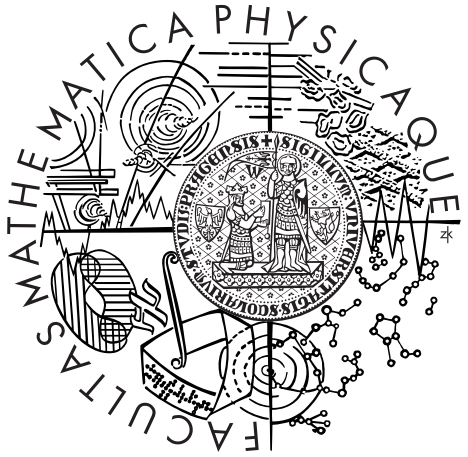
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Martin Holeček

Efektivní metody zobrazování objemových dat

Matematický ústav UK

Vedoucí bakalářské práce: RNDr. Ing. Jaroslav Hron, Ph.D.
Studijní program: Matematika
Studijní obor: Obecná matematika

Praha 2013

Poděkování za dobrou komunikaci, vedení a spolupráci patří panu J. Hronovi, dále pak pánům V. Krajíčkovi, J. Kolomazníkovi a J. Soukupovi. Za cenné poznatky pak též děkuji rodině.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Efektivní metody zobrazování objemových dat

Autor: Martin Holeček

Katedra: Matematický ústav UK

Vedoucí bakalářské práce: RNDr. Ing. Jaroslav Hron, Ph.D., Matematický ústav UK

Abstrakt: Cílem práce je prozkoumat přístupy a navrhnout implementaci vhodnou pro grafickou prezentaci nasimulovaných dat a dat získaných z CT, popřípadě MR. Budeme se zabývat metodami přímého zobrazování strukturovaných i nestrukturovaných objemových dat a směřovat k simultánnímu zobrazení původních dat a nasimulovaných výsledků. V první, spíše rešeršní, části probereme aspekty problému, vývoj jejich řešení a projdeme postupy, které se používají. Dále zvolíme dílčí řešení na základě poznatků o existujících algoritmech a prezentujeme vlastní implementace a modifikace algoritmů. Budeme se tedy zabývat numerickým řešením objemových integrálů předintegrováním a paralelizací procesu projektování čtyřstěnů a perspektivní korekcí. Pro praktické použití klademe důraz na efektivitu výsledného postupu, tedy výpočetní a paměťovou náročnost a přehlednost prezentovaných lékařských dat. Výsledky porovnáme s některými existujícími implementacemi (Paraview).

Klíčová slova: objemové zobrazování, předintegrace, nestrukturované sítě

Title: Efficient methods for visualization of volumetric data

Author: Martin Holeček

Department: Mathematical Institute of Charles University

Supervisor: RNDr. Ing. Jaroslav Hron, Ph.D., Mathematical Institute

Abstract: The aim is to make an overview of and present implementation usefull for rendering of simulated datasets and CT and MR datasets. We will examine the methods of direct volume rendering of structured and unstructured grids and head to meaningful simultaneous realtime rendering of both types. In the first part, we briefly present the development, problems and targets of the field. Next, based on knowledge about existing algorithms, we choose one solution and present our own implementation and modification of algorithms. In detail, the object of our study will be numerical solutions of volume rendering integral by preintegration and paralelization of the process of projecting tetrahedra and perspective correction. For practical reasons we focus on the efficiency, that means computation time, used memory and usefullnes for medical presentation. The results will be compared with some other existing implementations.

Keywords: volume rendering, preintegration, unstructured grid

Obsah

Úvod

Naší motivací je práce na okrajově zkoumaném problému rychlého zobrazování namodelovaných lékařských dat v kontextu dat původních s dobrou obrazovou kvalitou. To představuje vyřešení dvou problémů. Zaprvé fúze dvou objemových zobrazování – strukturovaných a nestrukturovaných dat (více dále); zadruhé zvolení vhodných (efektivních) postupů k rychlému a kvalitnímu zobrazování obrázků. V práci pro příklad představíme výsledky vizualizací pro napočtené proudění v cévě mozku a v některých místech porovnáme postupy a jejich vliv na zobrazený výsledek. Položme si cíl vyřešit i praktickou stránku problému – navržené aplikace vzniklé v kontextu práce jsou součástí příloh.

V první kapitole popíšeme a odkážeme na základní nutné postupy, které se při zobrazování dat používají. Sledujeme přitom dva cíle – tematicky setřídit a odkázat na potřebné vědecké zdroje a předestřít některé znalosti pro kapitolu druhou. Shrneme také vývoj zobrazování strukturovaných a nestrukturovaných sítí (structured, unstructured grids), metody předintegrování přechodové funkce (preintegration) a algoritmu projektování čtyřstěnů (cell projection, projected tetrahedra).

Druhá kapitola popisuje detailněji konkrétní vybraná řešení a vlastní přínosy, které jsou pak shrnuty v závěru spolu s porovnáním s jiným software.

K textu jsou přiloženy jak ilustrační obrázky, tak i obrázky vlastní, vytvořené vybranými prezentovanými metodami. Na vhodných místech poukážeme i na anglické ekvivalenty pojmů, které jsou více rozšířené než české.

Kapitola 1

Rešerše

Objemové zobrazování se používá v oborech inženýrských, medicínských i výukových. Cílem zobrazení je obecně podat informaci o objemových datech, která si lze představit jako trojrozměrné obrázky.

Objemová data z oblasti matematického a fyzikálního modelování reprezentují reálný objekt s vypočítanými daty (např. tlaky, rychlosti, ...). Data jsou uložená v uzlech, tedy bodech trojrozměrného prostoru. Blízké body tvoří jednoduché geometrické tvary, na kterých se data interpolují. Proto se takové síti bodů říká nestrukturovaná síť – poloha bodů a přesný tvar primitiv nejsou pevně dány.

Na druhou stranu medicínská data, získaná z tomografie (CT) nebo magnetické rezonance (MR), jsou uložená ve strukturovaných sítích, kde body leží na osově zarovnané trojrozměrné mřížce s pevnými vzdálenostmi.

1.1 Základy zobrazování

Celý problém je potřeba rozložit do dílčích oblastí. Zdrojová data jsou v našem případě vždy jednorozměrná – medicínská data jsou data intenzit a v případě nasimulovaných dat vždy zobrazujeme pouze jednu informaci (tlak, rychlost, ...).

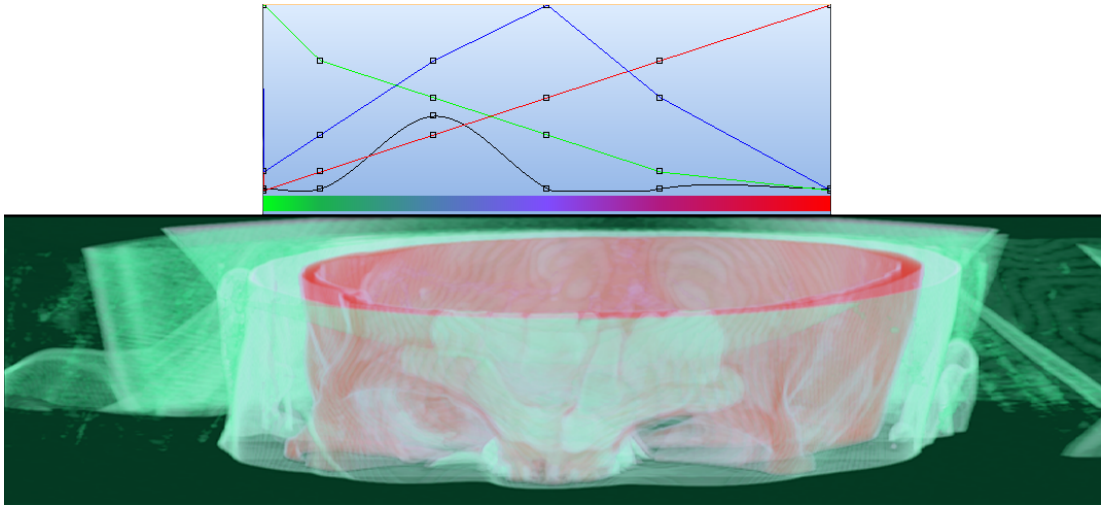
Proto se data obarvují (mapují na barevný přechod), což lze v literatuře nalézt pod označením přechodová funkce (transfer function). Jedná se o funkci, která intenzitám z normalizovaného rozsahu $[0, 1]$ přiřadí barvu a průhlednost alfa z prostoru RGBA.

Dalším důležitým bodem je filtrování dat kvůli zlepšení vizuálního dojmu. K rekonstrukci trojrozměrných obrazových dat se typicky používají filtry jako nejbližší soused, lineární interpolace, kubická interpolace. Porovnání a příklady budou uvedeny v další kapitole.

Důležitou roli pro výslednou kvalitu hraje i pořadí, v jakém aplikujeme přechodovou funkci a rekonstrukční filtr. Existuje zde několik postupů aplikací přechodové funkce a filtrování, jak uvádí [Had04].

- Preklasifikace – nejprve namapujeme barvy, pak interpolujeme již obarvené
- Postklasifikace – nejprve interpolujeme, následně mapujeme přechodovou funkcí
- Preintegrace – nejprve interpolujeme, následně mapujeme hodnoty dle metody předintegrování pomocí tabulky (viz dále).

Metody jsou seřazeny dle kvality výsledků. Nejhorší vizuální výsledky dávala preklasifikace, protože interpolaci provádíme na datech přechodové funkce a nikoliv na skutečných datech. Přicházíme tak často o podstatnou část informace (zdrojová data se mění často jinak než přechodová funkce). Důvod jejího používání byl historický – pouze tento postup bylo možné programovat na tehdejších grafických kartách.



Obrázek 1.1: Přechodová funkce v úzké oblasti kolem denzity kosti a pohled na výřez lebky.

Pro ilustraci užitečnosti přechodové funkce k vizuálnímu rozlišení objektů.

1.2 Objemové zobrazování strukturovaných sítí

Cílem této disciplíny je podat co nejlepší pohled na objemová data. Objemová data uložená ve strukturované síti si lze představit, jak bylo řečeno, jako trojrozměrné mračno bodů uložených v pevných vzdálenostech na osově zarovnané mřížce, tedy i jako řadu obrázků za sebou.

