Západočeská univerzita v Plzni

Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Metody přímého zobrazování isoploch ve volumetrických datech

Marek Kročák

Plzeň, květen 2000

Obsah

Úvod

Výpočetní technika stále ještě není schopná pojmout realitu ve všech jejích souvislostech a detailech. Ačkoliv její kvantitativní parametry neustále rostou a vyvíjejí se vysokou rychlostí, stále to není dostačující a pro automatické zpracování dat je potřeba vytvářet zjednodušené modely objektů reálného světa. Mezi všemi obory výpočetní techniky nemá v tomto směru ani počítačová grafika žádné výsadní postavení. I ona se zabývá zpracováním dat a informací z reálného světa a i ona musí řešit problematiku vytváření vhodných modelů a jejich vnitřních reprezentací. Rychlý vývoj ji rozdělil do mnoha proudů a každý z nich se s tím vypořádává odlišným způsobem vyplývajícím z jeho odlišných potřeb. Stává se však, že se některé proudy opět setkají a mohou se navzájem obohatit, jako se to stalo i v případě povrchových a objemových modelů pro počítačovou grafiku.

Povrchová reprezentace

Systémy počítačově podporovaného návrhu (*Computer Aided Design*, CAD) a většina dostupných systémů pro realistické zobrazování spoléhají na *povrchovou* (*hraniční*) reprezentaci objektů. Je to přirozené, poněvadž u naprosté většiny z nich je vidět a tedy zobrazován pouze povrch. Ten je zpravidla rozdělen na jednodušší geometrické útvary, které jej mohou buď reprezentovat přesně nebo nějakým způsobem aproximovat v závislosti na aktuálních potřebách.

Nejjednodušším způsobem hraniční reprezentace geometrických objektů je jejich *analytický popis*. Umožňuje přesně vypočítat pro zobrazování potřebné charakteristiky povrchu jako průsečík s paprskem a příslušný normálový vektor.

Alternativním způsobem povrchové geometrické reprezentace je aproximace obecné plochy množstvím rovinných geometrických primitiv, zpravidla trojúhelníků (tzv. *plošková hraniční reprezentace*).

Tento způsob doznal hojného rozšíření a je univerzálně přijímán pro svoji jednoduchost, úspornost a zejména pružnost. Usnadňuje totiž další manipulaci s takto reprezentovanými objekty a je přímo podporován i výrobci hardwarových akcelerátorů. Ačkoliv je pro mnoho situací a potřeb plně dostačující, existují oblasti, kde je vyloženě nevhodný. Všude tam, kde je potřeba zachovat či znázornit vnitřní strukturu, povrchový přístup selhává. Alternativou je v tomto případě reprezentace *objemová*.

Objemová reprezentace

Objemová, často pod názvem *volumetrická*, reprezentace má své nezastupitelné místo v lékařství. Snímky vnitřního uspořádání předmětů nebo lidského těla získané pomocí *magnetické rezonance* (*Magnetic Resonance Imaging*, MRI), *počítačové tomografie* (*Computer Tomography*, CT) nebo *ultrazvuku* (*Ultrasonography*, US) jsou jejími názornými představiteli. Ale nejenom v lékařství. Také například při řešení fyzikálních modelů proudění v tekutinách nebo při numerické analýze matematických modelů přichází objemová reprezentace ke slovu.

Protože je však paměťově náročná („objemná") a běžný výpočetní výkon dosud nepostačuje k jejímu efektivnímu zpracování, objevily se i přístupy kombinované, respektive pokusy o převody z jedné reprezentace do druhé. Ve volumetrických datech lze vyhledávat tzv. *isoplochy*, tedy plochy spojující body, které mají nějakou shodnou konstantní vlastnost, převést je na geometrickou reprezentaci, nejčastěji trojúhelníkovou síť, a tu dále zpracovávat. Příkladem budiž zobrazení povrchu kosti v tomografickém snímku (obr. 1).

Obrázek 1: Geometricky aproximovaný povrch kosti

„Inverzní" transformací lze geometrické objekty *voxelizovat*, tedy převádět do objemové reprezentace.

Tato práce se dále v souladu se zadáním soustředí na reprezentaci a zobrazování isoploch ve volumetrických datech.

