spojitá funkce, která hustotám načteným z CT řezů přiděluje vlastnosti odpovídajícího materiálu. V diplomové práci přechodová funkce řeší neprůhlednost, barvu a složky osvětlení materiálu. Funkce je realizovaná pomocí 2D Beziérových křivek popsaných například v [2]. Pro každou složku vektoru barvy je zadaná jedna křivka. Výpočet Beziérových křivek je relativně náročný, problém představují například Bernsteinovy polynomy, jež se odvíjejí od kombinačního čísla, pro které je nutno počítat faktoriál. Aby se algoritmus urychlil, vychází se z předpokladu, že k definování přechodové funkce nebude zapotřebí více než 9 řídících bodů na křivku. Prvních deset faktoriálů je tak nadefinovaných dopředu v konstantním poli. Implementace na CUDA pak využívá templatovanou funkci, jejímž parametrem je počet řídících bodů, a následně unroll cyklu počítajícího Beziérovu křivku přes všechny body. S těmito úpravami výpočet přechodové funkce na GPU jede prakticky stejně rychle, jako když je definovaná intervalově pomocí série podmínek. Výsledná aplikace má přechodovou funkci rozdělenou do 2, jedna vrací pouze míru neprůhlednosti a představuje klasifikační krok, druhá pak materiál odpovídající dané hustotě, jenž se použije při stínování vzorků (vizualizační krok). Nastavení přechodové funkce pro lékařská data bylo určeno experimentálně tak, aby výsledek vypadal realističtěji. Na některých obrázcích jsou tak například kosti žluté, jinde již je použita bílá barva. Vzhledem k rozdílným kvalitám datových setů a nastavením přístrojů, kterými byly pořizovány, se musí přechodová funkce odladit pro každý soubor zvlášť, což je časově relativně náročná úloha. V podstatě se dá říci, že by mělo být barevné schéma upravováno současně

s úpravou funkce definující neprůhlednost, neboť se pak výsledek liší v detailech, což je způsobeno rozdílnou viditelností jednotlivých částí těla.

5.1.5 Výpočet stínování a osvětlovací model

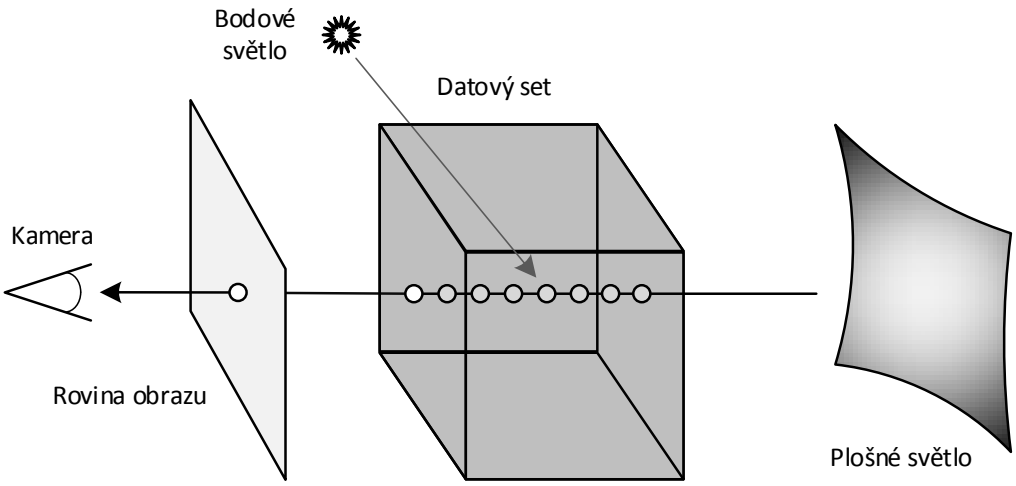
Osvětlovací model obecně definuje vizuální stránku scény, řeší, jak se bude který objekt nebo materiál jevit na výsledném obraze v závislosti na poloze světel, kamery a vlastností materiálů. U klasických Ray Tracingových algoritmů se nejčastěji používá Phongův osvětlovací model a Phongovo stínování (např. [12, 15]). Ten se hodí nejlépe pro lesklé povrchy s převládajícím reflexním odrazem. Dnes je již naprosto fundamentálním osvětlovacím modelem, takže jej zde nebudeme odvozovat. V případně trojúhelníkových sítí se využívá znalostí normál ve vrcholech trojúhelníků. Stínování podle Phonga pak počítá s normálou v bodě, kde paprsek trojúhelník protíná, na rozdíl od Goraurova stínování, které vypočte barvu ve vrcholech trojúhelníka a vevnitř ji pouze interpoluje. U volumetrických dat je situace trochu jiná. Vzhledem k tomu, že buňka je kvádrem, který má ve svých vrcholech umístěny voxely o nulovém rozměru, je zapotřebí hodnotu uvnitř buňky dopočítat trilineární interpolací zmiňovanou v 5.1.3.1. Ta se aplikuje na hustotu a normály ve vrcholech vypočtené gradientní metodou popsanou v kapitole 4.1. Takto tedy získáme hustotu a normálu v konkrétním bodě objemu. Hustotě je přechodovou funkcí přiřazena barva a jednotlivé složky materiálu. Zbývající postup je pak stejný jako v případě Phongova modelu. Osvětlovací model počítá s třemi typy odrazů světla ve scéně, které realizují jeho samotné šíření prostředím. Jsou jimi odrazy difúzní, spekulární a ambientní. Difúzní odraz převládá u materiálů, které jsou matné. Tento typ odrazu prakticky definuje základní barvu objektu. Spekulární složka naopak převládá u lesklých povrchů, kde se preferuje reflexivní odraz (vytváří světelné odlesky). Ambientní složka je pak jakési všudypřítomné světlo, které je zde proto, aby stíněné plochy objektů nenabývaly zcela černé barvy. V aplikaci je definovaná přechodová funkce pouze pro první dvě složky, ambientní se počítá jako míra difúzní, je tedy pouze dán koeficient, na základě kterého se přepočítá difúzní na ambientní. Tyto dvě složky nabývají obvykle pro jeden materiál stejné barvy pouze s jinou mírou světlosti či intenzity.

Použitím jednoduchého Phongova osvětlovacího modelu (viz [15]) získáme hezké obrázky především v místech, kde jsou jasně patrné iso-plochy (např. kosti), tedy tam, kde je prostředí nehomogenní a známe dobré odhady normál. V prostorách homogenních nastává problém, neboť zde je odraz světla nejednoznačný vzhledem k nejasným normálám. Homogenní oblasti se pak potlačují při Levoyově integrační metodě vynásobením neprůhlednosti velikostí normály přezásobené nějakým reálným číslem vyjadřujícím míru vlivu normály na neprůhlednost. Kromě toho nemáme zájem na tom, aby určité materiály byly tak výrazně lesklé. V případě tkání nebo svaloviny, která vyplňuje dosti velký prostor, chceme vytvořit dojem spíše matného materiálu s částečnou průhledností. Takového efektu lze dosáhnout rozumným nastavením přechodové funkce, která potlačí viditelnost i barvu u rozsáhlých homogenních oblastí, aby jimi naakumulovaná hodnota nebyla příliš ovlivněna. Utlumením barvy či jasu homogenních oblastí však nastává problém. Takové vzorky jsou ve výsledku přese všechno silně zastoupeny vzhledem k jejich množství a výsledný obraz se tím ztmaví a ztrácí na detailech. Když zvýšíme intenzitu světelného zdroje, výsledek je lepší, ale neřeší se tím odvrácená strana objektu, kam

padá stín. V aplikaci je ještě pro výpočet vrženého stínu vyslán z počítaného bodu vzorku stínový paprsek směrem ke světlu, podél kterého se počítá útlum světla objemem pomocí techniky Woodcock tracking zmíněné dříve. Tímto dojde ke ztmavení napříč celým objemem a odvrácené plochy scény se stanou prakticky černými. Jistá možnost je v posílení ambientní složky, která pro tento účel vznikla. Základní Phongův osvětlovací model je však modelem empirickým, nepříliš dobře respektuje fyzikální zákony šíření světla prostředím. Ambientní složka je zde chápána jako konstantní barva přičítaná k ostatním. Je tedy zcela nezávislá na poloze kamery a světla, čímž se obrázky stávají nepřirozenými. Výrazným posílením ambientního světla bychom tedy dostali méně plastický obraz s nevýraznými stíny, který je velmi přezářený. Z tohoto důvodu je dobré mít ve scéně světel více. Jedno bude zářit z čela a druhé zezadu. Světlo umístěné vzadu bude mít nižší intenzitu, aby nedošlo k úplné eliminaci stínu. Bodová světla však přispívají obvykle především k reflexivním odrazům a posílení odlesků. Kromě toho vytvářejí efekt ostrých přechodů mezi světlými místy a stínem. Pro nahrazení ambientní složky se tedy vyplatí implementovat jiný typ světel.

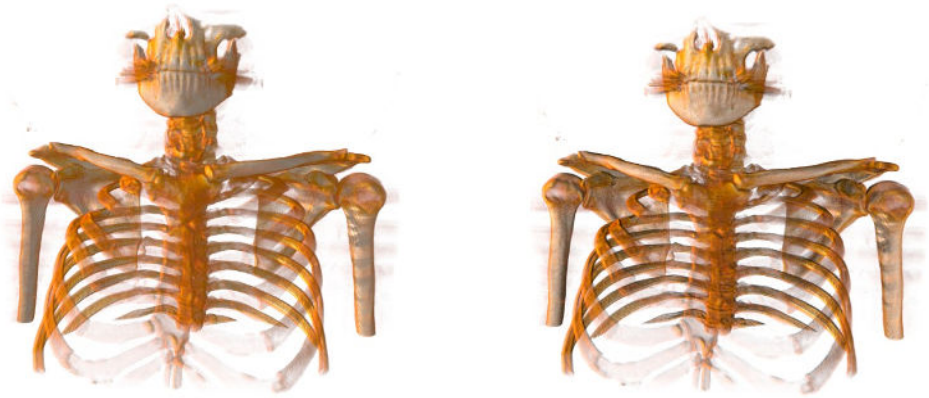
Rozšíření Phongova osvětlovacího modelu

Nahrazením ambientní složky a nestrannými modely se zabývají tzv. Globální osvětlovací modely [38]. Ty vycházejí z toho, že ambientní složka je nahrazena několika plošnými zářiči, nebo zářením prostředím. Na základě mapy prostředí se pak dá například simulovat efekt světla svítícího oknem do místnosti (viz [39]). Globální osvětlovací modely září na objekt prakticky ze všech stran, tedy i v místech, kam by bodové světlo vrhalo stín. U volumetrických dat není třeba vytvářet efekty mapou prostředí. Zde nám stačí myšlenka globálního osvětlovacího modelu. Zářením povrchem sféry do jejího vnitřku vytvoříme základní globální zářič, který sám o sobě však není dostačující. Objem by byl nasvícen ze všech stran stejně, což je nežádoucí. Lepší variantou je kulová plocha realizovaná výsečí povrchu koule. Vzhledem k tomu, že zdrojem světla zde není jen jeden bod, ale celé roviny nebo plochy, je zapotřebí rozhodnout se, které světlo bude jak ovlivňovat aktuální vzorek a jak na něj budou působit jednotlivá místa zářiče. Rovina i plocha mají obecně nekonečně mnoho bodů, je tedy nutné některé z nich vybrat. Výhodné je také, aby celá plocha nesvítila konstantní intenzitou, ale aby např. uprostřed bylo epicentrum záření, které by postupně do krajů oslabovalo. Pro vzorkování světel se používá tzv. Importance Sampling popisovaný v článcích [39, 40]. Ten řeší problém stochastickým vzorkování světelných zdrojů, bodů na plošných zářičích a BRDF (Bidirectional Reflection Distribution Function) diskutovanou v [40]. Každé světlo má určenou míru důležitosti, na základě které se pomocí Monte Carla vybere to, které bude ovlivňovat aktuální bod. Podobně je to také v případě vzorkování bodu plochy, která má přiřazenou pravděpodobnostní funkci, pomocí níž se stochasticky rozhodne o tom, které její místo bude na aktuální bod zářit. V případě BRDF se pak na základě materiálu (spekulární a difúzní složky) aktuálního vzorku stochasticky vypočte směr, kterým by se paprsek měl na povrchu odrazit. Jemným Monte Carlem tak dokážeme vytvořit rozumný poměr zastoupení difúzního a reflexního odrazu. Veach pak ve své práci [41] popsal metodu zvanou Multiple Importance Sampling, jež dokáže stochasticky kombinovat i několik zdrojů světla v jednom bodě. Pro naše účely jsou však tyto modely příliš komplexní. Primárně nám šlo o prozáření utlumeného objemu kulovou plochou, aby byla nahrazena ambientní složka Phongova osvětlovacího modelu a potlačeny nežádoucí odlesky na matných materiálech.