Disciplína se vyvíjela a vyvíjí ruku v ruce s hardwarovými možnostmi a požadavky na zobrazování. Nutno podotknout, že často není potřeba prezentovat data v přesném fyzikálním modelu a používá se i nerealistické renderování lépe zvýrazňující data. Dobrý přehled nerealistického renderování poskytuje například disertace M. Hadwigera [Had04].

Často se rozlišují dva způsoby zobrazování – přímé a nepřímé. Přímé metody (direct volume rendering, též DVR) pracují často pouze s původními daty, kdežto nepřímé metody (které se používají hlavně v případech, kdy nestačí výpočetní kapacita) si nejdříve data předzpracují. Nevýhodou přímých metod byla právě nemožnost rychlé implementace.

Pojďme se podívat na základní možnosti objemového zobrazování.

- MIP – projekce maximální intenzity (maximum intensity projection) – zobrazuje maximální intenzitu podél paprsků.
- MPR – multiplanární rekonstrukce – zobrazuje pouze vícero řezů objemem.
- Isoplochy (isosurface) – zobrazování isoploch odpovídajících pouze jisté hodnotě intenzity.

Zmíněné postupy se používaly zpočátku, kdy nebyl k dispozici výkonný hardware a byl velký rozdíl mezi kreslením v reálném čase a delší dobu počítanými obrázky. Například data isoploch se nejprve extrahovala pomocí algoritmu pochodujících kostek (marching cubes), popř. pochodujících čtyřstěnů [TPG98], aby se pak mohla rychle zobrazit. (Nyní se používají například i postupy založené na vaweletech [SG10].)

Následující postupy budou výpočetně náročnější, protože se snaží zobrazovat pomocí různých způsobů vyhodnocení tzv. volume rendering integrálu:

$$I(D) = \int_0^D C(s)\tau(s)\exp(-\int_s^D \tau(t)dt)dt$$

kde C je funkce barvy v bodě s, τ je funkce reprezentující průhlednost v bodě s a $I(D)$ je naintegrovaná barva. Základem je optický model svítící mlhy, který říká, že každý bod vyzařuje světlo dle své přechodové funkce a zároveň pohlcuje jakékoliv světlo mírou $1 - \alpha$. (Detailní odvození z fyzikálního modelu například v práci K. Morelanda [Mor04].)

- Řezy (slice based) – zobrazením několika řezů za sebou lze docílit aproximace požadovaného prostorového efektu, viz [SIY93].

Původně se používaly pouze tři sady dvourozměrných textur, mezi kterými se přepínalo, aby se vždy zobrazovala ta sada, která je nejvíce rovnoběžná se zobrazovací rovinou [RSEB+00]. Vylepšení v podobě trojrozměrných textur je popsáno v článcích o přímém zobrazování [CN94], [VGK96], [DKC+98]. Díky jednoduchosti tohoto řešení existují například i implementace v matlabu [Wan11], [Kro11], [Bru08], [Gue05], [Lud08].

- Shear warp algoritmus – interpretuje transformaci scény na skládání řezů do paměti přes sebe pouze posunem (warp) a změnou velikosti (shear), viz [LL94], [AGS95].

- Objektově orientované trasování paprsků (object oriented raycasting) – algoritmus pro ortogonální projekce počítající příspěvek texelů na jednotlivé pixely [MJC02].

- Fourierův řez (Fourier slice) – tzv. věta o fourierově řezu (fourier slice theorem) lze interpretovat tak, že jakmile máme k dispozici fourierův obraz dat, pak řez těmito daty poskytuje vyhodnocení integrálů podél paprsků kolmých na řez. Problematiku řeší práce [Lev92], zlepšení grafických výsledků [NNK04] a navrhování přechodových funkcí a implementaci na GPU [CC08].

- Statistické metody – využívají různé možnosti, jak zrychlit kreslení anebo vynechat pixely, které mají malý příspěvek k finálnímu vzhledu obrázku, viz např. práce o Monte Carlo volume renderingu [CSK03].

- Trasování paprsku (raytracing) – program přímo řeší pro každý pixel jeho výslednou barvu počítáním volume rendering integrálu podél vrženého paprsku do objemu/scény.

Trasování paprsků probíhá v opačném pořadí než popisuje fyzikální model. Vyhodnotí barvu každého pixelu tak, že numericky vyhodnocuje integrál podél paprsku, který míří z kamery do scény. Pokročilé techniky umožňují simulovat efekty jako lom nebo rozdvojení paprsku (subsurface scattering) [MKB+03], anebo realističtější osvětlení [HLY10]. Poslední vývoj spěl ke zdokonalení možností raytracingu na grafické kartě (graphical processing unit, GPU) [HG05]; tyto metody budeme také využívat.

Výpočet paprsku probíhá právě v pixel shaderu, úlohou programu je pak dodat grafické kartě počáteční a koncové body paprsku – toho se jednoduše docílí vykreslením předních i zadních stěn krychle a zakódováním pozic do barvy (viz např. [SK95]).

Často se zanedbává lom světla a šíření světla kamkoliv jinam než k pozorovateli. Nicméně některé práce se snaží stvořit i rychlé systémy pro tyto případy optických modelů pro lepší prezentace výsledků (jako například práce [vPV11]) a vizuální oddělení vyznačených segmentů [HBH03].

Pro zlepšení výsledného efektu obrazu se používá i osvětlení a stíny (viz též [Had04]), pro ty je potřeba definovat optické modely a normálu povrchu. Například pro nejvíce používaný Phongův model se intenzity barev vynásobí cosinem úhlu dopadajícího světla a povrchu.

V případě isoploch je jasné, co je normálou povrchu, nicméně pro klasické objemové zobrazování se ukazuje, že gradient intenzity poskytuje dobré výsledky. Rekonstrukce gradientu z diskrétních dat úzce souvisí s použitým filtrem. My použijeme přístup využívající derivací proloženého spline, viz [NVI99].

Pro lepší efekt (jako náhrada lomu světla) se dá používat i tzv. okolní (ambientní) světlo, popřípadě zvýšit vliv paprsků osvětlujících povrch pod úhlem, který se odrazí do kamery (spectacular).

Další zlepšování kvality a rychlosti

K tématu je k dispozici velké množství článků zaměřujících se na vylepšování rychlostí zobrazování a na zvýšení kvality obrazu (netriviálním předpočítáváním dat namísto pouhého snižování délky kroku).

Práce [KW03] a [YKEI06] popisují možná zrychlení založená na přeskakování míst (empty space skipping), kde integrál nasčítá nulu, a včasné zastavování výpočtu paprsků (early ray termination), kdy zastavíme výpočet, jakmile další pokračování neovlivní výslednou barvu. Zastavovacím kritériem je hodnota nasčítané alphy. R. Shen, P. Boulanger [SB07] prozkoumali vliv včasného zastavování paprsků a β-akcelerace. Možnou implementaci vyšší přesnosti a optimalizaci pro shadery představil tým S. Roettgera [RGW+03]. Článek o optimalizaci GPU kreslení [BR06] měří zrychlení pomocí rozkouskování celého objemu do malých částí a uložení do stromové struktury (často se tato metoda nazývá bricking, viz též [LMK03]).

A. Kull ve své disertaci [Ada12] řeší artefakty vznikající stejně dlouhými délkami kroků pomocí stochastického jitteringu.

Z. Nagy [NK03] použil ideu hloubkového ořezávání (depth peelingu) pro kreslení vícero isoploch najednou. Též existuje metoda opacity peeling [RSK06], která zobrazuje skryté isoplochy.

Krom zrychlování objemového zobrazování existují i práce zaměřující se ryze na isoplochy [XZC05].

1.2.1 Předintegrování

Klaus Engel popsal metodu předintegrování [EKE01] jako způsob, jak oddělit frekvence přechodové funkce od frekvencí v datech. Ta byla následně integrovaná do množství postupů (např. i do Shear warp algoritmu – viz [SKLE03]) i renderování nestrukturovaných sítí (viz dále).

Numericky se jedná o metodu, která si předpočítá všechny možné výsledky integrálu od počáteční intenzity A do koncové B, dle segmentu délky d, a pak při integraci složené funkce $f(S(x))$ počítá pouze s funkcí S intenzity původních dat. Jedná se tedy o předvypočítání všech hodnot integrálů barvy

$$\int_0^1 d \cdot C((1-\omega)\cdot A + \omega \cdot B) \cdot exp(-d \cdot \int_0^\omega \tau((1-\omega')\cdot A + \omega' \cdot B)\,\mathrm{d}\omega')\,\mathrm{d}\omega$$

a průhlednosti

$$1 - exp(-d \cdot \int_0^\omega \tau((1-\omega')\cdot A + \omega' \cdot B)\,\mathrm{d}\omega')$$

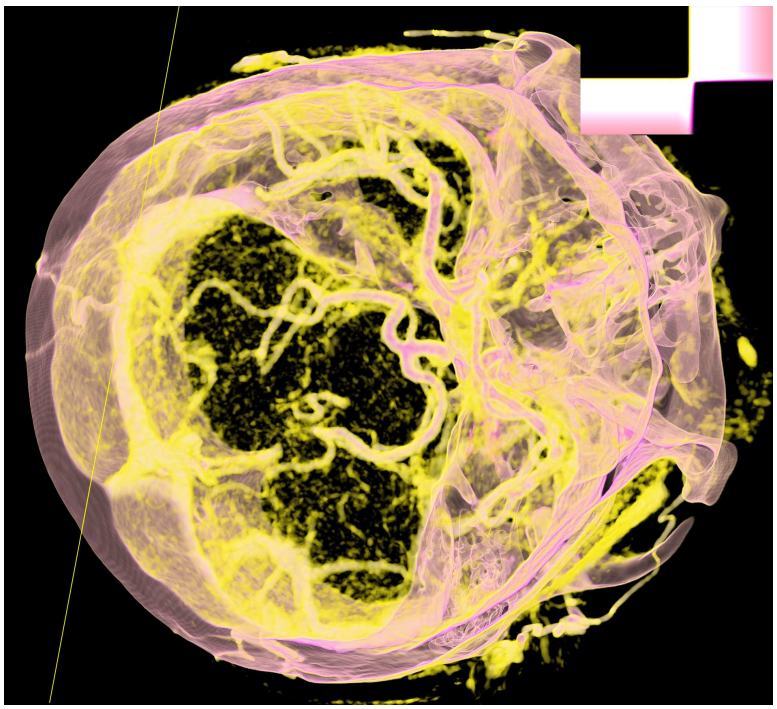
pro všechny parametry, tedy začátky a konce $A, B \in [0,1]$. Funkce $C(\omega)$ a $\tau(\omega)$ vrací barvu a průhlednost v bodě ω (jsou to přechodové funkce), parametry ω, ω' jsou proměnné, pomocí kterých se realizuje integrál z bodu A do bodu B.