Kapitola 1

Charakter objemových dat

Ačkoliv existuje způsob, jak analyticky popsat vnitřní strukturu objektu například převodem do frekvenční oblasti, je to cesta matematicky velmi náročná a proces zobrazování se tím spíše komplikuje. Objemová data proto můžeme téměř výhradně najít v diskretizované, navzorkované podobě, matematicky vyjádřeno se jedná o funkci tří nezávislých proměnných

$$f(x, y, z);\ x, y, z \in \mathbf{Z}$$

Podle původu lze objemová data zhruba rozdělit do tří skupin na:

- data naměřená
- výsledky výpočtů (data vypočítaná)
- data synteticky generovaná

Příkladem dat získaných *měřením* budiž již několikrát zmíněný tomografický snímek. Ať už se jedná o lékařský nebo průmyslový tomograf, používaný v defektoskopii, pomocí rentgenového záření je nasnímán zkoumaný předmět nebo člověk a po nezbytném zpracování naměřených údajů získáme voxelový datový soubor. Ten se skládá z tenkých rovnoběžných řezů, a hodnoty, jež obsahuje vyjadřují pohltivost rentgenového záření v tkáni či materiálu na daném místě. Řezy jsou nejčastěji zobrazovány tak, že intenzita bílé barvy přímo úměrně odpovídá míře pohltivosti (obr. 1.1).

Dalšími metodami měření je například již zmíněná magnetická rezonance, měření biologické aktivity tkání (SPECT), elektronová mikroskopie a další.

Druhá kategorie zahrnuje *data vypočítaná*. Obsahují zpravidla výsledky řešení soustavy nebo soustav diferenciálních rovnic. Podoba těchto dat je odvislá od zkoumané úlohy a použitého výpočetního postupu.

Data syntetická jsou aktuální zejména v oblasti modelování. Mohou do sebe pojmout modely předmětů, geometrických útvarů, průmyslových součástek nebo třeba krajiny. S jejich pomocí se dají simulovat obráběcí procesy, postup vytváření uměleckých plastik a podobně. V literatuře se lze setkat i s termínem *fantomová data*.

Při výzkumu pro tuto práci byla syntetická data využita zejména k testování a porovnávání různých zobrazovacích algoritmů a jejich kombinací.

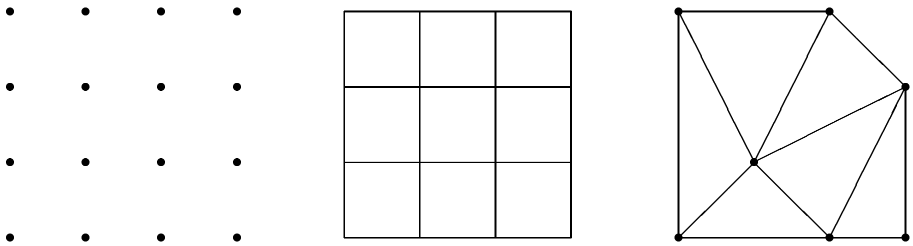
a) b) c)

Obrázek 1.1: Ukázky řezů objemovými daty. a) CT, b) MRI (s částečně odstraněnou lebeční kostí), c) segmentová informace k MRI snímku

1.1 Voxely a buňky

Volumetrická data si můžeme představit jako prostorovou síť, v jejíchž uzlech jsou umístěny hodnoty stejného typu (*voxely*). Ty mohou být skalární, vektorové, případně složitější. Vzdálenosti mezi sousedními uzly této sítě mohou ale nemusí být pravidelné. V nejjednodušším případě se jedná o pravidelnou pravoúhlou třírozměrnou mřížku. Toto chápání je běžné zejména v lékařství.

Prostor mezi uzly mřížky je rozdělen do geometrických *buněk* (*cells*), s nejbližšími uzly ve vrcholech. Nepravidelné prostorové sítě se nejčastěji skládají z buněk ve tvaru čtyřstěnů, v případě pravidelných mřížek jsou tvořeny osmi nejbližšími vrcholy kvádru či krychle.



a) b) c)

Obrázek 1.2: a) pravidelná voxelová mřížka, b) pravidelné buňky, c) nepravidelná síť

Obecné sítě mají jednoznačnou výhodu adaptivity. V místech, kde je to zapotřebí mohou větší hustotou buněk přesněji vystihnout lokální změny vlastností materiálu.