Obrázek 21: Implementovaný osvětlovací model

V práci jsou kombinovaná dvě světla (viz Obrázek 21). První je bodové, které přispívá k vytváření ostrých stínů a odrazů. Druhým zářičem je pak kulová plocha svítící obvykle na odlehlou stranu objemu, čímž se redukují jmenované nedostatky. Phongův osvětlovací model zde není přímo upraven pro stochastické vzorkování. Jeho rovnice je obecně definovaná pro více světelných zdrojů. Výsledný efekt je tedy vytvořen součtem příspěvků bodového světla a světla tvořeného bodem navzorkovaným na kulové ploše (viz Obrázek 22). Pro vzorkování plošného zářiče je využit Importance Sampling. Aby výpočet byl co nejrychlejší snažili jsme se vytvořit jednoduchý plošný světelný zdroj. Ten je realizovaný pomocí parametrické rovnice koule, jejíž parametry u a v jsou horizontální a vertikální parametry bodu povrchu. Vhodným nastavením intervalů u a v, které představují polární souřadnice koule, definujeme kulovou plochu (výseč). Vzorkováním těchto dvou parametrů vybíráme bod kulové plochy. Pravděpodobnostní funkce je zde zastoupena normalizovanou Gaussovou distribucí, jež nám zařídí, že body uprostřed kulové plochy budou protěžovanější než ty na okrajích (vytvoření epicentra záření). Aby se nemusela převádět náhodná čísla Monte Carla do Gaussovy distribuce, generují se přímo náhodné hodnoty v normálním rozdělení, které je normalizované do rozsahu 0 až 1, což umí i CUDA na GPU. Intenzita světla je pak vyjádřena euklidovskou vzdáleností (normalizovanou) od středního bodu plochy v soustavě polárních souřadnic, resp. jejím doplňkem do 1 (čím menší úhlová vzdálenost, tím vyšší intenzita). Takovýmto způsobem se nám podařilo naimplemenrčrtovat rychlý globální osvětlovací model, který objem prozáří a zároveň respektuje zásady šíření světla prostředím, tedy není konstantní jako ambientní složka. Kulová plocha se nastavuje počátečními úhly u_{start} a v_{start} a úhly rozsahu vyjadřujícími velikost kulové plochy ($u_{\text{glow}}, v_{\text{glow}}$). Oba typy světel mají své vlastní nastavení složek osvětlovacího modelu. Plošnému světlu se pak nastavuje ještě míra vlivu, aby se změnou jednoho parametru dala regulovat světlost scény. Předpokládáme, že použitím plošného světla dochází k nahrazení ambientní složky, tedy ta je pak nulová.



Obrázek 22: Srovnání osvětlení samotným bodovým zdrojem (vlevo) a modelu nahrazujícím ambientní složku plošným zářičem. Obrázek vlevo je světlejší a v důsledku přičtení konstantní ambientní složky postrádá detaily. Obrázek vpravo vypadá realističtěji a působí plastičtějším prostorovým dojmem

5.1.6 Souhrn zobrazovacího řetězce a integrační krok

Integrace hodnot podél paprsku je poslední fází přímého volumetrického rendereru. Naše aplikace implementuje Levoyovu metodu popsanou v kapitole 4.2.2.2. Výsledkem tohoto kroku je již finální barva pixelu obrazu. V případě, že je použit Super sampling, se příspěvky jednotlivých paprsků na závěr zprůměrňují. V této kapitole popíšeme souhrn celého zobrazovacího řetězce.

Zadané hodnoty:

- Vstupní kolekce řezů
- Rozměry řezů (jejich počet, výška, šířka)
- Rozměry mřížky (vzdálenost pixelů řezů a řezů mezi sebou)
- Rozlišení výstupního obrazu
- Hodnota Super Samplingu (SS×SS)
- Nastavení kamery
- Nastavení bodového a plošného světla
- Přechodová funkce (neprůhlednost, materiál)
- Normálový faktor (určuje vliv normály na neprůhlednost)
- Maximální / průměrná velikost gradientu, maximální útlum (zadané ručně, nebo vypočítané při načítání dat)
- Obalový box (zadaný ručně, nebo vypočítaný)
- Počet vzorků na diagonále

Nastavitelné parametry:

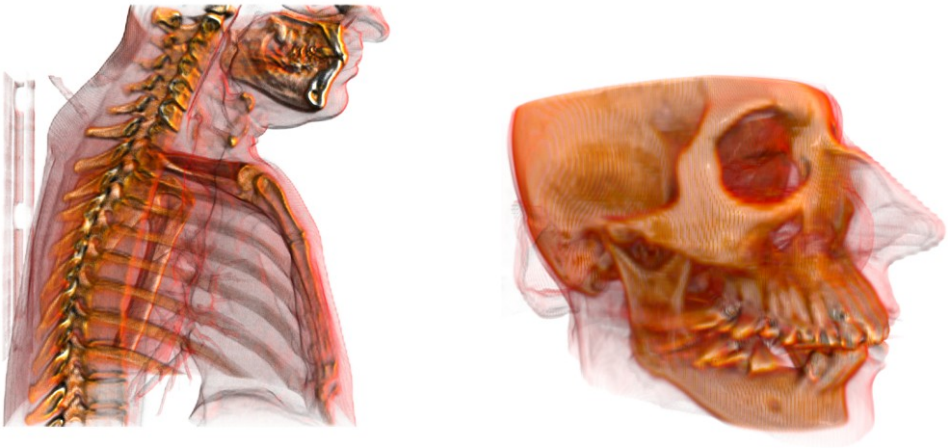
- COMPUTE_MAX_ATT – povolení / zakázání výpočtu maximálního útlumu
- COMPUTE_MAX_NORMAL_MAG – povolení / zakázání výpočtu maximální velikosti gradientu
- COMPUTE_AVG_NORMAL_MAG – povolení / zakázání výpočtu průměrné velikosti gradientu
- CAST_SHADOW – povolení / zakázání výpočtu vrženého stínu
- ELIM_HOMOGENOUS – povolení / zakázání eliminace homogenních prostorů
- ENABLE_WOODCOCK – povolení / zakázání Woodcock trackingu (jinak se používá Ray Marching)
- ENABLE_GLOBAL_SPHERE_LIGHT – povolení / zakázání plošného zářiče (když je 0, používá se klasicky ambientní složka u stínování)
- ENABLE_OPENGL – povolení / zakázání vizualizace výsledku pomocí OpenGL

Kompletní implementovaný renderer funguje následovně (viz také graf Příloha XV). Nejdříve se načte datový set do paměti na základě zadané adresy složky, kde se soubory řezů vyskytují, a jejich rozměrů. Provede se datová analýza, pokud je vyžadovaná, jejímž výstupem jsou maximální / průměrná velikost gradientu, maximální hodnota útlumu a indexy hraničních voxelů obalového kvádru (viz kapitola 5.1.1). Vypočítá se obalový box objemu, podle něj délka základního a minimálního kroku vzorkování podél paprsku, jak bylo uvedeno v 5.1.3.1. Provedeme inicializaci kamery a její báze a vyšleme jí paprsky do scény každým pixelem obrazu (Volumetrický Ray Casting), resp. každým subpixelem daným Super samplingem. Pro každý paprsek se spočítají jeho průsečíky s obalovým boxem. Objem se trasuje odzadu dopředu (Back to front propagation), tedy od t_{far} po t_{near}, stochasticky algoritmem Woodcock tracking (resp. Ray Marching konstantním krokem), kterým se předpočítají polohy vzorků podél paprsku (viz 5.1.3.3). K dohledaným vzorků se na základě zadaných rozměrů mřížky identifikuje buňka, ve které se vzorek nachází, a symetrickou diferencí získáme odhad normály ve voxelech buňky (viz 4.1). Trilineární interpolací dostaneme hodnotu hustoty a normálu uvnitř buňky, jak bylo vysvětleno v 5.1.3.1. K vypočítaným hodnotám hustoty se dle přechodové funkce popsané v 5.1.4 přiřadí neprůhlednost a materiál. Neprůhlednost vyjadřuje míru vlivu daného vzorku na výsledný pixel. V případě, že je povolená eliminace homogenních prostor, je hodnota neprůhlednosti přenásobená velikostí gradientu v bodě normalizovanou maximální velikostí gradientu v datasetu (zjištěno při načtení). Vzhledem k tomu, že CT snímky obecně zachycují lépe hustší materiály (např. kosti), eliminací homogenních prostor objem ztrácí na detailech. Aby tedy jemnější přechody nebyly úplně vynulovány, je zapotřebí velikost normály ještě podělit normálovým faktorem. Čím je větší, tím více se tlumí vzorky na jemných přechodech hustot a homogenní prostory. U datasetů s nevýraznými přechody se při nastavení faktoru menšího než 1 obrysy vizualizace naopak umocní. Pomocí získaného materiálu a normály se vypočítá osvětlení v bodě dle Phongova modelu (viz 5.1.5). Na scénu svítí několik světel (standardně jedno bodové a jedno plošné). Bodové světlo je obvykle nastaveno z čela datasetu a jeho intenzita je vyšší, plošné pak z druhé strany, aby prosvětlilo objem. Plošné světlo je realizováno pomocí parametrické rovnice koule a vhodného nastavení polárních souřadnic. Bod plošného zářiče je vybrán technikou Importance Sampling vzorkováním na základě Gaussovy distribuce se střední hodnotou 0 a rozptylem 1. Pokud je povolen výpočet vrženého stínu, vysílá

se ze vzorkovaného bodu stínový paprsek směrem ke světlu a vzorkováním dle Woodcock trackingu se načítá útlum světla od zdroje. Doplňkem této hodnoty k 1 se pak vynásobí dosud spočtená neprůhlednost. Nyní tedy známe finální neprůhlednosti jednotlivých vzorků podél paprsku i jejich barvu. Dosazením navzorkovaných hodnot do Levoyovy rovnice (4) z kapitoly 4.2.2.2 získáme finální naakumulovanou barvu subpixelu. Aritmetickým průměrem výsledných barev paprsků subpixelů daných Super samplingem dostaneme finální barvu pixelu obrazu. V aplikaci je ještě implementovaná gamma korekce, jejíž koeficient je součástí vstupních hodnot. Matice barev hotového obrazu se uloží do souboru PNG na disk a zobrazí se v OpenGL okně, pokud je to povoleno příslušným makrem.

5.1.7 Realizace uživatelských řezů

Součástí zadání diplomové práce bylo generování uživatelských řezů, tzn. nastavení určitých intervalů objemu, které se mají zobrazit. Oblasti mimo tyto intervaly jsou ignorovány. Aplikace umožňuje 2 typy řezů. Zaprvé je možné ručně nastavit hraniční indexy voxelů obalového kvádru objemu. Tím dosáhneme ořezání objemu rovinami rovnoběžnými s mřížkou (např. Obrázek 23 vpravo). Obvykle ale potřebujeme vytvářet řezy, které nebudou kolmé k rovinám buněk. Druhou možností je tedy řez objemu rovinou rovnoběžnou s obrazovou rovinou kamery (např. Obrázek 23 vlevo). Argumentem této funkce je parametr rovnice přímky kolmé na průmětnu. Kamera má zadaný svůj směrový vektor (normalizovaný) určující její orientaci. Přímka definovaná tímto vektorem a počátkem umístěným ve středu obrazové roviny je kolmá na tuto rovinu. Nastavením parametru uvedené přímky získáme kolmou vzdálenost od kamery, která je základem řezné roviny. Zobrazovací řetězec popsaný v předešlé kapitole se pak mírně upraví. Pro každý vyslaný paprsek se počítá průsečík s boxem, získáme t_{near} a t_{far}. Paprsky vyslané obecně z oka kamery spolu svírají nějaký úhel. Skalárním součinem normalizovaného směrového vektoru paprsku a normalizovaného směrového vektoru kamery získáme kosinus takového úhlu. Na základě kosinu úhlu a parametru vyjadřujícího kolmou vzdálenost od kamery vypočítáme parametr paprsku, který odpovídá bodu ležícímu na rovině řezu. Jinými slovy, takto vypočteme průsečík všech paprsků s rovinou řezu rovnoběžnou s obrazem kamery. Vypočtený parametr průsečíku se pak stává novým t_{near}, které nahradí původní blízký průsečík s obalovým boxem. V průběhu trasování objemu se tak výpočet zastaví na rovině řezu a dál nepokračuje (resp. blíže). Další obrázky jsou v příloze Příloha XII.



Obrázek 23: Řez hrudníkem rovinou rovnoběžnou s kamerou (vlevo) a řez obalovým boxem objemu (vpravo) omezeným shora

5.2 Implementace na GPU

Volumetrický Ray Casting je obecně úloha velmi náročná na výpočet. Prakticky je nutné řešit stejnou posloupnost výpočtů opakovaně pro každý paprsek a každý jeho vzorek. Podél jednoho paprsku se navzorkuje klidně i několik stovek hodnot. Pro každou z nich je nutné počítat vizualizační i klasifikační krok Levoyva algoritmu. Vzhledem k tomu, že paprsky vyslané z kamery jsou na sobě nezávislé, je možné výpočty barvy jednotlivých pixelů paralelizovat. Zadáním diplomové práce byla implementace na GPU pomocí technologie CUDA.