Pro lepší představu – integrování bez předintegrace si lze představit jako klasickou Riemannovu metodu sčítání sloupečků na funkci $f(S(x))$ s preintegrací jako lichoběžníkové pravidlo na funkci $S(x)$. Další zvýšení řádu přesnosti probíhá přesně jako Simpsonovo pravidlo, a to s použitím trojrozměrné textury – viz

práce [EHMDM08]. M. Kraus dále prezentuje preintegraci pro vícedimenzionální přechodové funkce [Kra08].

Preintegrace dává dobré výsledky (viz [Had04]). Zvolíme ji jako stěžejní metodu do další kapitoly také proto, že se používá společně i pro některé algoritmy kreslení nestrukturovaných sítí, kde je zapotřebí trojrozměrná tabulka, protože se musí uvažovat i měnící se délka segmentu. Dále je zde pak prostor pro optimalizace uložení takové trojrozměrné tabulky. Práce [RE02] uvažuje i možnosti generování preintegrační tabulky na GPU.

Preintegrace využívá $O(n^2)$ paměti a potřebuje generovat tabulku vždy při změně délky segmentu – generování probíhá v čase $O(n^3)$.



Obrázek 1.2: Ilustrace objemového zobrazení s příslušnou preintegrační texturou. Na preintegrační texturu (vpravo nahoře) se mapuje pouze úzký interval intenzit z původních dat, tím je docíleno obarvení pouze některých částí cév a tkání. Ostré přechody v preintegrační textuře způsobí dobré oddělení částí tkáně.

1.2.2 Inkrementální předintegrování

Inkrementální předintegrování z [LWM04] je metoda, která pomocí technik dynamického programování vylepšuje čas na $O(n^2)$.

V naší implementaci zvolíme právě algoritmus efektivního výpočtu a upravíme jej pro naše potřeby (např. přechodové funkce máme zadané jinak než tabulkou hodnot). Výsledný kód je součástí příloh.

Hodnoty na diagonále (kde se počáteční intenzita rovná koncové: $A = B$)

umíme spočítat analyticky – integrujeme konstanty.

$$\int_0^1 d \cdot C(A) \cdot exp(-d \cdot \int_0^\omega \tau(A)\,\mathrm{d}\omega')\,\mathrm{d}\omega = d \cdot C(A) \int_0^1 exp(-d \cdot \omega\tau(A))\,\mathrm{d}\omega$$
$$= d \cdot C(A)(1 - exp(-d\tau(A)))$$

(Zapsáno integrálem po lineární cestě.)

Dostáváme barvu pro diagonálu a člen $1 - exp(-d\tau(A))$ je přímo vypočtená alpha. Zde také vidíme důvod, proč se často v různých aplikacích dá alpha aproximovat jako $1 - exp(-\tau(A))$, případně rovnou $\tau(A)$, pokud nechceme počítat exponenciální funkci (například v shaderu).

Popíšeme vyplnění poloviny preintegrační tabulky (druhá se vyplní analogicky).

První myšlenka je založená na dynamickém programování a následující úvaze: ukazuje se, že chceme-li znát výsledek integrálu od A do C, lze jej dostat z výsledků integrálů $A - B$ a $B - C$, jak ukazují následující úpravy. Důležité bude, že nezávisí na pořadí skládání. (A, B, C leží v přímce, všechny funkce jsou reálné kladné s hodnotami menšími než 1.)

$$\int_A^C d \cdot C(\omega) \cdot exp(-d \cdot \int_A^\omega \tau(\omega')\,\mathrm{d}\omega')\,\mathrm{d}\omega =$$

$$= \int_A^B d/2 \cdot C(\omega) \cdot exp(-d/2 \cdot \int_A^\omega \tau(\omega')\,\mathrm{d}\omega')\,\mathrm{d}\omega + \int_B^C d/2 \cdot C(\omega) \cdot exp(-d/2 \cdot \int_A^\omega \tau(\omega')\,\mathrm{d}\omega')\,\mathrm{d}\omega =$$

$$= \int_A^B d/2 \cdot C(\omega) \cdot exp(-d/2 \cdot \int_A^\omega \tau(\omega')\,\mathrm{d}\omega')\,\mathrm{d}\omega$$

$$+ exp(-d/2 \cdot \int_A^B \tau(\omega')\,\mathrm{d}\omega') \int_B^C d/2 \cdot C(\omega) \cdot exp(-d/2 \cdot \int_B^\omega \tau(\omega')\,\mathrm{d}\omega')\,\mathrm{d}\omega$$

(Zapsáno křivkovým integrálem.)

Lepší náhled do úprav může přinést i popis z fyzikálního pohledu modelu. Integrál má sčítat vliv paprsků ze světélkující mlhy jdoucích do jednoho bodu. Na začátku mají přispívající paprsky plnou intenzitu a při pronikání dále do mlhy snižuje mlha jejich příspěvek do celkové intenzity (což je v rovnici vyjádřeno druhým integrálem). Tedy příspěvek od A do C lze počítat jako příspěvek od A do B plus příspěvek od B do C, ovšem zamlžený o naintegrovaný útlum od A do B.

Toto platí i obecně pro k sčítanců a kousíčky délky d/k (v rovnici výše bylo $k = 2$). Protože je potřeba modifikovat dělitele d v exponenciále, nemůžeme dostat jednoduchý rekurentní vzorec, a tak musíme při vyplňování preintegrační tabulky použít jiný princip, založený na výpočtu kousíčků d/k pro celky velikosti k.

V každém kroku naintegrujeme kousíček $x \to x+1$ se všemi délkami d_x rovno $d/2...d/(n-1)$, ty budeme ukládat do přihrádek. Jakmile je v nějaké přihrádce číslo k uloženo alespoň $2k$ hodnot, můžeme díky triku skládání barvy a útlumu výše (subrange integration) vyplnit k hodnot do výsledné tabulky. S výjimkou

pro přihrádky délky 1 a n, které stačí reprezentovat jedním číslem, protože není potřeba nic skládat. Na konci ještě dovyplníme hodnoty z přihrádek, ve kterých ještě něco zbývá. Celková paměť na přihrádky pak bude $1+1+\sum_{d=2}^{n-1}\min(n,2d)$.

Jak v konkrétním kroku x skládat barvu a útlum? Aktualizujeme hodnoty pro sloupce $1...x$ takto:

$$C_x = C_{x-1} + \exp(-A_{x-1}) \cdot Clr$$

$$T_x = T_{x-1} + T$$

Alpha v preintegrační textuře bude $\exp(-A_x)$. (C_x je nová barva, C_{x-1} je stará barva, Clr je právě spočítaná barva dílku, T_x je nový útlum, T_{x-1} je starý útlum, T je právě naintegrovaný útlum. Stará barva a starý útlum pro sloupec x jsou hodnoty z diagonály.)

Celková práce na vyplnění tabulky je $O(n^2)$ – pro každý pixel děláme práci pouze jedenkrát a nevracíme se ke starým pixelům – plus práce na integraci n dílků $x \to x+1$. Spotřebovaná paměť algoritmu je $O(n)$ – pamatujeme si pouze hodnoty minulého sloupce – plus paměť na integraci jednoho dílku (navíc samozřejmě máme v paměti celou tabulku $O(n^2)$). V naší implementaci zvolíme C a T jako hodnoty ve středech intervalů.

Co se týče vyplňování trojrozměrné preintegrační tabulky (třetí rozměr pro všechny uvažované vzdálenosti v), kterou budeme také potřebovat, lze samozřejmě volat vícekrát funkce vyplňující dvourozměrnou tabulku, popřípadě použít heuristické zrychlení.

1.3 Zobrazování nestrukturovaných sítí

Nestrukturovaná síť je složená z buněk (cells), které sdílí stěny, vrcholy a hrany. Nemají, na rozdíl od strukturovaných sítí, pevně dané pravidelné rozmístění uzlových bodů (typicky se čtyřstěny hromadí v místech, kde je potřeba zobrazit vyšší detaily).

Data v nestrukturovaných sítích úzce souvisí s metodou konečných prvků, protože často představují výsledek nějaké simulace. Mohou být uložena jako hodnoty ve vrcholech (pokud uvažujeme lineární interpolaci), případně středech stěn nebo středech hran (pro vyšší stupeň přesnosti).

Nejjednodušší sítí je síť složená z čtyřstěnů (tetrahedra) s daty pouze ve vrcholech. Budeme k zobrazování tohoto typu sítě směřovat a ostatní metody jen zmíníme.

Zkoumána jsou též data proměnná v čase, pro která platí totéž; je však nutno postupovat po jednotlivých snímcích. Dále pak data složená z šestistěnů, pro která navíc existují výsledky, zkoumající možnosti jejich převodu na šestistěny [CMS06].

Pro vykreslování se přebíraly postupy ze strukturovaných sítí a vznikaly nové problémy a algoritmy (viz též [SCCB05]). Pojd'me se podívat na možné postupy kreslení:

- Chytřejší převedení na strukturované sítě – v práci [SS06] je ukázáno, jak lze podle potřeby měnit úroveň detailu.

- Kreslení pomocí řezů nad nestrukturovanými daty – [CM].