V dalším textu a v praktické části budou brána v úvahu jen voxelová data v pravidelné pravoúhlé mřížce a to zejména z důvodu snadné dosažitelnosti již existujících datových souborů.

1.2 Obsažená informace

Samotné hodnoty skalárních dat mohou reprezentovat mnoho různých druhů informace:

šedotónová data vyjadřují nějakou skalární vlastnost v daném místě (hodnota pohltivosti, rezonance, teplotu, elektrický potenciál apod.) Jedná se o údaje, které se dají velmi dobře interpolovat i jednoduchými metodami.

segmentová (indexová) data nenesou přímou informaci o vnitřní struktuře, ale přesto jsou s šedotónovými údaji v přímé souvislosti neboť určují příslušnost stejnolehlých voxelů ke konkrétním objektům obsaženým ve scéně – například ohraničují jednotlivé orgány v lidském těle. Tato informace není vždy jednoznačně odvoditelná přímo ze získaných dat a proto bývá doplněna poloautomaticky (s lidskou asistencí) použitím různých druhů filtrace a morfologických operátorů z oblasti zpracování obrazu (*image processing*).

vzdálenostní data jsou nepříliš známou variantou skalárních dat. Plocha je reprezentována objemem veličin vyjadřujících vzdálenost daného voxelu k nejbližšímu bodu hledané plochy. Ta je v tomto případě reprezentována všemi body, jejichž vzdálenost od povrchu je nulová. Jedná se vlastně o negeometrickou povrchovou reprezentaci. Její vlastnosti jsou uvedeny v kapitole 5.

vektorová data mohou představovat například gradient skalárních veličin, směr a rychlost proudění, ale také RGB složky třírozměrné barevné textury.

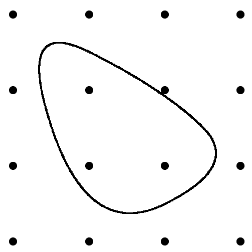
Obecně mohou objemová data obsahovat téměř jakékoliv hodnoty. Zaměříme se na *skalární šedotónová objemová data*. Jsou základem, na kterém stojí a z kterého vycházejí všechny běžně používané algoritmy.

1.3 Isoplochy ve volumetrických datech

Nejběžnější způsob reprezentace isoploch jsou přímo voxelové hodnoty přítomné v datech. Z datového souboru tedy pro zobrazování vybereme jen ty voxely, které mají hledanou hodnotu. Ačkoliv se tento způsob nabízí jako nejjednodušší, není takto jednoduše přímo použitelný. Hodnoty se v datovém prostoru nemění spojitě, ale často s krokem, který je dán buď charakterem objektu nebo způsobem měření dat. Často jsou hodnoty diskretizovány a omezeny na celá čísla v rozsahu 1 až 12 bitů. Nikde není zaručena velikost rozdílu hodnot mezi sousedními vzorky. Jestliže tedy hodnota jednoho vzorku je menší než hodnota hledané isoplochy a hodnota jeho souseda je větší, prochází zřejmě isoplocha prostorem mezi nimi, ale algoritmem by nebyl vybrán žádný z nich. Je zapotřebí nějaký mechanismus, který zjistí, respektive pokusí se alespoň přibližně určit (interpolovat ze známých hodnot), kudy hledaná plocha prochází.

Jaký má vlastně význam reprezentovat a modelovat isoplochy pomocí objemových dat?

- voxelová scéna má konstantní velikost i když se mění počet objektů, které obsahuje,
- při dostatečném rozlišení se její velikost nemění ani s narůstající složitostí objektů,



Obrázek 1.3: Příklad isoplochy v pravidelné mřížce

- klasický raytracing počítá analyticky průsečíky s mnoha druhy objektů (koule, plochy, NURBS, ...), ale objemový raytracer si vystačí s jediným elementem – voxelem. Je tedy principiálně jednodušší.