První podkapitola je teoretickým úvodem do architektury CUDA, která slouží především jako souhrn terminologie a vysvětlení některých pojmů, které se později v textu vyskytují. Kapitola 5.2.2 se věnuje interoperabilitě mezi OpenGL a CUDA. Kapitola 5.2.3 popisuje samotný návrh implementace na GPU a jeho problémy. Poslední část pak komentuje naměřené výsledky a porovnává je s implementací na CPU.

5.2.1 Architektura CUDA

CUDA je technologie od společnosti NVIDIA vyvíjená jako nástroj pro masivní paralelní výpočty na grafických kartách (viz [45]). Nejznámější konkurencí je OpenCL, která je multiplatformní. CUDA na rozdíl od OpenCL funguje pouze na grafických kartách NVIDIA. Výpočty na GPU a CPU jsou od sebe odděleny.

CUDA Dodržuje klasickou klient-server komunikaci mezi CPU a GPU (např. v [44]). CPU část se obecně nazývá host a GPU device. Architektura vychází z následující pipeliny:

1. Preprocessing na CPU
2. Inicializace GPU

3. Alokace paměti na GPU
4. Zkopírování dat na GPU
5. Provedení funkcí kernelu
6. Stažení výsledků z GPU
7. Uvolnění paměti na GPU
8. Zpracování výsledků, uložení na disk

Pracuje tedy tak, že data vypočtená CPU a uložená v RAM se musí zkopírovat do paměti GPU. Nejdříve je však nutné si paměť na GPU vyalokovat, což se dá provést pouze ze strany hosta. V rámci kódu prováděného na GPU (kernel) si novou paměť alokovat nemůžeme, můžeme pouze pracovat s pamětí vyalokovanou před zavoláním kernelové procedury. Po provedení výpočtu na kartě se výsledek stáhne zpět do RAM a na GPU se paměť vymaže, dále se data zpracovávají na straně CPU. CUDA umožňuje kromě nastíněného postupu také provádět asynchronní volání kernelu a asynchronní kopírovaní dat tam i zpět, pokud jsou tyto operace na sobě nezávislé.

Výpočet na kartě probíhá na několika multiprocesorech, které v sobě mají integrovaný určitý počet samostatných výpočetních jednotek. Počet procesorů jednoho multiprocesoru určuje velikost tzv. warpu. V rámci jednoho warpu se zpracovávají instrukce skutečně paralelně, tzn. vlákna jednoho warpu provádějí paralelně stejnou sekvenci instrukcí. Multiprocesory jsou pak schopny pracovat zároveň a každý provádět jiný výpočet. Pro vysokou míru paralelizace se vlákna běžící na kartě seskupují ještě do větších celků. Jsou jimi gridy a bloky, přičemž platí, že grid se dělí do několika bloků a bloky se dělí na jednotlivá vlákna. Toto rozdělení se nastavuje při volání kernelové procedury, a takto organizovaná vlákna se pak provádějí na GPU. Grid i blok mohou být až trojrozměrné, ale jejich velikosti jsou limitovány v závislosti na možnostech konkrétní grafické karty. Seskupování vláken do bloků umožňuje grafické kartě provádět výpočty paralelně nejen na úrovni vláken ale také na úrovni celých bloků, které jsou obecně považovány za nezávislé. Reálný běh programu a jeho efektivita se odvíjí od mnoha dílčích faktorů. Při implementaci je třeba dávat pozor na to, jak je který warp využitý, jestli se algoritmus příliš nevětví. Paralelizovatelná část výpočtu se pak zmenšuje a instrukce se musí provádět sériově. Dalším typickým omezením je špatná organizace paměti a přístupů vláken k ní. Problémové pak bývají synchronizace vláken a komunikace mezi nimi. Obvykle to znamená, že část vláken je nečinných, protože musí čekat na vlákna ostatní, tedy výpočetní zdroje nejsou naplno využity. Synchronizace na úrovni kernelu je možná pouze v rámci jednoho bloku. Pro synchronizaci celého kernelu je zapotřebí rozdělit ho na dvě samostatná volání. Komunikace mezi CPU a GPU je však nejdražší operací, je tedy dobré počet volání omezit na rozumné množství. V případě synchronizace na úrovni bloku je možné využít faktu, že v rámci jednoho warpu se všechna vlákna provádějí stejně, tedy také zároveň končí svůj běh. Pokud nám stačí synchronizovat počet vláken menší nebo roven velikosti warpu, nákladnému procesu synchronizace se dá vyhnout.

CUDA disponuje pěti druhy paměti:

- **konstantní** – rychlá paměť, velikostně omezená, využívá se pro často používané hodnoty napříč celým kernelem (např. parametry konfigurace), viditelná pro všechna vlákna i bloky, pouze pro čtení

- **globální** – hlavní paměť omezená prakticky jen velikostí grafické paměti, opět společná pro všechna vlákna i bloky, je však nejpomalejší, pro čtení i zápis
- **texturovací** – paměť uchovávající typicky obrazová data. Standardně je určena pouze pro čtení. Má výhodu v cashování blízkých hodnot ve 2D obrázku, tedy nenačítá se lineárně po řádcích textury, ale po 2D podblocích.
- **sdílená** – viditelná pro všechna vlákna jednoho bloku, určená pro komunikaci mezi vlákny (možný zápis a čtení), skoro stejně rychlá jako registry, velikost je omezená pro jeden blok
- **registry** – rychlá lokální paměť určená pro uchovávání hodnot proměnných prováděného algoritmu, stanovená kompilátorem

Uvedený přehled teorie je pouze výčtem základních vlastností CUDy. Architektura je neustále vyvíjena a možností pořád přibývá. Podrobnosti jsou velmi dobře rozepsány v oficiálních dokumentacích [45, 46]. Parametry pamětí, multiprocesorů, velikostí bloků atp. závisí na konkrétním zařízení. Výpočetní možnosti se vyjadřují verzí takzvané Compute capability (viz [46]), kterou je označena daná grafická karta. Podle toho se dá dohledat, jaké výpočty se na ní dají provádět. Naprogramování efektivního kernelu je do značné míry experimentální záležitost. Je nutné ho napsat tak, aby co nejlépe využíval zdroje. Míra využití se odvíjí od schopnosti karty provádět výpočty paralelně. Ta může být limitována velikostí bloků, rozdílným počtem instrukcí prováděných ve vláknech jednoho warpu, příliš velikou lokální či sdílenou pamětí nebo špatným přístupem vláken do globální paměti. Pokud si vlákna konkurují ve využívání zdrojů, zasahují do stejné části paměti nebo jsou některá vlákna příliš výpočetně náročná, jejich běh se serializuje. K ladění kernelů slouží nástroj Visual Profiler vyvíjený společností NVIDIA (popis např. v [45, 47]).

5.2.2 Interoperabilita technologií CUDA a OpenGL

CUDA umožňuje také dynamickou interoperabilitu s OpenGL nebo DirectX. V práci je použita interoperabilita s OpenGL využívající sdílené PBO a texturu, do které se pak výsledky zapíšou a zobrazí v OpenGL okně. CUDA disponuje vlastním API, které dokáže s OpenGL sdílet společnou paměť a komunikovat tak na úrovni GPU bez nutnosti data nahrávat z karty na hosta a zpět (popsáno např. zde [49]). Nejprve je nutné vytvořit si GL okno (pomocí knihovny GLUT) a nastavit kameru. Pak se alokuje paměť pro CUDu a nastaví se pro interoperabilitu pomocí souvisejících funkcí CUDA API. Na straně OpenGL se generují a registrují buffer objekty a textura, přes které budou obě technologie komunikovat. Pro OpenGL jsou namapovány požadované zdroje, čímž je zajištěno, že s nimi bude OpenGL aktuálně pracovat. Poté následuje zobrazovací smyčka, kdy CUDA po provedení výpočtu přepíše z úrovně kernelu sdílené PBO, jehož obsah se překlopí do 2D textury, která je zobrazena v OpenGL. V práci je tento princip použit pro zobrazení výsledků. Výhodou je, že velikost GL okna je možné měnit a výsledná textura v něm je dynamicky interpolovaná dle jeho rozměrů. Kromě této vizualizace se obrázek také ukládá do souboru na disk. Zobrazení výsledku v OpenGL je možné zakázat makrem.

5.2.3 Realizace DVR na GPU

Jak bylo v úvodu nastíněno, volumetrický Ray Casting je vysoce paralelizovatelným algoritmem. Jeho implementace na GPU je tedy zcela logická. Před samotnou realizací jsme se zamýšleli nad možnostmi, jakými by se náš renderer popsaný v kapitole 5.1 dal paralelizovat. V prvé řadě bylo nutné uvědomit si několik faktů souvisejících s řešeným problémem:

- Objemová data jsou velká (řádově klidně stovky MB)
- Výpočet jednoho paprsku je náročný (řeší se pro něj mnoho vzorkovaných hodnot)
- Výpočty paprsků jsou nezávislé
- Výpočty vzorků jednoho paprsku jsou závislé
- Jak se bude řešit Super sampling
- Kamera může být vzhledem k objemu umístěná libovolně
- Výpočet celého obrazu se musí rozdělit na dílčí volání kernelu

Uvažujme, že vstupní hodnoty a konfigurace našeho Ray Castingu, jak je zakresleno v grafu Příloha XV, byly vypočítány na straně CPU (datová analýza, světla, kamera, přechodová funkce). Konfigurační hodnoty rendereru jsou vesměs uloženy v konstantní paměti grafické karty, aby byly rychle přístupné. Objemný datový set se nahraje do globální paměti jako zarovnané lineární 3D pole hustot. Vzhledem k jeho velikosti není moudré předpočítávat si předem pole normál voxelů, neboť by to znamenalo čtyřikrát vyšší nároky na paměť (jeden float reprezentuje voxel, další tři jeho normálu). Normála se tedy počítá dynamicky až za běhu. Kromě toho je nutné vyalokovat si paměť pro uložení výsledného obrazu. Ten je reprezentován stejně. Pro generování náhodných čísel na CUDě je použita knihovna CURAND (viz [48]), která ke svému běhu taktéž potřebuje mít vyalokovaný prostor v globální paměti pro každé vlákno, které random využívá, aby si ta vzájemně nepřepisovala stavy generátoru. Stavy náhodných generátorů je nutné nejdříve inicializovat, což je relativně náročná operace, proto jsou inicializace izolované do zvláštního volání kernelu, který toto provede pro všechna vlákna. Druhý kernel pak již řeší samotné vykreslování.

Na vykreslování jsme nahlíželi dvojím způsobem, oba jsme se snažili vyzkoušet. Prvním případem je paralelizace celého výpočtu na úrovni jednotlivých pixelů obrazu. Druhá možnost je pak paralelizace nejen na úrovni pixelů, ale také na úrovni výpočtu jednotlivých vzorků.

Paralelizace na úrovni vzorků

Začněme od druhého případu, který se na první pohled jevil jako efektivnější z hlediska využití GPU, ale nakonec se příliš neosvědčil. Důvod, proč jsme šli touto cestou je fakt, že každý paprsek protíná objem jinak dlouhou úsečkou. Kvůli tomu také logicky získáme podél každého paprsku jiný počet vzorků. Když k tomu vezmeme v úvahu, že Woodcock tracking trasuje objem stochasticky, rozdílnost počtu vzorků na paprsek je vysoká. Kdyby tedy byl každý paprsek počítán jedním vláknem, výpočet jednotlivých vláken by silně divergoval, a tím snižoval míru využití zdrojů grafické karty. V případě paralelizace na úrovni vzorků by se měl tento efekt redukovat, neboť každý vzorek by měl být počítán zhruba stejně, tedy divergence v rámci warpu by měla být nízká. Parametr rovnice paprsku se však stejně musel navzorkovat, tak jsme renderovací kernel rozdělili do dvou dílčích. První předpočítával body vzorků podél

paprsku a ukládal je do zarovnané globální paměti. Trasování bylo realizováno 2D gridem s dvojrozměrnými bloky. Každý blok představoval jeden čtvercový (resp. obdélníkový) výřez obrazu. Každé vlákno bloku pak počítalo vzorky jednoho paprsku.