- Sledování paprsků (raytracing). Metoda raytracingu má stejné charakteristiky jako u strukturovaných sítí. Navíc je zde nutno různými způsoby vyřešit, do jakého dalšího konečného prvku paprsek pokračuje, viz H. Berk [BAG03]. Dále A. Maximo ukázal, jak lze i raytracing provádět akcelerovaně s nižší paměťovou náročností [MRB+08].

- Projekce buněk (cell projection). Cell projection řeší vykreslování jednotlivých prvků na obrazovku a barvu pixelů, které zabírají. Pokud si vybereme tento algoritmus, musíme ještě zkoumat, jak prvky správně setřídit (viz dále). V průběhu vývoje se tento algoritmus používal nejvíce kvůli intuitivnější možnosti implementace na GPU a je základem novějších metod, které se snaží řešit třídění jinak.

- Stochastické a bodové (point based) metody. Zde odpadá nutnost třídění, protože výsledná barva pixelu se určuje pomocí stochastických metod, viz např. [SKKK09]. Používají se často pro data s vyšším řádem přesnosti než lineárním, protože výpočet přesné barvy by byl náročný, a tak se barva paprsku procházejícího přes prvek aproximuje nějakým dobře počítaným průměrem. Příkladem je též [CSK03] a bodové objemové zobrazování z [ZG06], které zobrazuje výsledek jako shluk velkého množství bodů.

- Speciální a hybridní metody. Existují i další metody obsahující myšlenky z vícero typů, například algoritmus HAVS [CICS05] využívá třídění dat až v rámci jednotlivých pixelů (pomocí nové struktury k-bufferu) a byl dále vylepšen v [JH12]. Nejnovější metodou je práce M. Hadwigera řešící optimalizaci paměti s použitím nových datových struktur (TSFL – two sided face lists) pro zobrazování prvků s nelineárními stěnami a s využitím geometry shaderů [MHDG11].

Cell projection a třídění

Naivní přístup k třídění je prosté řazení středů prvků dle jejich hloubky. Pro přesné setřídění vyvinul L. Williams přesnou metodu MPVO, popsanou v [Wil92], která ovšem funguje pouze pro konvexní souvislé sítě, a navrhl také aproximativní metodu MPVONC, která dává přibližné setřídění v případě vícero komponent nesouvislé nebo nekonvexní sítě. Alternativou je algoritmus na konvexifikaci sítí od Roettgera [RGSE04], který ovšem přidává prvky navíc.

T. Silva popsal metodu pohybující se roviny ZSWEEP [SMK96] a XMPVO [SMW98], která byla následně vylepšena [CKM+99] o třídění pomocí BSP stromů.

Ukázalo se, že chyba v důsledku nedokonalého setřídění je při použití algoritmů malá (konkrétně rozdíl barev pixelů při třídění podle středů a použitím MPVONC, v [MMF10] viz dále).

Postupy renderování samotných čtyřstěnů se vyvíjely společně s hardwarovými možnostmi. Základem je PT (projected tetrahedra) algoritmus, který pro daný pohled na scénu určí pro každý prvek, jak vypadá jeho projekce na obrazovku, a tu popíše pomocí trojúhelníků. O typu projekce se rozhoduje dle znaménka čtyř vektorových součinů. Ovšem díky rozdílnému počtu trojúhelníků, potřebných k popisu jednotlivých případů, nelze vykreslení čtyřstěnu zpracovat zcela

standardně, jak vyžadují grafické ovladače. S tímto problémem se lze vypořádat přímým výpočtem každého prvku zvlášť, nebo využít vlastností grafických karet a vždy posílat maximální počet trojúhelníků ve formě základního grafu, z nichž některé lze ze zpracování vyřadit jejich degenerováním, jak popisují práce [BpB], [WMFC02].

Algoritmus IPTINT [MMFE08], [MMo06] si všímá faktu, že stačí znát projektivní typ (a eventuelně průsečík) a že tyto výpočty lze provést paralelizovaně pomocí shaderu na grafické kartě. Největšího výkonu pak dosáhla metoda HAPT [MMF10], která prozkoumala třídění i na grafické kartě pomocí platformy CUDA a použila geometry shadery, které umožňují generovat data triangulací přímo v grafické kartě, čímž elegantně vyřešila problém různého počtu trojúhelníků.

Zkoumají se též metody vylepšující kvalitu obrazu. S. Guthe, S. Roettger [GRS+02] používají metodu preintegrace (která je uzpůsobena pro projekci buněk) s využitím 3D textur (kde se do třetího rozměru uloží možné délky intervalu) a vypořádává se s možnými artefakty. Následně bylo prozkoumáno v [KQE04], že uložení dat preintegrační tabulky v logaritmickém smyslu – vzhledem k délkám intervalu – produkuje lepší výsledky a šetří paměť. Kvalita obrazu také vzroste po aplikování správné perspektivní korekce – zbavíme se tak možných viditelných artefaktů.

Kapitola 2

Vybrané metody

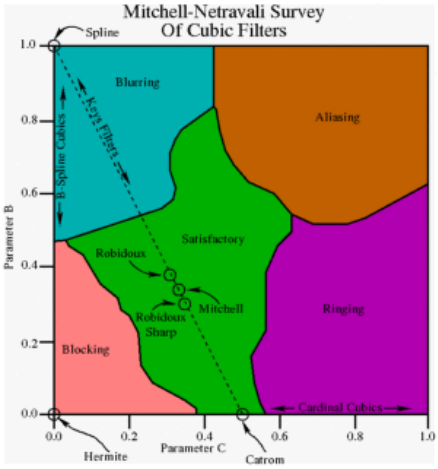
V této kapitole se budeme při popisu vlastních přínosů práce a řešení odkazovat na znalosti počítačové grafiky a grafických rozhraní, včetně pojmů zmíněných v předchozí kapitole. Navržená vylepšení a algoritmy jsou na konkrétním rozhraní nezávislá.

2.1 Strukturované sítě

Dílčí řešení

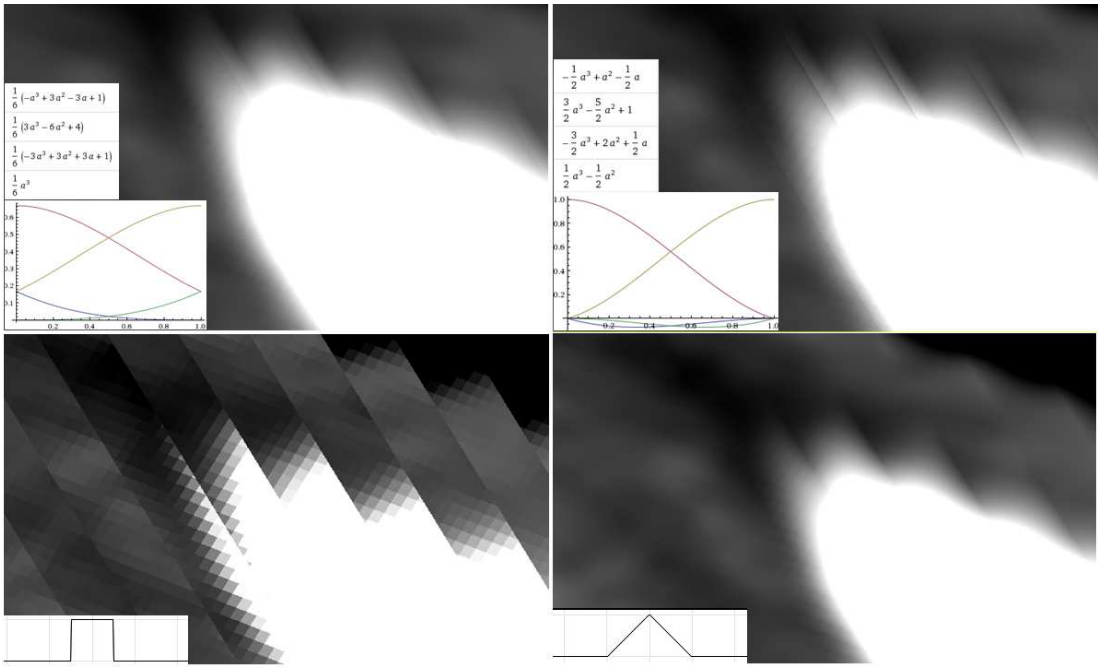
Při tvorbě vhodného rekonstrukčního filtru použijme implementaci na grafické kartě s využitím pixel shaderů. Vhodné koeficienty volíme také dle práce [MN88].

V jednorozměrném případě interpolace nad daty se jakožto zdrojové uzly berou 4 zdrojové hodnoty jasu (rovnoměrně vzdálené o jednotku vzdálenosti zavedenou jako jeden pixel). Přičemž bod x, ve kterém chceme znát jasovou hodnotu, leží mezi vnitřními dvěma. Ke čtyřem uzlům má postupně kladné vzdálenosti: $\alpha+1, \alpha, 1-\alpha, 2-\alpha$, kde $\alpha = (x - \lfloor x \rfloor)$ je vzdálenost k nejbližšímu uzlu vlevo.



Obrázek 2.1: Vlastnosti rekonstrukčních filtrů.

Interpolace nad více rozměry se dostane rekurzivním postupem – nejprve se spočítají interpolované hodnoty na všech přímkách (v jednom zvoleném směru) a ty se vezmou jakožto zdrojová data pro další interpolaci (v kolmém směru na původní zvolený směr). (Ve 2D případě jsme právě hotovi, protože máme hodnotu pro plochu 4×4 pixelů. Ve 3D máme 4 interpolované hodnoty na 4 plochách a provede se ještě jedna iterace.)



Obrázek 2.2: Výsledky aplikace různých filtrů.

Zleva: BSpline, Catmulli-rom, původní data, lineární.

Pro váhové funkce se použije filtr "Mitchell-Netravali", viz [MN88], s váhovou funkcí:

$$(-B-6C)x^3+(6B+30C)x^2+(-12b-48C)x+(8B+24C), 1 \leq |x| < 2$$
$$(12-9B-6C)x^3+(-18+12B+6C)x^2+(6-2B), 0 \leq |x| < 1$$

Analogickým postupem jako v [NVI99] lze upravit pro použití na grafické kartě celou tuto třídu kubických polynomů, a to dosazením $\alpha+1, \alpha, 1-\alpha, 2-\alpha$ za x. Tedy využitím faktu, že hardware sám o sobě umí používat lineární interpolaci, kterou můžeme využít pro snížení potřebného počtu uzlových bodů z 4^d na 2^d (d je počet dimenzí), jsme odvodili potřebné polynomiální funkce a postup pro trojrozměrné textury a vztahy pro výpočet gradientu. Navíc, dalším využitím schopnosti provádět vektorové operace rychleji, lze optimalizovat filtr pro rychlejší interpolaci zdrojových dat intenzit (pak lze v shaderu použitím rychlých vektorových operací interpolovat data čtyřikrát rychleji).

Porovnání výsledků různých filtrů na objemovém řezu (MPR), spolu s naznačenými váhovými funkcemi, na obrázku 2.2.

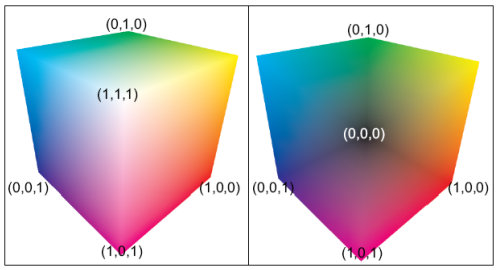
Toto filtrování použijeme i pro jiné módy objemového zobrazování.

Zbývá dodat, že pro aplikaci jsme zvolili předintegrování prvního řádu jednorozměrných přechodových funkcí, protože do třetího rozměru textury může být vhodné uložit další přechodové funkce pro segmentaci (a následně i konečné prvky potřebují třetí rozměr pro délku segmentu). Nakonec pro uživatelské zadávání přechodových funkcí volíme po částech lineární nebo kubický spline s implementací dle knihy [PTVF92].

2.1.1 Trasování paprsků

Počáteční a koncové body trasování

Pro trasování paprsku je shaderu zapotřebí zadat počáteční a koncové body pro každý pixel (formou textury). Protože mají data vždy podobu kvádru, vykreslíme do první textury stěny kvádru blíže ke kameře a do druhé textury stěny kvádru dále od kamery. Barvy vrcholů krychle pak určují texturové souřadnice, které se vyinterpolují na grafické kartě přímo do dat jednotlivých pixelů (viz obr. 2.3).



Obrázek 2.3: Počáteční a koncové body.
Zdroj [Tri].

Jak vyřešíme pixely, které budou oříznuté? Pohled na scénu má totiž blízkou a vzdálenou ořezávací rovinu, ty vyřazují ze zpracování všechny pixely mimo pohled (zůstanou pouze pixely přesně mezi těmito rovinami). Tradiční řešení uvažuje vypočtení geometrického modelu ořezané krychle, kterou pak pošle do grafického zpracování. My ale využijeme možnosti ořezávání v shaderu.

Nejprve vykreslíme celou přední ořezávací rovinu (s patřičně – inverzně – transformovanými texturovými souřadnicemi) a necháme projít testem pouze takové pixely, které skutečně leží uvnitř krychle – tedy vykreslíme přesně takovou oblast, která bude oříznutá. Pak vykreslíme celou krychli, která se automaticky ořízne. Dohromady tak v první textuře dostaneme požadované počáteční pozice bez počítání průsečíků (ty za nás spočítá grafická karta po pixelech).

Stejný postup aplikujeme pro zadní stěny a zadní ořezávací rovinu. A právě sem přidáme ještě jedno zrychlení – tradičně se rozhoduje shader až při trasování paprsku, které body patří krychli – na kterých trasovat – a které jsou mimo (na obrázku černé). Celé kreslení bychom ovšem mohli zrychlit, pokud by vůbec nedošlo ke spuštění programu a pixely mimo krychli by byly vyřazeny ze zpracování ještě dříve. Jak toho dosáhnout? Přidáme do celého problému z-buffer, a to následovně.

Při kreslení souřadnic do druhé textury nejprve resetujeme z-buffer na hodnotu 0 a nastavíme, aby prošly vzdálenější pixely. Jako předtím pak vykreslíme průnik celé zadní ořezávací roviny (hloubka maximální) s krychlí a také zadní stěny krychle (hloubka kladná). V z-bufferu jsou tedy hodnoty kladné tam, kde chceme trasovat paprsek, a nulové pro paprsky mimo krychli.

Následně stačí nastavit porovnávání tak, aby prošly pouze hodnoty z ostře větší, a spustit shader na přední ořezávací rovině s nulovou hloubkou, kde z-test vyřadí ze zpracování právě pixely s nulovou hloubkou.

Průchod paprsku

Paprskem postupujeme odpředu dozadu, abychom mohli využít kritéria včasného zastavení. Trasovat začneme od hodnot uvedených na vstupu z předchozího procesu.

V každém bodě přečteme hodnotu intenzity a gradientu, minulou a součas-

nou intenzitu použijeme jako index do předpočítané tabulky předintegrovaných hodnot. Na přečtenou barvu aplikujeme osvětlovací model (s využitím gradientu jako normály povrchu) a složíme s napočítanou hodnotou následovně:

```
Colored.rgb=Colored.rgb*(dot(Frag.xyz,Light.xyz)+Light.w);
Accum.rgb+=Accum.a*Colored.rgb;
Accum.a*=saturate(1.0f-Colored.a);
```

Jakmile akumulovaná hodnota v alfa kanálu klesne pod jistou hranici (volíme např. 0.01), víme, že můžeme trasování zastavit, protože každý další příspěvek barvy bude velmi malý a alfa se již nikdy nad tuto hranici nedostane (v průběhu trasování je nerostoucí). Následně se posuneme o další pevně zvolený krok a pokračujeme dále, dokud neprojdeme maximální délku, nebo se nezastavíme.

Protože ale využíváme shader model 3, je množství provedených instrukcí limitované – pokud bychom snižovali délku kroku, nemuseli bychom projít celou trasu. Proto je potřeba přidat zapamatování si bodu, do kterého jsme dospěli. To uděláme pomocí druhého výstupu (render target), do kterého uložíme pozici. V dalším běhu pak stačí poslat místo původní textury obsahující počáteční body tuto vygenerovanou. Včasné zastavování se pak dá implementovat uložením koncové hodnoty, nebo zápisem nuly do z-bufferu. Ke skládání hodnot lze využít přímo operace průhlednosti (alfa blendingu). Abychom se vyhnuli problémům s možným 8bitovým primárním výstupem a mohli stále využívat včasné zastavování paprsku, nastavíme blending na pouhé sčítání a alfa hodnotu si budeme pamatovat spolu s pozicí v druhém výstupu.

Pro odstranění artefaktů (jako na obrázku 1.1) se používá tzv. hloubkové rozmazání (depth jittering) – tedy při prvním čtení z textury přičíst malé změny v hloubce. Stochastický šum generujeme s použitím doporučeného vzorce:

```
float random = sin(I.ProjPo.x * 12.9898 + I.ProjPo.y * 78.233);
random =  frac(random * 43758.5453);
Front.xyz = Front.xyz + RayDiff * random;
```

Kód takového shaderu pro objemové zobrazování s depth jitteringem, inkrementální preintegrací a vyhlazováním B-spline s rekonstrukcí gradientů je součástí elektronických příloh.

2.2 Nestrukturované sítě

Naše úvahy pro výběr algoritmu se opřou o následující fakta z předchozí kapitoly. Pro data vyšší přesnosti než lineární je k dispozici buď podrozdělování, nebo aproximativní obarvování pomocí heuristik, nebo vykreslování mračna bodů. Naproti tomu, pro data lineárně interpolovaná jsou k dispozici přesné výsledky díky preintegraci.

Pro obecné sítě je nejnovější algoritmus z [MHDG11]. Pro čtyřstěny s lineárními hodnotami je nejlepší používat metodu HAPT, pokud bychom chtěli uvažovat geometry shadery.

Protože v současné době máme k dispozici čtyřstěnná data s lineárními hodnotami a požadujeme pro začátek implementaci bez geometry shaderů, vybereme metodu IPTINT, která je pro tento případ efektivnější.

Řazení použijeme stejné jako v originálním zdroji a pro testovací účely porovnáme i s MPVONC (implementací z HAPT). Do algoritmu přidáme perspektivní korekci zmíněnou v [KQE04].

Algoritmus

Kreslení probíhá ve třech fázích.

Při každé změně pohledu na scénu je potřeba přepočítat projektivní typy jednotlivých čtyřstěnů. To provedeme v první fázi paralelně výpočtem na grafické kartě tak, že data vrcholů uložíme do jedné textury, do které budeme indexovat z textury druhé obsahující data čtyřstěnů. V shaderu pak pro každý čtyřstěn načteme data jeho vrcholů, spočítáme projektivní typ (tj. číslo jednoznačně určující tvar čtyřstěnu po promítnutí na projekční rovinu), tloušťku a souřadnice centrálního vrcholu (thick vertex). Centrální vrchol může být jedním z vrcholů čtyřstěnu, nebo také průsečíkem dvou hran.

V druhé fázi vygenerovaná data přečteme, setřídíme na CPU (postupy prezentovanými v [MMo06]) a aktualizujeme paměťové struktury pro projektované prvky. Každý čtyřstěn je po proběhnutí této fáze uložen přesně ve tvaru své projekce na projekční rovinu, a to jako shluk trojúhelníků kolem centrálního vrcholu. Centrální vrchol má jako jediný nenulovou tloušťku a kolem něj je právě tři až pět trojúhelníků (záleží na projektivním typu při daném pohledu).

Geometrie se pak pošle ke zpracování a vykreslení do poslední fáze, všechny vrcholy se transformují a interpolují se lineárně jejich vlastnosti důležité pro určení barvy v daném bodě. Těmito vlastnostmi jsou intenzita/barva bodu, ve kterém paprsek vchází do čtyřstěnu, intenzita/barva bodu, ve kterém paprsek čtyřstěn opouští, a délka trajektorie, kterou v objektu urazí. Shader na základě těchto vlastností jednoznačně určí příspěvek barvy daného čtyřstěnu do paprsku, a to právě podle dat uložených v trojrozměrné preintegrační textuře. Ta byla generovaná tak, aby obsahovala všechny výsledky výpočtu výsledné barvy pro paprsek dané délky začínající a končící v daných intenzitách.