Druhý kernel potom k těmto navzorkovaným bodům dopočítal hodnoty vizualizačního a klasifikačního kroku Levoyouva integrálu. Tam byla vlákna organizovaná do 2D gridu jednorozměrných bloků. Každý blok měl dostatek vláken na to, aby byl schopen vypočítat všechny zjištěné vzorky jednoho paprsku. Vycházelo se z nejhoršího případu, tedy byl určen maximální počet vzorků na paprsek. Nejhorší případ představuje diagonála obalového boxu. V kapitole 5.1.3.1 jsme popisovali výpočet základního a minimálního kroku trasování. Na základě stanoveného minimálního kroku jsme schopni definovat maximální počet vzorků na paprsek. Od tohoto čísla se pak odvíjela velikost alokované paměti pro výpočet vzorků i počet vláken v jednom bloku. Při trasování se vzorky průběžně ukládaly do připravené paměti. Do přebývajících polí se uložila hodnota -1. Druhý kernel pak každému vláknu bloku přidělil jeden vzorek z paměti k výpočtu. Do této chvíle algoritmus postupoval slibně a na GPU se docela dobře paralelizoval. Problémem ale je, že výsledky klasifikačního a vizualizačního kroku se musí „skloubit“ dohromady při dosazování do rovnice (4), kde se jednotlivé barvy sčítají přenásobené hodnotou neprůhlednosti. Rovnice (4) je vyjádřená rekurentně. Prakticky z toho vyplývá, že pro výpočet příspěvku jednoho vzorku ve výsledném pixelu potřebujeme znát neprůhlednosti všech vzorků, které leží na přímce paprsku před ním ve směru k oku kamery. To potvrzuje poznámku v úvodu, že výpočet vzorků jednoho paprsku je závislý proces. K vyřešení finální barvy pixelu by tedy bylo zapotřebí uložit si neprůhlednosti jednotlivých vzorků do sdílené paměti, ze které by si ji mohla jednotlivá vlákna přečíst. Každé vlákno v závislosti na poloze jeho vzorku na přímce paprsku (vzdálenosti od kamery) by pak počítalo s různým počtem neprůhledností. Vezměme také v úvahu, že vzorků na paprsek můžou být stovky, tedy budeme potřebovat několik kilobajtů sdílené paměti na blok. S rostoucí velikostí sdílené paměti se snižuje schopnost grafické karty vykonávat bloky současně. Další nevýhodou je nutnost použití atomických operací při přidávání výsledků do akumulátoru, což způsobuje prodlevy při činnosti jednotlivých vláken, která přistupují do stejného místa v paměti.

Paralelizace na úrovni paprsků

První zmiňovanou možností byla paralelizace výpočtu obrazu na úrovni jednotlivých pixelů, resp. paprsků. Vlákna jsou tedy rozdělena do 2D gridu a 2D bloků, které představují dílčí výřezy výsledného obrazu. Každé vlákno pak řeší celou posloupnost instrukcí souvisejících s výpočtem barvy naakumulované podél paprsku včetně trasování objemem. V konečném důsledku se to jeví jako lepší varianta. Nejsme zde omezováni velikostí sdílené paměti a synchronizací vláken v bloku pro výpočet barvy akumulátoru. Také zde neexistuje problém s počtem vláken na blok, který musel být v předešlém případě vysoký, aby pojal proces výpočtu všech vzorků paprsku. Sice zde existují stále problémy divergence vláken, ale ty vzhledem k vysoké míře režie nezmizely ani v druhé variantě. Výhodou tohoto přístupu je také možnost výpočtu Super samplingu pomocí sdílené paměti. Předpokladem je, že rozměry Super samplingu nebudou větší než rozměry bloku. Každé vlákno pak zapíše svůj výsledek do sdílené paměti, jejíž obsah se na závěr zprůměruje. Sice se zde nevyhneme synchronizaci vláken, ale

průměr se počítá až na závěr, kdy stejně ostatní vlákna již nemají co dělat, takže se to ve výsledku neodrazí.

Testování ve Visual Profileru

V této části okomentujeme několik hodnot naměřených programem Visual Profiler (VP). Ten upozorňuje na nedostatky kernelu a hodnotí efektivitu výpočtu. Byly otestované oba přístupy a srovnány výpočty s Ray Marchingem a Woodcock trackingem. Měření bylo provedeno na datasetu CT Head (viz [26]) s dostatečným přiblížením kamery, aby žádný paprsek neminul objem. Rozlišení obrazu bylo 1024×768px s vypnutým Super samplingem (kvůli délce výpočtu). Test proběhl na GPU GeForce GTX460 1GB, Compute capability 2.1.

Při kombinaci metody paralelizace výpočtu vzorků a RM se u provádění obou kernelů podařilo dosáhnout nízké míry divergence (průměrně zhruba 5%). Tušení rovnoměrného zatížení vláken u této varianty tedy bylo správné. Trasovací kernel využívá zdroje asi ze dvou třetin, u druhého kernelu však VP hlásí nízkou míru využití okolo 18%. Jako hlavní důvod je uveden vysoký počet proměnných v registrech na vlákno (49). U RM je tento závěr logický, počet lokálních proměnných je malý a funkce prakticky nedělá nic jiného než, že přičítá k parametru paprsku konstantní hodnotu a výsledky ukládá do zarovnané paměti. Rozšířením paralelizace vláken na výpočty jednotlivých vzorků ale došlo ke zvýšení nároků na zdroje. Je to způsobeno pravděpodobně tím, že když jedno vlákno počítá všechny vzorky paprsku, stačí mu deklarovat si související proměnné pouze jednou, a ty si průběžně přepisovat, nebo k nim přičítat nově vypočtené hodnoty. V případě, kdy každé vlákno počítá jen jeden vzorek, musí mít jednotlivá vlákna pro svou činnost znovu deklarované stejné proměnné jako všechna ostatní. Počet proměnných na vlákno je tak sice stejný, ale vláken v bloku je více, ta tedy nedokážou běžet zároveň, neboť jeden multiprocesor má jen omezenou paměť určenou pro registry. RM v součtu pro celý obraz proběhne během 255ms, zatímco výpočet akumulované barvy trvá čtyřikrát déle. Efektivita přístupu do globální paměti je u obou kernelů nízká (13%), což se dalo očekávat. Kamera může být vzhledem k objemu polohovaná libovolně, nelze zde tedy zařídit efektivní čtení z globální paměti. Paprsky procházejí celým objemem napříč a protínají různé hladiny buněk. Paměť tak není vzhledem k vláknům nijak zarovnaná a přístup do ní je prakticky náhodný.

Slabým místem varianty s Woodcock trackingem je právě stochastické trasování. Kernel předpočítávající vzorky má míru divergence 62,8%. Efektivita přístupu do globální paměti je pak pouze 3,7%. Je to způsobeno tím, že náhodné krokování parametru přímky paprsku vnáší do výpočtů vláken daleko vyšší diverzibilitu počtu vzorků a jejich pozic, než je tomu u RM. Kromě toho počet lokálních proměnných je o něco vyšší a k tomu se zde generují náhodná čísla. Míra využití zdrojů je zde 31,5%, což je zhruba polovina ve srovnání s RM. Trasování Woodcockem trvá celkem 428,8 ms. Na druhé straně ale výpočet barev proběhne za 447,5 ms. Celkově je tedy tento přístup rychlejší. Stochastické trasování redukuje nepodstatné části objemu a prochází prázdné prostory, tím snižuje počet vzorků, který musí druhý kernel počítat. Zmenší se tak velikost bloků a míra konkurence jednotlivých vláken. Míra využití zdrojů je zde zhruba 27%, což je o něco více než v předchozím případě.

Dalším testovaným přístupem je paralelizace na úrovni paprsků, tedy jedno vlákno počítá celý paprsek. Test byl prováděn při stejném nastavení kamery i všech ostatních parametrů. Variantě s RM byla naměřena míra divergence 8,3%, což má obdobné důvody jako ve výše zmiňovaném případě. Konstantní krok trasování přispívá k vyrovnané zátěži jednotlivých vláken. Rozdíl je jen v tom, že tady trasování probíhá společně s akumulací barvy v jednom vlákně. Vzhledem ke značnému přiblížení kamery se počty vzorků u jednotlivých paprsků tolik neliší. Při vzdálenějším pohledu je ale situace podobná, neboť vlákna, která objem míjí okamžitě končí, tedy zátěž vláken jednoho warpu je vyrovnaná. Problém představují pouze okrajové případy (část paprsků míjí, část ne). Míra využití zdrojů je zhruba třetinová. Zajímavé je, že počet registrů na vlákno je zde 60, tedy o 11 více, než bylo předtím. Přesto je výpočet efektivnější. Může to být způsobeno počtem a konkurencí vláken v bloku, jak bylo popsáno dříve, ale také vyšším zatížením vláken, protože tady jedno vlákno počítá to, co předtím řešil celý blok. Efektivita přístupu do globální paměti je opět nízká (14,8%) ze stejných důvodů. Výpočet celého kernelu zde trval jen 513,67 ms, což je asi o 60% kratší doba, než v předchozím případě.

Stejný algoritmus se stochastickým trasováním má sice míru divergence 60,6%, ale využití zdrojů je opět zhruba na třetině. Ve srovnání s RM běží opět rychleji (celý kernel 323,1 ms). Jako limit výkonu je zde také označen vysoký počet registrů na vlákno (54). Efektivita přístupu do globální paměti je 12,6%. Prakticky i v tomto případě byl potvrzen očekávaný trend. S použitím Woodcocka dramaticky roste větvení algoritmů řešených jednotlivými vlákny, a s tím míra divergence, ale v konečném důsledku je výpočet rychlejší než s RM. Woodcock redukuje množství vzorků, tedy také počet přístupů do paměti a nároky vláken na zdroje. Ze srovnání obou druhů paralelizace vychází přímočarý postup výpočtu obrazu po pixelech (resp. paprscích) lépe. Je to zřejmě kvůli nižší míře režie, menšímu počtu registrů na vlákno a ušetření opakovaných přístupů do globální paměti, ke kterým dochází, když je výpočet rozdělen na dva kernely. Při provedení měření pro Super sampling 4×4 paprsků na pixel vyšly hodnoty prakticky stejné, jen čas výpočtu samozřejmě narostl. Efektivita algoritmu tedy není závislá na hodnotě Super samplingu. Velikost sdílené paměti pro tento účel je vždy 3kB při rozměrech 2D bloku 16×16 vláken. Ta se tedy nejeví jako omezující. Z těchto důvodů je popsaný způsob paralelizace také součástí výsledné implementace diplomové práce.

5.2.4 Měření

Datové sety používané v práci jsou řezy hrudníku dodané vedoucím práce, CT Head ze Stanfordu [26] a kompletní volumetrické datové sety ženy z projektu Visible Human [27].

Název	Rozlišení (x,y,řezy)	Formát
CT Head	256×256×99	TIFF
Hrudník	512×512×220	PNG
Visible Female Head	512×512×234	DCM
Visible Female Pelvis	512×512×150	DCM
Visible Female Ankle	512×512×150	DCM

Tabulka 1: Tabulka datových setů

Zdrojové snímky se liší svým rozlišením i kvalitou (viz Tabulka 1). Stanfordský dataset trpí poměrně výraznými vadami v obraze, které při vykreslování způsobují nežádoucí artefakty. Typickými chybami v CT snímcích jsou odrazy od kovových plomb, skokové jasové přechody mezi jednotlivými řezy, prstence způsobené špatnou kalibrací zařízení, šum nebo rozmazání způsobené pohybem pacienta. Některé z těchto chyb se dají řešit obrazovými filtry nebo vysokým Super samplingem, jiné se dají odstranit pouze opakovaným měřením. Některé však poškozují obraz natolik, že jejich odstraněním by došlo k znehodnocení snímků, které by pak neodpovídaly skutečnosti. Podrobnosti k vadám CT snímků jsou uvedeny v článku [28]. U datových setů uvedených výše se projevovaly především dvě vady, a to odrazy od plomb a jasové skoky mezi řezy. Nejhorším případem byl soubor CT Head, který jak svým nízkým rozlišením, tak vysokým počtem vad vedl k téměř matoucím výsledkům. Do měření byl zařazen jako referenční příklad, protože je ve výzkumu velmi populární a může sloužit pro srovnání. Soubory s rozlišením snímků 512×512px obvykle byly kvalitní, docházelo zde pouze k jasovým skokům. Opravu jasových vad jsme řešili ruční editací jasových složek v grafickém editoru. Výsledky byly uspokojivé, tudíž nebylo nutné implementovat nějaké automatické korekční algoritmy. V některých případech však byly rozdíly tak velké, že se pruhování nedalo vyhnout.

Testovaná zařízení

Náš renderer jsem testovali na několika zařízeních. Aplikace existuje ve dvou variantách. První jsme implementovali pro CPU, ve které byla použita jednoduchá paralelizace pomocí OpenMP. Tato verze však neumí všechno. Vzhledem k stále rostoucím nárokům na hardware se výpočetní čas dramaticky zvyšoval, některé vlastnosti byly tak implementovány pouze na GPU, kde výpočet jel podstatně rychleji. V měření jsme však testovali takovou konfiguraci rendereru, která byla srovnatelná v obou případech. Podstatné je, že na CPU i GPU bylo implementováno stochastické trasování Woodcock tracking a výpočet útlumu vysláním stínového paprsku ke světelnému zdroji. Tyto operace se jeví jako výpočetně nejnáročnější. Implementace pro GPU byla provedena na verzi CUDA 5.0 jako 32-bitová aplikace napsaná v jazyce C++. Přestože v ní nevyužíváme mnoho z v5.0, přišlo nám rozumné ji psát a kompilovat pro nejnovější standard,

který by měl umět lépe využívat zdroje moderních grafických karet. Požadavkem pro spuštění aplikace je tedy splnění této verze.