Příspěvky jednotlivých čtyřstěnů se skládají operací blendingu v paměti grafické karty. Tato blending operace přesně simuluje útlum procházejícího paprsku. Z toho důvodu bylo také nutné data setřídit, aby se operace provedly v korektním pořadí. Je vidět, že první dvě fáze algoritmu se provádějí pouze při změně pohledu na scénu.

Další vylepšení algoritmu, které provedeme, byla zmíněná úprava pro projektivně korektní interpolaci veličin. Zmíněná lineární interpolace je samozřejmě správné řešení, lineárně byla data spočítaná metodou konečných prvků, ovšem lineární interpolaci je potřeba provádět v koordinátovém systému dané buňky a nikoliv v koordinátech projektivní roviny. Zmíněná projektivně korektní lineární interpolace popisuje, jaké vzorce použít, abychom mohli počítat v koordinátech projektivní roviny (díky architektuře grafických rozhraní nelze v tomto kroku počítat v souřadném systému buňky).

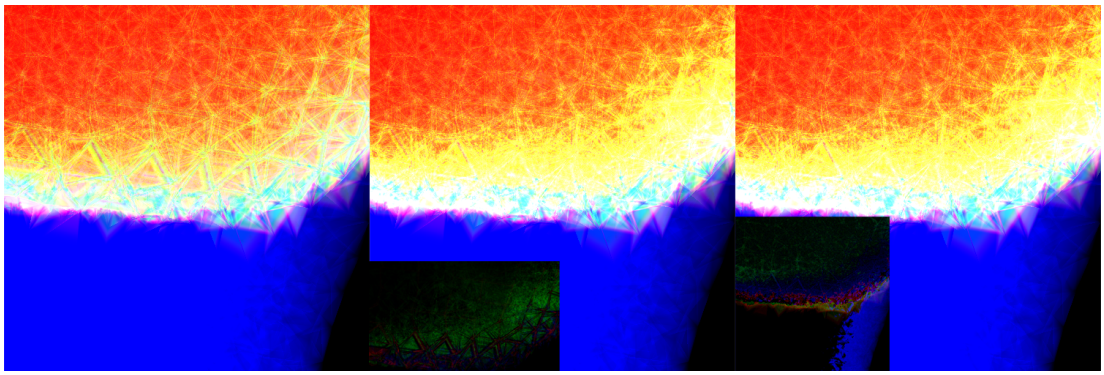
Filtrování a preintegrace

Odpovězme si zde na otázku, jaký vliv má filtrování preintegrační textury a výběr paměti pro operaci blendingu na kvalitu obrazu.

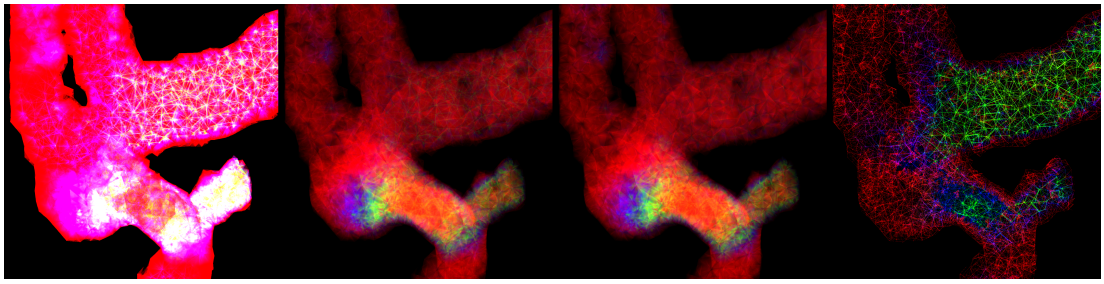
V programu ukládáme preintegrační texturu v hloubkovém rozměru logaritmicky (tedy pro parametry d, $2d$, $3d$, ... pro dvojkový základ) a parametr hloubky pro hodnoty pod spočtený rozsah dopočítáváme lineárně k nule přímo v shaderu. Velký vliv na kvalitu zobrazování má přesnost použité paměti v grafické kartě.

Co se týče filtrování preintegrační textury, nebylo zatím nikde zkoumáno, jaký vliv má na kvalitu obrazu. Při porovnávání výsledků dává kubická interpolace pozvolnější změny a plnější jas v oblastech s měnící se tloušťkou objemu. Ovšem nedosahuje takového zvýšení kvality, aby bylo možné zmenšit preintegrační texturu a podstatně uspořit paměť.

Odpověď poskytuje obrázek 2.4, který ukazuje vliv porovnání kubického filtrování a formátu paměti. Obrázek 2.5 ukazuje vliv různých základů logaritmu na kvalitu vykreslení.



Obrázek 2.4: Porovnání přechodu v datech torza.
Zleva: 8bitová paměť, 32bitová float a 32bitová float paměť s trikubickým filtrováním. U druhého a třetího obrázku zobrazené zmenšeně i diference oproti předchozímu. Všimněme si, že filtrování se projevuje nejvíce na okraji objemu (modrá oblast vpravo) a že 8bitové textury drasticky omezí kvalitu pro průhlednější a tlustší objemy jako je tento. Obrázek je též přiložen jako elektronická příloha.



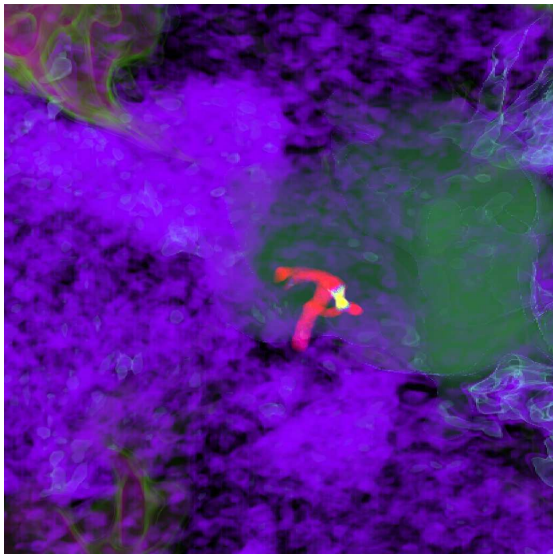
Obrázek 2.5: Vliv různých hloubek na kvalitu obrazu.
Hloubky Zleva: 8, 14, 16, 24 (odpovídají základům logaritmů 4, 2.3, 2, 1.6). Je vidět, že snížení hloubky má velký vliv na zhoršení kvality, nicméně zvýšení se projeví jen velmi málo. Poslední obrázek byl na pohled totožný s předchozím, takže je zde znázorněný pouze jejich rozdíl, dvacetkrát zesílený.

2.3 Fúze

Pro praktické účely je potřeba zobrazovat napočítané hodnoty (např. tlaky proudění v cévě) v kontextu originálních dat. Primárním cílem není realistické kreslení, ale srozumitelnost. Při realistickém kreslení by se musel při trasování paprsků vzít v úvahu průchod oběma typy prostředí (napočítanými hodnotami a originálními hodnotami), což by bylo pomalé a navíc komplikované dvěma typy sítí.

Umožníme tedy několik stylů fúze obrázků – pouhé zobrazení přes sebe a výřezové zobrazení.

Díky výše zmíněnému návrhu DVR můžeme totiž při kreslení přední a zadní části krychle (která určuje počáteční a koncové body raytracingu) využít vlastnost z-bufferu a vykreslit další ořezávací geometrii. Jako důsledek můžeme určovat počáteční a koncové body raytracingu. Tedy například vykreslením nejvzdálenějších bodů modelu cévy do předního bufferu zařídíme, že se bude kreslit objem až za fyzickou pozicí cévy. Následným překreslením dat cévy vznikne obrázek, kde napočtená data nejsou ničím překryta a jsou vidět v celém kontextu modelu.



Obrázek 2.6: Simultánní rendering cévy a jejího okolí.
Céva je obarvená dle amplitudy rychlostí proudění tak, že světlá oblast v cévě těsně pod malformací značí nejrychlejší proudění. Fialová barva ukazuje jemnější tkáně v mozku. Přechodová funkce je nastavená tak, že pohled na vnitřní strukturu mozku není rušen lebkou, tedy není nutné aplikovat ořezávání.

Alternativou je vyříznout kruh (opět technikou popsanou u zadávání počátečních a koncových bodů) s větším poloměrem okolo cévy, a tak zdůraznit, že se jedná o hloubkový výřez dat.

Na ukázku fúzního zobrazení, dávajícího do kontextu cévu a její okolí, se podívejme na obr. 2.6.

2.4 Porovnání výsledků

Pojďme se podívat na časy potřebné ke generování dat a odpovědět si na otázku, jestli jsme našli použitelné řešení.

Generování trojrozměrné preintegrační tabulky o rozměrech $1024 \times 1024 \times 16$ zabralo dle očekávání v průměru 10 sekund. Naštěstí přechodovou funkci na generovaných datech nepotřebuje uživatel měnit často, takže takovýto výpočet je potřeba provést jen jedenkrát, při startu programu. Aktualizace přechodové funkce pro pevně danou vzdálenost (tedy generování 2D tabulky) trvá pouhých 0.62 sekundy, tedy je možné provádět za běhu změny takovéto přechodové funkce a změny délky kroku.

Časy renderování samotné strukturované sítě, změřené pouze pro dobu běhu shaderu, se pro vygenerované obrázky pohybovaly od 0.04 ms pro cílovou velikost 1024×1024 s délkou kroku 0.003, do 0.3 ms pro velikost 2048×2048 s délkou kroku 0.002. Samotná rychlost zobrazování lékařských dat je tedy dostačující pro běžnou práci, což si lze vyzkoušet i s pomocí přiložené aplikace.

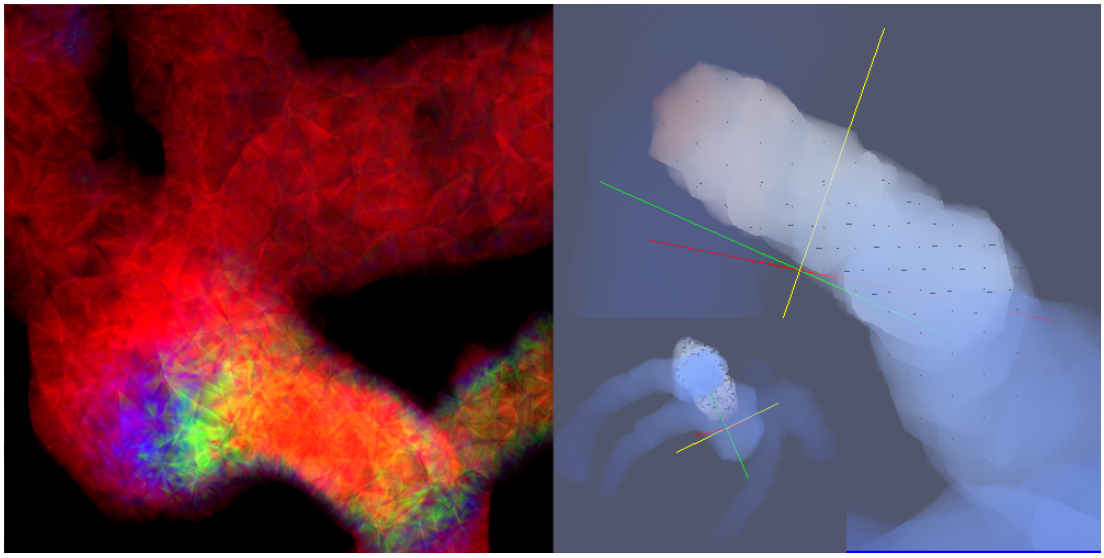
Pro náhled na výkon nad nestrukturovanými daty se podívejme do tabulky 2.1.

Software paraview uměl až donedávna vykreslovat sítě pouze bez hardwarové akcelerace, v nové verzi byl přidán algoritmus HAPT, který též umožňuje zobrazování v reálném čase. Pro porovnání kvality přikládáme vykreslené obrázky 2.7. Protože implementace v paraview používá stochastické obarvování, jsou při jistých úhlech pohledu a oddáleních patrné artefakty v jednotlivých pixelech. Stochastický přístup také neukazuje strukturu jednotlivých čtyřstěnů, což je žádaná vlastnost, protože lze díky ní odhadovat, s jakou přesností lze výpočtu důvěřovat. Kdybychom od tohoto požadavku upustili, bylo by možné osvětlením dodat datům plastičnosti a potlačit tvar jednotlivých buněk.

	Počet buněk	Paralelizovaný PT	Třídění, nahrávání	Kreslení
fighter	70k	0.46 ms	117.9 ms	2.1 ms
céva	40k	0.27 ms	68.1 ms	1.8 ms
torzo	1M	0.54 ms	2333.9 ms	1.8 ms

Tabulka 2.1: Časy částí algoritmu IPTINT na různých zdrojových datech.

První dvě fáze algoritmu zabírají nejvíce času, jsou ovšem potřeba provádět jen při změně pohledu na scénu. Častým řešením tedy je zanedbat třídění při aktivní manipulaci s daty a provést jej až dodatečně. Řešení, která se nemusí omezovat požadavky na hardware, mohou také optimalizovat třídění dat.



Obrázek 2.7: Porovnání prezentovaného IPTINT (vlevo) s paraview (vpravo).

2.5 Závěr

Implementovali jsme hardwarově akcelerované objemové zobrazování strukturovaných a nestrukturovaných sítí na grafické kartě pomocí technik preintegrace, IPTINT, perspektivní korekce, depth jitteringu a uzpůsobených kubických filtrů. Nastínili jsme několik způsobů fúze obou typů obrázků a porovnali vliv na obrazovou kvalitu některých prezentovaných postupů a jejich kombinací.

Výsledky ukazují, že často se nemusí projevit všechna uvažovaná vylepšení, každé zlepšuje pouze data s určitou vlastností, nicméně mohou dobře fungovat najednou.

Časová a paměťová náročnost prezentovaných metod umožňují rychlé vykreslování a reakci na vstup uživatele. Pro vyšší kvality obrazu je pak kdykoliv možné v čekacích časech zmenšit délku kroku a zobrazovat kvalitnější výstup.

V přílohách lze nalézt zdrojové soubory aplikace a soubory použité ke generování obrázků v tomto textu. Aplikace nechť plní účel ryze technického dema. Návod k jejímu základnímu použití na generování obrázků je uveden v přílohách.

Dalšími cíli může být implementace do uživatelsky příjemné aplikace, implementace dalších volitelných možností zobrazování (jako jsou zobrazování isoploch, bricking, empty space skipping, geometry shadery – pro tetrahedra a nonplanar-faces, preintegrace pro multidimenzionální textury, preintegrace se světly, local ambient occlusion).

V neposlední řadě bude sepsaný seznam článků a metod používaných v objemovém zobrazování sloužit jako rozcestník a část programové dokumentace pro možný další vývoj aplikace.

Příloha A

Elektronické přílohy

Obsah příloh

Všechny přílohy lze najít jak na přiloženém disku u tištěné verze práce, tak na webové adrese `https://sites.google.com/site/effdvr`. Součástí práce jsou také odkazované zdrojové kódy, a to ve složce *src*, vygenerované obrázky ve velkém rozlišení ve složce *imgs*. Ke kompilaci byl použit nástroj Borland Developer Studio Turbo C++.

Za pozornost stojí zdrojové kódy shaderů ve složce *src\Images\Shaders*.

Ovládání

Jednou z příloh je nástroj WMPacsViewerDXscreen, kterým byly vygenerované obrázky v této práci a jehož zdrojové kódy jsou přiloženy. Tato aplikace má spíše podobu technického dema s možností vykreslit pomocí grafické karty různé pohledy na scénu složenou z jedné strukturované a jedné nestrukturované sítě a výsledek uložit do souboru. Všechny uvažované parametry popisuje konfigurační soubor *sampleinput.txt*.

Pro vyzkoušení zkopírujeme složku *program* na lokální disk (protože program tvoří obrázek a zapisuje naměřené časy) a spustíme příkazem

```
WMPacsViewerDXscreen.exe -r input.txt
```

Tak byly také vygenerované obrázky v této práci.

Seznam obrázků

Seznam tabulek

Literatura

[Ada12] Kull Adam. Ray tracing of volumetric data in real time. `http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2012/rapporter12/kull_adam_12064.pdf`, 2012. [Online; accessed 19-02-2013].

[AGS95] Minesh B. Amin, Ananth Grama, and Vineet Singh. Fast volume rendering using an efficient, scalable parallel formulation of the shear-warp algorithm. In *Proceedings of the IEEE symposium on Parallel rendering*, PRS '95, pages 7–14, New York, NY, USA, 1995. ACM.

[BAG03] Hakan Berk, Cevdet Aykanat, and Ugur Gudukbay. Direct volume rendering of unstructured grids. *Computer I& Graphics*, 27:387–406, 2003.

[BpB] SĂ©bastien Barbier and Georges pierre Bonneau. Gpu improvements on the sorting and projection of tetrahedral meshes for direct volume rendering.

[BR06] A. Vilanova Bartroli and D. Ruijters. Optimizing gpu volume rendering. In *WSCG - Winter School of Computer Graphics*, 2006.

[Bru08] Anders Brun. Image3. `http://www.mathworks.com/matlabcentral/fileexchange/21881-image3`, 2008. [Online; accessed 19-02-2013].

[CC08] Chang-Chieh Cheng and Yu-Tai Ching. Transfer function design for fourier volume rendering and implementation using gpu. pages 691806–691806–10, 2008.

[CICS05] S.P. Callahan, M. Ikits, J.L.D. Comba, and C.T. Silva. Hardware-assisted visibility sorting for unstructured volume rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 11(3):285–295, may-june 2005.

[CKM+99] JoĂŁo Comba, James T. Klosowsk, Nelson Max, Joseph S. B. Mitchell, ClĂˇudio T. Silva, and Peter L. Williams. Fast polyhedral cell sorting for interactive rendering of unstructured grids. *Computer Graphics Forum*, 18(3):369–376, 1999.

[CM] Prashant Chopra and Joerg Meyer. Incremental slicing revisited: Accelerated volume rendering of unstructured meshes. In *Proceedings of IASTED Visualization, Imaging, and Image Processing 2002, Ma'laga*, pages 533–538.

[CMS06] Hamish Carr, Torsten Moller, and Jack Snoeyink. Artifacts caused by simplicial subdivision. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):231–242, March 2006.

[CN94] Timothy J. Cullip and Ulrich Neumann. Accelerating volume reconstruction with 3d texture hardware. Technical report, Chapel Hill, NC, USA, 1994.

[CSK03] Balazs Csebfalvi and Laszlo Szirmay-Kalos. Monte carlo volume rendering. *Visualization Conference, IEEE*, 0:59, 2003.

[DKC+98] Frank Dachille, Kevin Kreeger, Baoquan Chen, Ingmar Bitter, and Arie Kaufman. High-quality volume rendering using texture mapping hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '98, pages 69–ff., New York, NY, USA, 1998. ACM.

[EHMDM08] J.-F. El Hajjar, S. Marchesin, J.-M. Dischler, and C. Mongenet. Second order pre-integrated volume rendering. In *Visualization Symposium, 2008. PacificVIS '08. IEEE Pacific*, pages 9 –16, march 2008.

[EKE01] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '01, pages 9–16, New York, NY, USA, 2001. ACM.

[GRS+02] Stefan Guthe, Stefan Roettger, Andreas Schieber, Wolfgang Strasser, and Thomas Ertl. High-quality unstructured volume rendering on the pc platform. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '02, pages 119–125, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[Gue05] Daniel Guellmar. Surface2volume. `http://www.mathworks.com/matlabcentral/fileexchange/8772-surface2volume`, 2005. [Online; accessed 19-02-2013].

[Had04] Markus Hadwiger. *High-Quality Visualization and Filtering of Textures and Segmented Volume Data on Consumer Graphics Hardware*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 2004.

[HBH03] Markus Hadwiger, Christoph Berger, and Helwig Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 40–, Washington, DC, USA, 2003. IEEE Computer Society.