Testování probíhalo na CPU Intel Core i5 760 (4 × 2,8GHz), 16 GB RAM DDR3 a grafických kartách GeForce GTX460 a Tesla C2050 (srovnání parametrů v tabulce Tabulka 2).

NVIDIA Tesla C2050	
CUDA Driver Version / Runtime Version	5.0 / 5.0
CUDA Capability version number:	2.0
Total amount of global memory:	2688 MBytes (2818572288 bytes)
(14) Multiprocessors x (32) CUDA Cores/MP:	448 CUDA Cores
GPU Clock rate:	1147 MHz (1.15 GHz)
Memory Clock rate:	1500 Mhz
Memory Bus Width:	384-bit
L2 Cache Size:	786432 bytes
Total amount of constant memory:	65536 bytes
Total amount of shared memory per block:	49152 bytes
Total number of registers available per block:	32768
Warp size:	32
Maximum number of threads per multiprocessor:	1536
Device has ECC support:	Enabled
NVIDIA GeForce GTX460	
CUDA Driver Version / Runtime Version	5.0 / 5.0
CUDA Capability version number:	2.1
Total amount of global memory:	1024 MBytes (1073741824 bytes)
(7) Multiprocessors x (48) CUDA Cores/MP:	336 CUDA Cores
GPU Clock rate:	1430 MHz (1.43 GHz)
Memory Clock rate:	1800 Mhz
Memory Bus Width:	256-bit
L2 Cache Size:	524288 bytes
Total amount of constant memory:	65536 bytes
Total amount of shared memory per block:	49152 bytes
Total number of registers available per block:	32768
Warp size:	32
Maximum number of threads per multiprocessor:	1536
Device has ECC support:	Disabled

Tabulka 2: Tabulka srovnání vlastností obou GPU sestavená z výpisu utility DeviceQuery

Výsledky

V této části shrneme naměřené časy pro různé datové sety a pohledy kamery. Měření testuje výpočet na CPU i obou grafických kartách. Jedno je prováděno pro obrázky bez vrženého stínu,

druhé pak se stínem. Poslední měření na datovém setu pánve slouží pro srovnání obou grafických karet a bylo prováděno s nastavením vlastností, které v CPU verzi nebyly implementovány (především globální osvětlovací model). Testovací obrázky byly renderovány v rozlišení 1024×768px. V následujících tabulkách jsou zaznamenány naměřené časy pro tři různé datové sety a dva pohledy (čelní a boční), prostorově opačné pohledy jsou z hlediska výpočetní náročnosti prakticky stejné. Pod tabulkami je pak uveden komentář a zhodnocení měření.

	Bez stínu (čas v s)			Se stínem (čas v s)		
SS	C2050	GTX460	Core i5	C2050	GTX460	Core i5
1	3	3	12	7	7	37
2	6	7	45	17	21	143
4	18	26	173	56	75	557
8	63	99	688	208	290	2233

Tabulka 3: Datový set Hrudník (zepředu)

	Bez stínu (čas v s)			Se stínem (čas v s)		
SS	C2050	GTX460	Core i5	C2050	GTX460	Core i5
1	3	2	12	7	7	40
2	5	6	44	18	20	154
4	14	20	170	59	73	611
8	50	77	687	214	280	2428

Tabulka 4: Datový set Hrudník (z boku)

	Bez stínu (čas v s)			Se stínem (čas v s)		
SS	C2050	GTX460	Core i5	C2050	GTX460	Core i5
1	3	3	7	10	11	16
2	5	7	26	20	29	62
4	17	24	104	66	103	248
8	61	96	402	249	400	981

Tabulka 5: Datový set CT Head (zepředu)

	Bez stínu (čas v s)			Se stínem (čas v s)		
SS	C2050	GTX460	Core i5	C2050	GTX460	Core i5
1	4	3	6	10	12	15
2	7	8	24	22	31	60
4	19	30	98	71	111	235
8	73	115	386	269	431	947

Tabulka 6: Datový set CT Head (z boku)

	Bez stínu (čas v s)		Se stínem (čas v s)	
SS	C2050	GTX460	C2050	GTX460
1	2	2	4	4
2	3	4	9	9
4	8	13	24	33
8	30	47	86	126

Tabulka 7: Datový set Visible Female Pelvis (zepředu)

	Bez stínu (čas v s)		Se stínem (čas v s)	
SS	C2050	GTX460	C2050	GTX460
1	1	1	3	4
2	3	3	8	10
4	7	10	25	32
8	23	36	87	124

Tabulka 8: Datový set Visible Female Pelvis (zepředu)

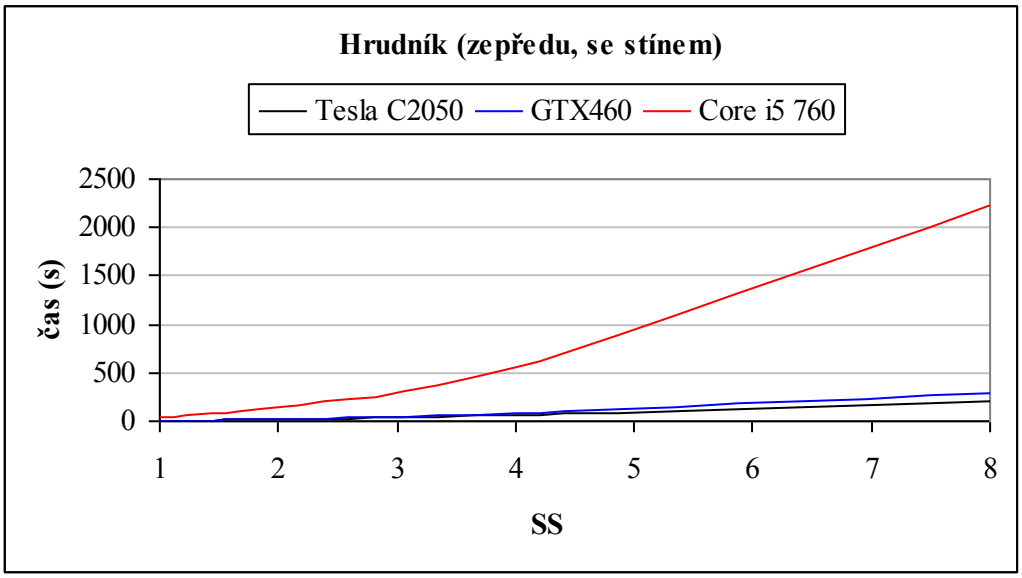
Z uvedených tabulek je vidět, že renderování není pohledově závislé. Ať se podíváme na kterýkoliv datový soubor, u žádného nevychází naměřené hodnoty nijak dramaticky různé pro pohled z čela a z boku. Je to pravděpodobně způsobeno tím, že řezy jsou čtvercové a v důsledku šumu a vad je obalový kvádr relativně velký, tedy ani v jednom případě nedochází k větší míře míjení boxu paprsky. Zadruhé je to kvůli stochastickému trasováním, které dokáže rychle procházet prázdné prostory. Dále si pak můžeme všimnout, že výpočetní GPU Tesla C2050 je skutečně o něco rychlejší než GeForce GTX460 (dle případu zhruba o 30% až 40%) především při vyšším Super samplingu. Tesla disponuje větším počtem CUDA jader (viz Tabulka 2), proto je schopná vykonávat více operací zároveň. Ačkoliv GTX460 má vyšší frekvence jader i paměti, v rámci možností paralelizace je na tom Tesla lépe, protože s vyšším počtem multiprocesorů se zvyšuje počet warpů, které mohou běžet zároveň. Větší počet jednotek navíc také umožňuje zpracovávat více vláken náročných na registry současně.

Když srovnáme libovolnou GPU s procesorem Intel Core i5, který má jen 4 jádra, vidíme, že akcelerací na grafické kartě došlo k velmi výraznému urychlení. Měření ukazuje, že u hrudníku (Tabulka 3 a Tabulka 4) došlo implementací na CUDě až k desetinásobnému zrychlení výpočtu. To potvrzuje fakt, že ačkoliv jádra procesoru mají vyšší výkon a jsou schopna pracovat prakticky nezávisle (není omezení paměti, registrů, velikosti warpu atp.), masivní paralelizací na GPU dosáhneme daleko vyššího výkonu.

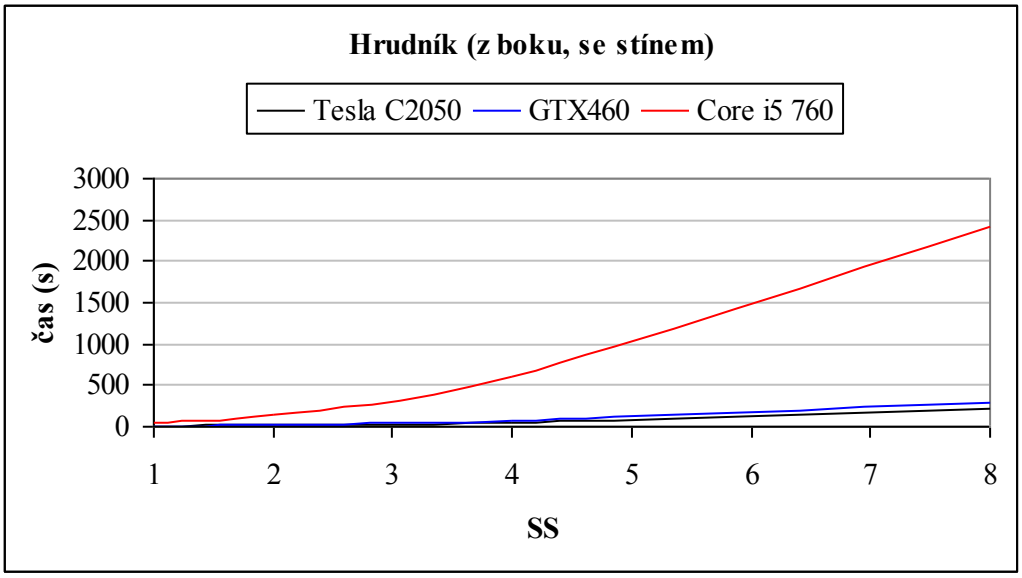
Dramatický rozdíl je pak mezi případy, kdy se počítá a kdy nepočítá vržený stín. U většiny datasetů počítání útlumu podél stínového paprsku navyšuje celkový čas zhruba na trojnásobek. V případě CT Head (Tabulka 5 a Tabulka 6) je to dokonce čtyřnásobek. Problém spočívá v tom, že vzorkování stínového paprsku znamená mnoho nových náhodných přístupů do paměti. U GPU výpočtů se tak jedná o neoptimalizované přístupy do paměti a provádění sekvenčních algoritmů navíc. Když vezmeme v úvahu, že trasování je realizováno stochasticky pomocí Woodcock trackingu, vnáší nám to do výpočtu ještě vyšší míru větvení a nesourodosti.

Z hlediska paralelizace je pak takový proces silně divergentní, a proto dochází k jeho serializaci. V případě GPU nám zde také naroste počet registrů na vlákno, což se z předchozí analýzy Visual Profilerem (viz odstavce výše) jevilo jako závažný limitující faktor. Z pohledu složitosti je nárůst času zcela logický. Bez vrhání stínu se provádí výpočet n paprsků odpovídající počtu pixelů obrazu (při vypnutém SS). Pro každý paprsek se pak počítá n vzorků. Když k tomu trasujeme stínový paprsek, znamená to, že pro každý vzorek každého paprsku kamery musíme ještě počítat stínový paprsek, podél kterého se počítá útlum na n vzorcích. Řádová složitost takového Ray Castingu tedy narůstá n-krát, což se nutně musí projevit na čase výpočtu, ačkoliv se při akumulaci útlumu nepočítá přímo barva a osvětlení. V naší aplikaci je výpočet vrženého stínu urychlen zdvojnásobením základního kroku trasování. I tak je akumulovaný útlum dostatečně reprezentující a bez toho by časy mohly být ještě o dost vyšší.

Následující grafy zobrazují závislost času výpočtu na velikosti Super samplingu. SS je v tabulkách nahoře uváděn jako jedna strana subpixelové mřížky, výsledný počet paprsků je pak kvadrát SS. SS roven 1 tedy znamená, že žáden aplikován nebyl. Závislosti byly ve všech případech podobné, uvádíme zde proto jen několik příkladů.

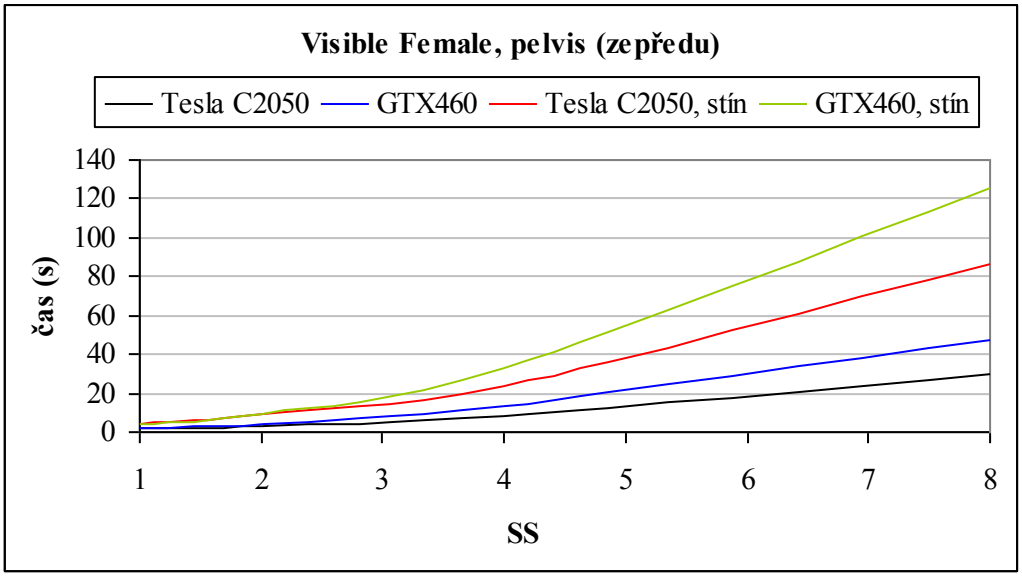


Graf 1: Závislost výpočetního času na SS, Hrudník, čelní pohled

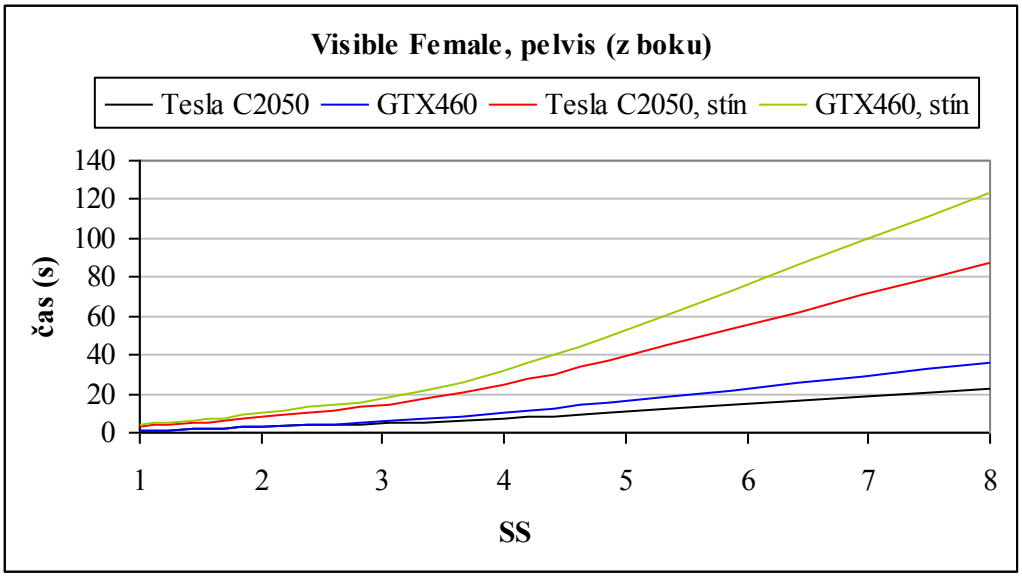


Graf 2: Závislost výpočetního času na SS, Hrudník, boční pohled

Grafy Graf 1 a Graf 2 vyjadřují růst výpočetního času v závislosti na velikosti Super samplingu u datového setu hrudníku. Vidíme, že v obou případech je vztah skoro stejný. Při malé hodnotě SS jsou si časy velmi blízké. CPU vychází o něco málo hůře, zatímco obě GPU začínají na téměř stejných hodnotách. Se zvyšujícím se SS se rozdíly prohlubují (u CPU dramaticky). Očekávali bychom víceméně lineární vztah, grafy však při nižších hodnotách vykazují spíše exponenciální závislost (podobně je tomu v případech dole). To je pravděpodobně způsobeno nízkou mírou vytížení zdrojů u malého nebo žádného SS. Roli zde bude hrát cashování paměti na GPU a také nižší počet výpočtů připadající na jeden multiprocesor. Od SS 4 a výše se křivka rovná do lineární závislosti.



Graf 3: Závislost výpočetního času na SS, Pánev, čelní pohled

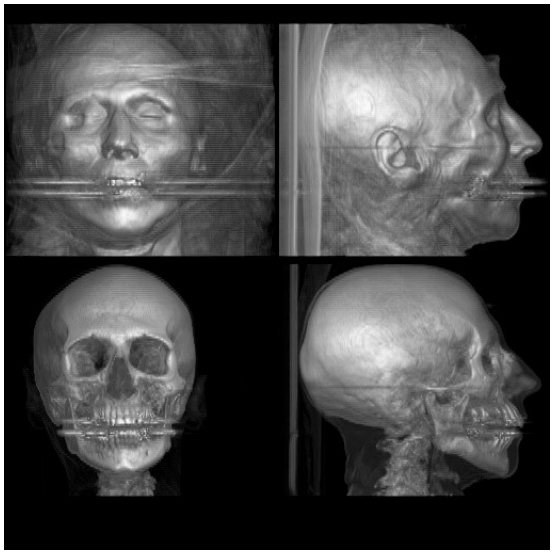


Graf 4: Závislost výpočetního času na SS, Pánev, čelní pohled

Měření nad datovým souborem Visible Female Pelvis bylo zaznamenáno do grafů Graf 3 a Graf 4. Důvodem, proč jsou časy u tohoto výpočtu nižší než v předchozích případech, je nastavení přechodové funkce pouze na zobrazování kostí (tedy se sníží počet řešených vzorků). Průběhy u čelního a bočního pohledu jsou si opět velmi podobné. Zde máme v jednom grafu zaznamenané jak výsledky bez stínu, tak se stínem. Vidíme, že mezi těmito variantami dochází u obou GPU ke skoku výpočetního času, jak bylo vysvětleno dříve.

Dosažená kvalita obrázků

Pár obrázků vykreslených naším rendererem bylo uvedeno již dříve pro vysvětlení některých algoritmů a efektů. V této části uvádíme několik dalších příkladů a porovnáváme je s výsledky cizích výzkumů. Stanfordský CT Head [26] je přes svou nízkou kvalitu velmi populárním datovým setem, který zde použijeme jako referenční. Levoy na něm prováděl svůj výzkum o vykreslování volumetrických dat, proto zde uvádíme jeho výsledný obrázek (Obrázek 24) realizující metodu přímého generování iso-ploch (viz [43]) stažený se stránek Stanfordu. Obrázek byl zveřejněn pouze v malém rozlišení na černém pozadí, je z něj ale patrné, že CT Head trpí množstvím nežádoucích artefaktů. Nejvýraznější jsou špatné odrazy od kovových plomb a vrstva polštáře, na kterém pacient ležel. Šum viditelný okolo hlavy je pravděpodobně způsoben dlouhými vlasy, které tento člověk asi měl.



Obrázek 24: Levoyův výsledek dosažený ve výzkumu o přímém zobrazování iso-ploch nad datovým setem CT Head. Obrázek ukazuje, jakými zásadními vadami datový set trpí

Levoyův výzkum je však starý již řadu let (1988), proto jsme chtěli naši práci srovnat s nějakým moderním algoritmem. Stejný datový set jsme tedy otestovali také v Exposure rendereru, který je výsledkem práce popisované v doporučovaném článku [1]. Aplikace pracuje s datovými sety ve formátu RAW. Náš program však vychází z jednotlivých obrázkových souborů reprezentujících řezy. CT Head ve formátu RAW má o něco více řezů než náš datový set, což je na našich obrázcích vidět u kratší páteře a useknuté brady.

Obrázek 25: Srovnání našeho výsledku (vlevo) a výsledku Exposure rendereru (vpravo) nad datovým setem CT Head (zobrazení lebky)

V obou případech jsme se snažili nastavit přechodovou funkci zhruba stejně. Na obrázku Obrázek 25 je srovnání Exposure rendereru (vpravo) s tím naším (vlevo). Na obrázcích je zobrazena lebka, která je na CT snímcích nejvýraznější, a proto v těchto místech jsou také nejlepší normály. Výsledek vykreslený Exposure rendererem se zdá být srovnatelný s tím naším. I když vpravo mají evidentně implementovaný komplexnější osvětlovací model, který vede k zářivějším barvám a výrazným odleskům, není rozdíl mezi výsledky nijak propastný. Je také otázkou, jakou praktickou vypovídací hodnotu takový „nablýskaný“ obrázek má. V diplomové práci jsme tedy profesionální renderer vizuálně nepředčili, ale podařilo se nám k němu alespoň přiblížit. Ve srovnání s Levoyovými černobílými obrázky se ten náš jeví hezčí.

V druhém případě jsme se snažili porovnat oba renderery pro přechodovou funkci nastavenou tak, aby zobrazovala i jemnější tkáně a svalovinu (viz Obrázek 26). Tyto oblasti typicky mají horší normály. Na obrázku Obrázek 26 vpravo je opět viditelný výrazný odlesk, zatímco povrch hlavy v našem výsledku (vlevo) je víceméně matný. Pozice zdrojů světla je v každém z nich trochu jiná, což se projevuje na odlišně vrženém stínu, který je na obrázku vpravo ostřejší a směřující šikmo dolů, zatímco v druhém případě je vržen směrem vpravo. Zde je rozdíl opět zdůvodnitelný především odlišným osvětlovacím modelem. Výsledek Exposure rendereru také vypadá hladší, což svědčí o efektivnějším zpracování homogenních oblastí se špatnými normálami. Dle článku [1] je v jejich implementaci zakomponovaná Monte Carlo technika vzorkující fázovou funkci mezi izotropní v oblastech s nevýraznými normálami a BRDF v místech s dobrými odhady normál. Mimo to používají pro vyhlazení postprocessing rozmazáním obrazu Gaussovým operátorem. Exposure renderer si také dokázal lépe poradit s artefakty u zubních plomb. Zde se tedy jejich technika jeví jako lepší. Datový set CT Head příliš upřednostňuje kosti na úkor ostatních částí, což je vidět na obou obrázcích, které mimo obrysů lebky a kůže postrádají výraznější detaily. Prostory svaloviny a tkání mají podobnou hustotu, jsou tedy silně homogenní.

Obrázek 26: Srovnání našeho výsledku (vlevo) a výsledku Exposure rendereru (vpravo) nad datovým setem CT Head. Obrázky zobrazují jemnější tkáně a svalovinu

Ukázky dalších výsledků jsou vloženy do příloh diplomové práce. Naše aplikace byla testována na všech datových setech zmíněných v tabulce Tabulka 1.

6 Závěr

Diplomová práce shrnuje dnešní techniky vizualizace volumetrických dat se zaměřením na zpracování medicínských snímků a problémy, které s tím souvisí. Byly vysvětleny přímé zobrazovací metody a srovnány s technikami hledajícími povrch. Dále byl popsán charakter zpracovávaných diskrétních dat a práce s nimi. V jednotlivých kapitolách jsme se pak věnovali dílčím algoritmům vedoucím k vytvoření Stochastického Volumetrického Ray Castingu.

V praktické části byly vysvětleny implementované metody a porovnány s jinými možnými variantami. Práce popisuje celý zobrazovací řetězec našeho přímého rendereru. Program byl napsán v jazyce C++. Aplikace využívá metodu Volumetrického vrhání paprsků, podél kterých se vzorkuje prostor napříč objemem. Datová struktura vychází z vrcholové reprezentace voxelů v mřížce tvořících buňky, jejichž obsah je popsán trilineární interpolací. Normály ve voxelech jsou v práci odhadovány symetrickou diferencí sousedících voxelů. Dále byly implementovány dvě metody trasování zvané Ray Marching, která prochází objem po konstantních krocích, a stochastický Woodcock tracking procházející prostor dle odvozené distribuční funkce s náhodným krokem odvíjejícím se od důležitosti vzorkovaných oblastí objemu. K nasbíraným vzorkům se pak vypočítaly barvy a průhlednosti dané přechodovou funkcí. Barva v bodě je určena osvětlovacím modelem. Klasický Phongův model byl rozšířen o sférický zářič stochasticky vzorkovaný dle normálního rozdělení technikou Importance Sampling, čímž jsme vytvořili rychlý globální osvětlovací model nahrazující ambientní složku Phongova modelu. Globálním modelem se nám podařilo prozářit objem, který se pouze za použití bodového světla jevil jako velice tmavý, a snížit míru nežádoucích odlesků. Vypočítané neprůhlednosti a barvy pak byly integrovány Levoyovou metodou pro diskrétní prostor. Pro vyhlazení obrazu byl využit algoritmus Super sampling řešící šum, ostré hrany a falešné obrazce. Aplikace je určená pro vizualizaci volumetrických dat s průhledností. Umožňuje nastavování různých barevných schémat pomocí přechodové funkce realizované Beziérovými křivkami a vytváření uživatelských řezů rovinou rovnoběžnou s rovinou kamery.