[HG05] Yang Heng and Lixu Gu. Gpu-based volume rendering for medical image visualization. In *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, pages 5145 –5148, jan. 2005.

[HLY10] F. Hernell, P. Ljung, and A. Ynnerman. Local ambient occlusion in direct volume rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 16(4):548 –559, july-aug. 2010.

[JH12] Qianli Ma Sikun Li Jian Hu, Wenke Wang. Optimization and implementation of unstructured grid volume rendering algorithm. *Advances in Intelligent Systems Research*, 2012.

[KQE04] Martin Kraus, Wei Qiao, and David S. Ebert. Projecting tetrahedra without rendering artifacts. In *Proceedings of the conference on Visualization '04*, VIS '04, pages 27–34, Washington, DC, USA, 2004. IEEE Computer Society. [Online; accessed 19-02-2013].

[Kra08] M. Kraus. Pre-integrated volume rendering for multi-dimensional transfer functions. In *Proceedings of the Fifth Eurographics / IEEE VGTC conference on Point-Based Graphics*, SPBG'08, pages 97–104, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[Kro11] Dirk-Jan Kroon. Viewer3d. `http://www.mathworks.com/matlabcentral/fileexchange/21993-viewer3d`, 2011. [Online; accessed 19-02-2013].

[KW03] J. Kruger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 38–, Washington, DC, USA, 2003. IEEE Computer Society.

[Lev92] Marc Levoy. Volume rendering using the fourier projection-slice theorem. In *Also Stanford University*, pages 61–69, 1992.

[LL94] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 451–458, New York, NY, USA, 1994. ACM.

[LMK03] Wei Li, Klaus Mueller, and Arie Kaufman. Empty space skipping and occlusion clipping for texture-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 42–, Washington, DC, USA, 2003. IEEE Computer Society.

[Lud08] Eric Ludlam. Sliceomatic. `http://www.mathworks.com/matlabcentral/fileexchange/764`, 2008. [Online; accessed 19-02-2013].

[LWM04] Eric B. Lum, Brett Wilson, and Kwan-Liu Ma. High-quality lighting and efficient pre-integration for volume rendering. In *Proceedings of the Sixth Joint Eurographics - IEEE TCVG conference on Visualization*, VISSYM'04, pages 25–34, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[MHDG11] Philipp Muigg, Markus Hadwiger, Helmut Doleisch, and Meister Eduard Gröller. Interactive volume visualization of general polyhedral grids. *IEEE Transaction on Visualization and Computer Graphics*, 17(12):2115–2124, 12 2011.

[MJC02] Benjamin Mora, Jean Pierre Jessel, and René Caubet. A new object-order ray-casting algorithm. In *Proceedings of the conference on Visualization '02*, VIS '02, pages 203–210, Washington, DC, USA, 2002. IEEE Computer Society.

[MKB+03] Tom Mertens, Jan Kautz, Philippe Bekaert, Frank Van Reeth, and Hans-Peter Seidel. Efficient rendering of local subsurface scattering. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, PG '03, pages 51–, Washington, DC, USA, 2003. IEEE Computer Society.

[MMF10] A. Maximo, R. Marroquim, and R. Farias. Hardware-assisted projected tetrahedra. In *Proceedings of the 12th Eurographics / IEEE - VGTC conference on Visualization*, EuroVis'10, pages 903–912, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[MMFE08] R. Marroquim, A. Maximo, R. Farias, and C. EsperanĂ§a. Volume and isosurface rendering with gpu-accelerated cell projection. *Computer Graphics Forum*, 27(1):24–35, 2008.

[MMo06] AndrĂ© Maximo, Ricardo Marroquim, and Ricardo Farias (orientador. Gpu-based cell projection for interactive volume rendering. In *SIBGRAPI*, pages 147–154, 2006.

[MN88] Don P. Mitchell and Arun N. Netravali. Reconstruction filters in computer-graphics. *SIGGRAPH Comput. Graph.*, 22(4):221–228, June 1988.

[Mor04] K.D. Moreland. *Fast, High Accuracy Volume Rendering*. University of New Mexico, 2004. [Online; accessed 19-02-2013].

[MRB+08] A. Maximo, S. Ribeiro, C. Bentes, A. Oliveira, and R. Farias. Memory efficient gpu-based ray casting for unstructured volume rendering. In *Proceedings of the Fifth Eurographics / IEEE VGTC conference on Point-Based Graphics*, SPBG'08, pages 155–162, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[NK03] Zoltán Nagy and Reinhard Klein. Depth-peeling for texture-based volume rendering. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, PG '03, pages 429–, Washington, DC, USA, 2003. IEEE Computer Society.

[NNK04] Zoltán Nagy, Marcin Novotni, and Reinhard Klein. Enhancing fourier volume rendering using contour extraction. In *The 7th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI 2004)*, September 2004.

[NVI99] NVIDIA. Chapter 20. fast third-order texture filtering. `http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter20.html`, 1999. [Online; accessed 19-02-2013].

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C (2nd ed.): the art of scientific computing*. Cambridge University Press, New York, NY, USA, 1992.

[RE02] Stefan Roettger and Thomas Ertl. A two-step approach for interactive pre-integrated volume rendering of unstructured grids. In *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, VVS '02, pages 23–28, Piscataway, NJ, USA, 2002. IEEE Press.

[RGSE04] Stefan Roettger, Stefan Guthe, Andreas Schieber, and Thomas Ertl. Convexification of unstructured grids. In *In Proceedings of Workshop on Vision, Modeling, and Visualization 2004*, pages 283–292, 2004.

[RGW+03] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser. Smart hardware-accelerated volume rendering. In *Proceedings of the symposium on Data visualisation 2003*, VISSYM '03, pages 231–238, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[RSEB+00] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume on standard pc graphics hardware using multitextures and multi-stage rasterization. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '00, pages 109–118, New York, NY, USA, 2000. ACM.

[RSK06] Christof Rezk-Salama and Andreas Kolb. Opacity peeling for direct volume rendering. *Computer Graphics Forum*, 25(3):597–606, 2006.

[SB07] Rui Shen and Pierre Boulanger. Hardware-accelerated volume rendering for real-time medical data visualization. In *Proceedings of the 3rd international conference on Advances in visual computing - Volume Part II*, ISVC'07, pages 801–810, Berlin, Heidelberg, 2007. Springer-Verlag.

[SCCB05] ClĂ˘udio T. Silva, JoĂŁo L. D. Comba, Steven P. Callahan, and Fabio F. Bernardon. A survey of gpu-based volume rendering of unstructured grids. *Brazilian Journal of Theoretic and Applied Computing (RITA*, 12:9–29, 2005.

[SG10] Markus Steinberger and Markus Grabner. Wavelet-based multiresolution isosurface rendering. In *Proceedings of the 8th IEEE/EG international conference on Volume Graphics*, VG'10, pages 13–20, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[SIY93] J. Edward Swan, II, and Roni Yagel. Slice-based volume rendering, 1993.

[SK95] Lisa M. Sobierajski and Arie E. Kaufman. Volumetric ray tracing. In *VVS*, pages 11–18, 1995.

[SKKK09] Naohisa Sakamoto, Takuma Kawamura, Hiroshi Kuwano, and Koji Koyamada. Sorting-free pre-integrated projected tetrahedra. In *Proceedings of the 2009 Workshop on Ultrascale Visualization*, UltraVis '09, pages 11–18, New York, NY, USA, 2009. ACM.

[SKLE03] J. P. Schulze, M. Kraus, U. Lang, and T. Ertl. Integrating pre-integration into the shear-warp algorithm. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, VG '03, pages 109–118, New York, NY, USA, 2003. ACM.

[SMK96] Cláudio Silva, Joseph S. B. Mitchell, and Arie E. Kaufman. Fast rendering of irregular grids. In *Proceedings of the 1996 symposium on Volume visualization*, VVS '96, pages 15–ff., Piscataway, NJ, USA, 1996. IEEE Press.

[SMW98] Claudio T. Silva, Joseph S. B. Mitchell, and Peter L. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes, 1998.

[SS06] Ralf Sondershaus and Wolfgang Strasser. Segment-based tetrahedral meshing and rendering. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, GRAPHITE '06, pages 469–477, New York, NY, USA, 2006. ACM.

[TPG98] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: Improved iso-surface extraction. *Computers and Graphics*, 23:583–598, 1998.

[Tri] Peter Triers. Gpu raycasting. `http://www.daimi.au.dk/~trier/?page_id=98`. [Online; accessed 18-05-2013].

[VGK96] Allen Van Gelder and Kwansik Kim. Direct volume rendering with shading via three-dimensional textures. In *Proceedings of the 1996 symposium on Volume visualization*, VVS '96, pages 23–ff., Piscataway, NJ, USA, 1996. IEEE Press.

[vPV11] Veronika Šoltészová, Daniel Patel, and Ivan Viola. Chromatic shadows for improved perception. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, NPAR '11, pages 105–116, New York, NY, USA, 2011. ACM.

[Wan11] Gang Wang. 3d ctmri images interactive sliding viewer. `http://www.mathworks.com/matlabcentral/fileexchange/29134-3d-ctmri-images-interactive-sliding-viewer`, 2011. [Online; accessed 19-02-2013].

[Wil92] Peter L. Williams. Visibility-ordering meshed polyhedra. *ACM Trans. Graph.*, 11(2):103–126, April 1992.

[WMFC02] Brian Wylie, Kenneth Moreland, Lee Ann Fisk, and Patricia Crossno. Tetrahedral projection using vertex shaders. In *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, VVS '02, pages 7–12, Piscataway, NJ, USA, 2002. IEEE Press.

[XZC05] Daqing Xue, Caixia Zhang, and Roger Crawfis. isbvr: Isosurface-aided hardware acceleration techniques for slice-based volume rendering, 2005.

[YKEI06] Hacer Yalim Keles, Alphan Es, and Veysi Isler. Acceleration of direct volume rendering with programmable graphics hardware. *Vis. Comput.*, 23(1):15–24, December 2006.

[ZG06] Yuan Zhou and Michael Garland. Interactive point-based rendering of higher-order tetrahedral data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):2006, 2006.