Renderer byl akcelerován grafickou kartou. Implementace byla provedena na architektuře CUDA. Vyzkoušeli jsme dvě metody: paralelizace na úrovni vzorků a paralelizace na úrovni jednotlivých pixelů obrazu. První varianta se příliš neosvědčila, byla pomalejší a trpěla rozsáhlou režií vláken. Druhý postup byl použit ve výsledné aplikaci, neboť dokázal GPU využít lépe. Volumetrický Ray Casting je vysoce paralelizovatelným algoritmem, přesto jsou výpočty jednotlivých paprsků silně divergentní, což je způsobeno rozdílným množstvím prováděných výpočtů v jednom vlákně. Každý paprsek protíná objem jinak dlouhou úsečkou, liší se tedy i počty vzorků nasbíraných podél jednoho paprsku. Implementací na GPU však i tak došlo ke značnému zrychlení. Měření bylo prováděno na CPU a dvou grafických kartách, z nichž jedna byla výpočetní NVIDIA Tesla C2050. Tesla běžela rychleji oproti standardní grafické kartě (GTX460) zhruba o 30-40%. Akcelerace vzhledem k čtyřjádrovému CPU byla v případě datového setu hrudníku až desetinásobná. Implementace na architektuře CUDA se tedy jednoznačně vyplatila. Z naměřených dat bylo zjištěno, že pro vyšší hodnoty SS je nárůst výpočetního času lineární. Čas výpočtu také razantně roste se zavedením trasování útlumu

(zhruba na čtyřnásobek), pomocí kterého je realizováno vržení stínu. Aplikace byla testována pro několik různých datových setů.

V diplomové práci jsme dokázali naimplementovat Stochastický Volumetrický Ray Casting pro vizualizaci volumetrických dat vykreslující obrázky poměrně slušné kvality. Získané obrázky jsme porovnali s profesionálním Exposure rendererem, který se jeví jako špička v dnešní době. Naše výsledky jej sice nepřekonaly, ale podařilo se nám k němu alespoň přiblížit. Zjistili jsme, že pro vykreslování hustých prostorů s kvalitními normálami se náš renderer velice dobře hodí. Nevýrazné a homogenní části objemu jsou však v obrázcích hůře patrné. Akcelerací pomocí CUDA se nám podařilo vytvořit zobrazovací algoritmus, který i na běžně dostupných zařízeních (GeForce GTX460) dokáže při nižších rozlišeních proběhnout během několika málo sekund.

Možná budoucí práce

Vylepšením aktuální implementace by mohlo být zavedení komplexnějšího osvětlovacího modelu, který by se rozšířil o Importace Sampling několika plošných světel a BRDF funkce. Zaměřit bychom se také mohli na zobrazování homogenních prostor a nevýrazných částí objemu. Z měření vyplývá, že při nižších rozlišeních by mohl renderer pracovat i v reálném čase. Podrobnějším rozebráním problematiky paralelizace Stochastického Ray Castingu a otestováním několika dalších variant by mohlo dojít k zefektivnění paralelizace výpočtu na GPU. Zajímavou úlohou by pak mohlo být zpracovávání rozsáhlých datových souborů, které se nevejdou celé do paměti grafické karty.

Literatura

[1] KROES, Thomas, POST, Frits H., BOTHA, Charl P. *Exposure Render: An interactive photo-realistic volume rendering framework*. PLoS, 2012.

[2] ŽÁRA, Jiří, BENEŠ, Bedřich, SOCHOR, Jiří, FELKEL, Petr. *Moderní počítačová grafika*. 1. vyd. Brno: Computer Press, 2004. ISBN 80-251-0454-0.

[3] GIBSON, Sarah F. Frisken. *Using Distance Maps for Accurate Surface Representation in Sampled Volumes*. Proceedings of the 1998 IEEE symposium on Volume visualization, strany 23-30, New York, USA: ACM , 1998.

[4] KREJZA, Marek. *Methods of iso-surface visualization and methods of iso-surface extraction from Volumetric Data*. Plzeň, 2000. Diplomová práce na fakultě Aplikovaných věd Západočeské univerzity na katedře Informatiky a výpočetní techniky. Vedoucí diplomové práce Prof. Ing. Václav Skala, CSc.

[5] PAWASAUSKAS, John. *Volume Visualization With Ray Casting* [online]. Worcester Polytechnic Institute, Massachusetts, USA, 1997. [cit. 2013-05-05]. Dostupné z WWW <http://web.cs.wpi.edu/~matt/courses/cs563/talks/powwie/p1/ray-cast.htm>

[6] VÁVRA, Radomír. *Rychlé zobrazovací algoritmy pro volumetrická data*. Praha, 2010. Diplomová práce na fakultě Elektrotechniky ČVUT na katedře Počítačů. Vedoucí diplomové práce Ing. Vlastimil Havran, Ph.D.

[7] ARVO, James, DUTRE, Phil, KELLER, Alexander, JENSEN, Henrik Wann, OWEN, Art, PHARR, Matt, SHIRLEY, Peter. *Monte Carlo Ray Tracing.* SIGGRAPH 2003 Course 44.

[8] DURAND, Frédo, CUTLER, Barb. *Monte-Carlo Ray Tracing.* Massachusetts Institute of Technology, USA, 2003. Dostupné z WWW <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-837-computer-graphics-fall-2003/lecture-notes/19_montecarlo.pdf>

[9] HUMPHREYS, Greg. *Ray Tracing Acceleration Techniques.* University of Virginia, USA, 2003. Dostupné z WWW <http://www.cs.virginia.edu/~gfx/Courses/2003/ImageSynthesis/scribed_notes/03_acceleration.pdf>

[10] CLINE, Harvey Ellis, LUDGE, Siegwalt. *Fast method of creating 3D surfaces by "Stretching cubes"* [patent]. USA. US6115048. Uděleno 5. srpna 2000. FPO.

[11] PARKER, Steven, PARKER, Michael, LIVNAT, Yarden, SLOAN , Peter-Pike, HANSEN, Charles, SHIRLEY, Peter. *Interactive Ray Tracing for Volume Visualization.* IEEE Transactions on Visualisation and Computer Graphics, roč. 5, č.3, strany 238-250. Salt Lake City, USA: IEEE, 1999.

[12] LYON, Richard F.. *Phong Shading Reformulation for Hardware Renderer Simplification.* Apple Technical Report #43. USA, 1993.

[13] TOSUN, Elif. *Phong Shading.* New York University, USA, 2007. Dostupné z WWW <http://cs.nyu.edu/~elif/phong.pdf>

[14] BLINN, James F. *Models of light reflection for computer synthesized pictures.* Proceedings of the 4th annual conference on Computer graphics and interactive techniques, roč. 11, č. 2, strany 192-198. New York, USA: ACM, 1977.

[15] PHONG, Bui Tuong. *Illumination for Computer Generated Pictures.* Communications of the ACM, roč. 18, č. 6, strany 311-317. New York, USA: ACM, 1975.

[16] CALHOUN, Paul S., KUSZYK, Brian S., HEATH, David G., CARLEY, Jennifer C., FISHMAN, Elliot K. *Three-dimensional Volume Rendering of Spiral CT Data: Theory and Method.* RadioGraphics, roč. 19, č. 3, strany 745-764. RSNA, 1999.

[17] FISHMAN, Elliot K., NEY, Derek R., HEATH, David G., CORL, Frank M., HORTON, Karen M., JOHNSON, Pamela T. *Volume Rendering versus Maximum Intensity Projection in CT Angiograph: When, and Why.* RadioGraphics, roč. 26, č. 3, strany 905-922. RSNA, 2006.

[18] DALRYMPLE, Neal C., PRASAD, Srinivasa R., FRECKLETON, Michael W., CHINTAPALLI, Kedar N. *Introduction to the Language of Three-dimensional Imaging with Multidetector CT.* RadioGraphics, roč. 25, č. 9, strany 1409-1428. RSNA, 2005.

[19] LORENSEN, William E., CLINE, Harvey E. *Marching cubes: A high resolution 3d surface construction algorithm.* Proceedings of the 14th annual conference on Computer graphics and interactive techniques, roč. 21, č. 4, strany 163-169. New York, USA: ACM, 1987.

[20] LEVOY, Marc. *A hybrid ray tracer for rendering polygon and volume data.* IEEE Computer Graphics and Applications, roč. 10, č. 2, strany 33-40. New York, USA: IEEE, 1990.

[21] TREECE, G.M., PRAGER , R.W., GEE, A.H. *Regularised marching tetrahedra: improved iso-surface extraction.* Computers & Graphics, roč. 23, č. 4, strany 583-598. Cambridge, UK: MSRA, 1999.

[22] LOW, Kok-Lim, TAN, Tiow-Seng. *Model simplification using vertex-clustering.* Proceedings of the 1997 symposium on Interactive 3D graphics, strany 75-ff. New York, USA: ACM, 1997.

[23] SOJKA, Eduard. *Počítačová grafika II: metody a nástroje zobrazování 3D scén.* 1. vyd. Ostrava: VŠB TUO, RCCV, 2003. ISBN 80-248-0293-7 Strany 8-32 a 64-84.

[24] LEVKINE, Henry G. *Prewitt, Sobel and Scharr gradient 5x5 convolution matrices.* Vancouver, Canada, 2012. Dostupné z WWW <http://www.hlevkin.com/articles/SobelScharrGradients5x5.pdf>

[25] FreeImage. *The FreeImage Project* [online]. [cit. 2013-05-05]. Dostupné z WWW: <http://freeimage.sourceforge.net>

[26] LEVOY, Marc. *The Stanford volume data archive* [online]. [cit. 2013-05-05]. Dostupné z WWW: < http://www-graphics.stanford.edu/data/voldata/>

[27] Megnetic Resonance Research Facility. *Visible Human Project* [online]. [cit. 2013-05-05]. Dostupné z WWW: <https://mri.radiology.uiowa.edu/visible_human_datasets.html>

[28] BOAS, Edward F., FLEISCHMANN, Dominik. *CT artifacts: Causes and reduction techniques.* Imaging in Medicine, roč. 4, č. 2, strany 229-240. Stanford, USA: ingentaconnect, 2012.

[29] BORMAN, Sean. *Raytracing and the camera matrix - a connection.* Notre Dame, USA, 2003. Dostupné z WWW: <http://www.seanborman.com/publications/raycam.pdf>

[30] WILLIAMS, Amy, BARRUS, Steve, MORLEY, R. Keith, SHIRLEY, Peter. *An efficient and robust ray-box intersection algorithm.* ACM SIGGRAPH 2005 Courses, Article No. 9. New York, USA: ACM, 2005.

[31] OWEN, G. Scott. *Ray - Box Intersection* [online]. [cit. 2013-05-05]. Dostupné z WWW: <http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter3.htm>

[32] FUJIMOTO, A., TANAKA, T., IWATA, K. *ARTS: Accelerated Ray-Tracing System.* Computer Graphics and Applications, roč. 6, č. 4, strany 16-26. 1986.

[33] AMANATIDES, John, WOO, Andrew. *A Fast Voxel Traversal Algorithm for Ray Tracing.* In Eurographics ’87, strany 3-10. Amsterdam, North-Holland: citeulike, 1987.

[34] BOURKE, Paul, COOKSEY, Chris. *Antialiasing and Raytracing* [online]. [cit. 2013-05-05]. Dostupné z WWW: < http://paulbourke.net/miscellaneous/aliasing/>

[35] SZIRMAY-KALOS, László, TÓTH, Baláže, MAGDICS, Milán. *Free Path Sampling in High Resolution Inhomogeneous Participating Media.* Computer Graphics Forum, roč. 30, č. 1, strany 85-97. Budapest, Hungary: ingentaconnect, 2011.

[36] YUE, Yonghao, IWASAKI, Kei, CHEN, Bing-Yu, DOBASHI, Yoshinori, NISHITA, Tomoyuki. *Unbiased, adaptive stochastic sampling for rendering inhomogeneous participating media.* ACM SIGGRAPH Asia 2010 papers, Article No. 177. New York, USA: ACM, 2010.

[37] BERG, Mark, CHEONG, Otfried, KREVELD, Marc, OVERMARS, Mark. *Computational Geometry.* 1. vyd. Berlin: Springer Berlin Heidelberg, 2008. ISBN 978-3-540-77973-5.

[38] RAAB, Matthias, SEIBERT, Daniel, KELLER, Alexander. *Monte Carlo and Quasi-Monte Carlo Methods 2006.* 1. vyd. Berlin: Springer Berlin Heidelberg, 2008. ISBN 978-3-540-74495-5. Kapitola *Unbiased Global Illumination with Participating Media,* strany 591-605.

[39] CLINE, David, EGBERT, Parris K., TALBOT, Justin F., CARDON, David L. *Two Stage Importance Sampling for Direct Lighting.* Proceedings of the 17th Eurographics conference on Rendering Techniques, strany 103-113. Aire-la-Ville, Switzerland: ACM, 2006.

[40] LAFORTUNE, Eric P., WILLEMS, Yves D. *Using the Modified Phong Reflectance Model for Physically Based Rendering.* New York, USA: citeseerx, 1994.

[41] VEACH, E. *Robust Monte Carlo methods for light transport simulation.* Stanford, 1997. Doktorská práce na Stanfordské univerzitě.

[42] LEVOY, Marc. *Efficient ray tracing of volume data.* ACM Transactions on Graphics (TOG), roč. 9, č. 3, strany 245-261. New York, USA: ACM, 1990.

[43] LEVOY, Marc. *Display of Surfaces from Volume Data.* IEEE Computer Graphics and Applications, roč. 8, č. 3, strany 29-37. North Carolina, USA: IEEE, 1988.

[44] DING, Chong. *CUDA Tutorial* [online]. [cit. 2013-05-05]. Dostupné z WWW: < http://geco.mines.edu/tesla/cuda_tutorial_mio/index.html>

[45] LUITJENS, Justin, RENNICH, Steven. *CUDA Warps and Occupancy.* GPU Computing Webinar 7/12/2011.

[46] NVIDIA. CUDA C Programming Guide v.4.2.

[47] NVIDIA. CUDA C Best Practices Guide v.4.1.

[48] NVIDIA. CUDA Toolkit 5.0 CURAND Guide

[49] BRAITHWAITE ,Wil. *Mixing Graphics & Compute with multi-GPU.* GPU Technology conference 2013.

Seznam příloh

Přílohy

Příloha I: Hrudník, zvýraznění páteře a kostí. Obrys měkkých tkání je potlačen, scéna je nasvícená zleva

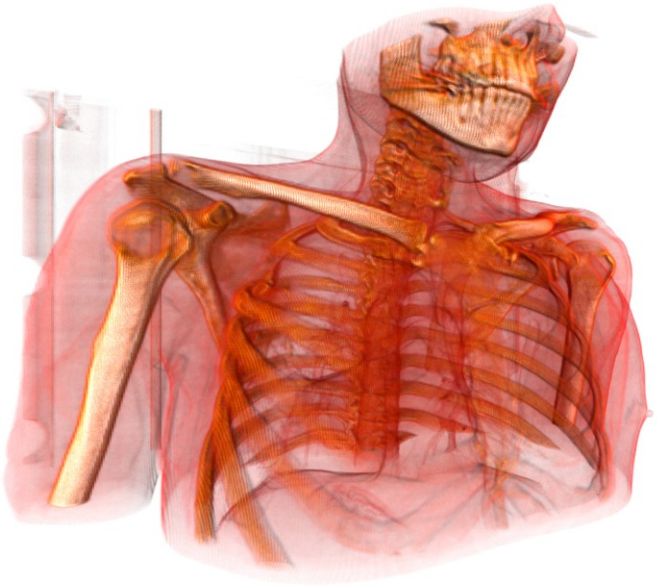
Příloha II: CT Head, lebka s odlesky, nasvícená zprava

Příloha III: CT Head, lebka nasvícená ze strany kamery, bez výrazných odlesků

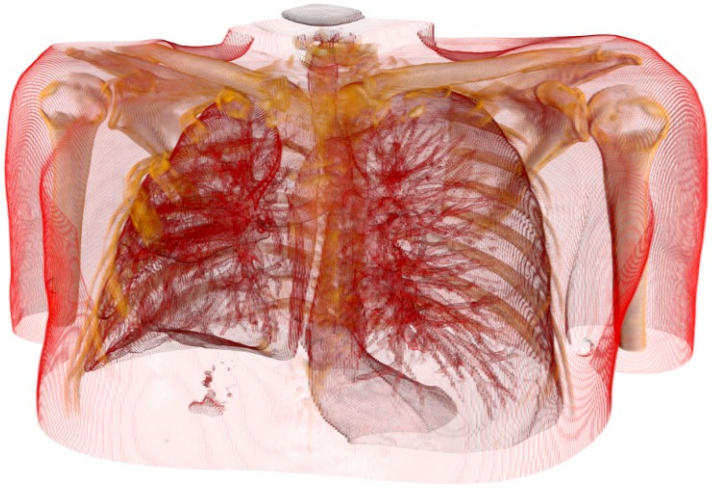
Příloha IV: Hrudník, čelní pohled. Ve scéně se nepočítá vržený stín, objem je hned viditelně světlejší a barvy zářivější

Příloha V: Visible Female Pelvis, zobrazení kostí rukou a pánve. Datový set trpí výraznými jasovými skoky, což způsobuje „obláčky“ dole a nahoře, které nebyly v rámci přechodové funkce zcela potlačeny

Příloha VI: Visible Female Pelvis, pohled ze strany



Příloha VII: Hrudník, pohled z úhlu

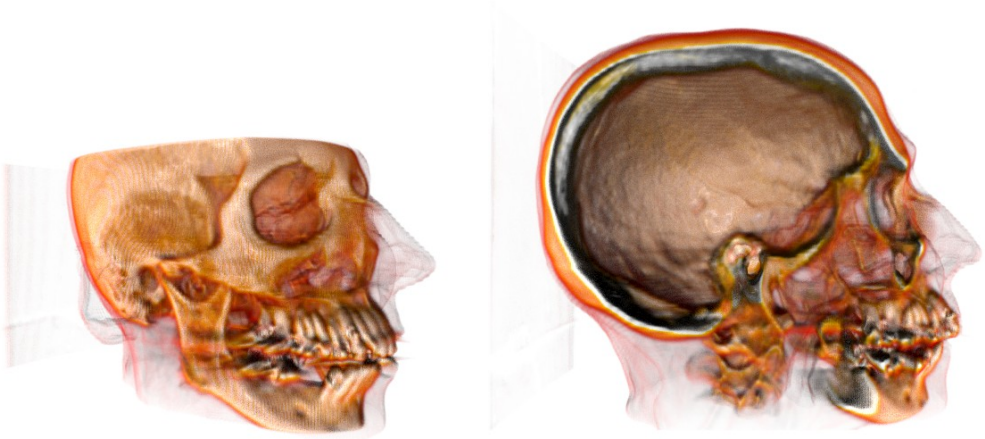


Příloha VIII: Hrudník, zobrazení struktury plic. Vzorkování pro tento účel muselo být velmi jemné, krok v hustších oblastech byl tak malý, což způsobuje pruhování v obraze. Důvodem jsou neznámé hodnoty mezi jednotlivými řezy, které se musí odhadovat trilineární interpolací

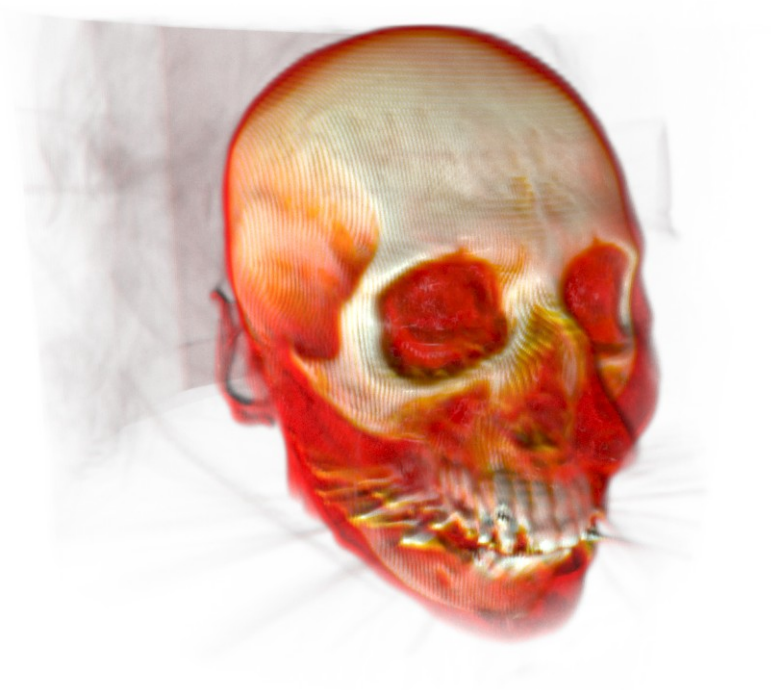
Příloha IX: Visible Female Ankle, pohled z úhlu

Příloha X: Visible Female Head, zobrazení lebky a povrchu kůže s průhledností

Příloha XI: Visible Female Head, zobrazení lebky. Na obrázku je viditelné, že datový set má dvojnásobné rozlišení oproti CT Head. Výsledek se jeví jako hladší

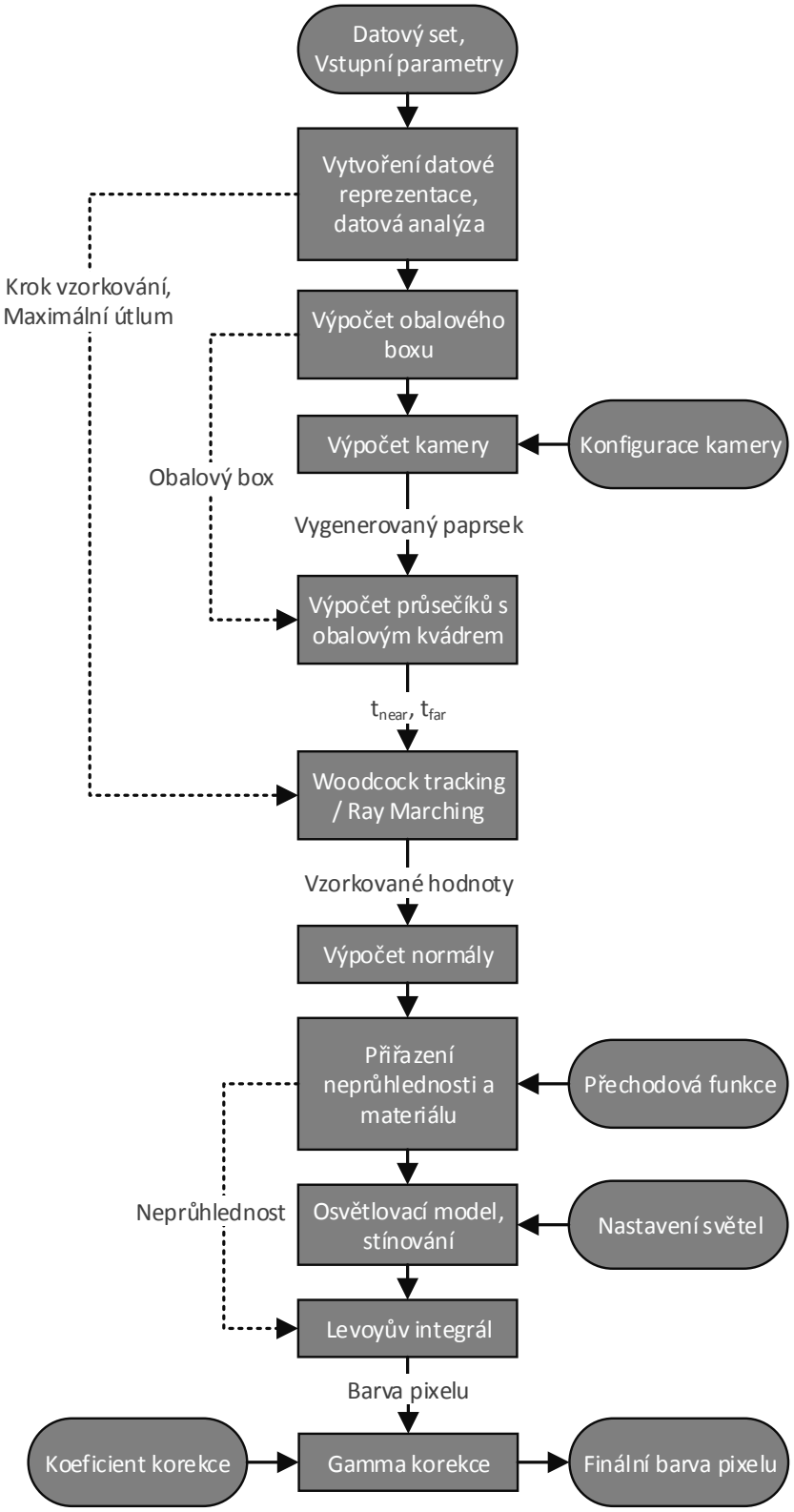


Příloha XII: CT Head, řezy obalovým boxem (vlevo) a rovinou rovnoběžnou s rovinou obrazu kamery (vpravo)



Příloha XIII: CT Head, zobrazení svaloviny a jemných tkání s průhledností

Příloha XIV: Hrudník, zobrazení svaloviny a jemných tkání s průhledností. Ve srovnání s CT Head je poznat vyšší rozlišení datasetu (ostřejší a hladší obraz)



Příloha XV: Kompletní diagram zobrazovacího řetězce našeho rendereru