Kapitola 2

Zobrazování volumetrických dat

2.1 Nepřímé metody

Velká část metod, které se zabývají získáváním a zobrazováním isoploch spoléhá na jejich převod do geometrické ploškové reprezentace. Algoritmy pro zpracování geometrických objektů jsou dobře propracované a běžně k dispozici v existujících programových produktech. V žádném případě nelze zanedbat ani hardwarovou podporu, která další zpracování geometrických primitiv velmi výrazně urychluje a je dnes součástí naprosté většiny komerčně dostupných zobrazovacích zařízení. Díky této podpoře dosáhla geometrická reprezentace dnešní popularity a bez ní by interaktivní práce s datovými soubory běžných rozměrů nebyla myslitelná.

Mezi nejznámější a nejvíce rozšířené metody extrakce isoploch z objemových dat patří metody *Marching Cubes* (MC) a *Marching Tetrahedra* (MT), jejichž výsledkem je trojúhelníková síť, a poněkud opomíjená metoda *Dividing Cubes* (DC), která generuje množství bodů ležících na povrchu hledané isoplochy. Množství a hustota těchto bodů je zvolena tak, aby při výstupu na rastrové zařízení vznikl obraz spojitého povrchu.

Výsledkem dlouholetého výzkumu těchto metod je množství modifikací základních algoritmů zaměřených zejména na urychlování celého výpočetního procesu extrakce, eliminaci nadbytečných dat ve výsledné síti a rozhodování některých singulárních případů.

2.2 Přímé metody

Pod pojmem *přímé zobrazování* je chápán postup, kdy při vytváření dvourozměrného obrazu nevzniká dočasná geometrická podoba objemových dat. Většina metod je založena na *metodě vrhání paprsku.* Její princip je jednoduchý. Známe-li pozici pozorovatele, směr pohledu a polohu obrazu, potom je z pozice pozorovatele vyslán „skrz“ každý bod tvořeného obrazu fiktivní paprsek do zobrazované scény a při jejím průchodu paprsek může zasáhnout některý z objektů. Pro nalezený průsečík se vypočítá barva, tzv. *barva paprsku*, a ta je v obrazu přiřazena na místo, kterým paprsek procházel.

Způsob výpočtu barvy paprsku může být jednoduché přiřazení barvy tělesa, nebo mohou být brány v úvahu faktory jako například osvětlení, vzdálenost od pozorovatele, průhlednost či průsvitnost povrchu, jeho odrazivost, lom světla a mnohé další. Všechny tyto faktory komplikují výpočet, ale výsledný efekt, zaplacený delší dobou výpočtu, je často velmi realistický a významným způsobem napomáhá pozorovateli vytvořit si správnou představu o prostorových vztazích ve scéně.

Základními metodami pro přímé zobrazování objemových dat jsou:

- metoda vrhání paprku (*raycasting*)
- metoda sledování paprsku (*raytracing*)

Obě výše jmenované metody mohou být upraveny tak, aby mohly alespoň částečně zastoupit jedna druhou, nebo mohou být zakomponovány do jednoho systému.

2.2.1 Metoda vrhání paprku

Primárně je určena pro zobrazování vnitřní struktury objemových dat. Hodnoty obsažené v jednotlivých uzlech či buňkách můžeme chápat například jako hodnoty jasu a pohltivosti/průhlednosti materiálu. Každý voxel potom díky své pohltivosti zastíní voxely za sebou ve směru paprsku, ve směru k pozorovateli jistou část světla odrazí a zbytek propustí. Paprsek tedy prochází objemem dat a cestou akumuluje hodnoty jasu a barvy v závislosti na pohltivosti a výsledný jas je následně přiřazen příslušnému bodu na obrazovce. Čím hlouběji postoupí paprsek „do materiálu“, tím slaběji budou vidět přítomné detaily, neboť budou zastíněny hmotou v popředí.

2.2.2 Metoda sledování paprku

Klasický postup je založen na hledání průsečíků s plochami a proto se hodí zejména pro zobrazování isoploch. Z tohoto důvodu byla tato metoda vybrána pro implementaci a jejímu podrobnému popisu je vyhrazena celá kapitola 3.

2.3 Další způsoby klasifikace

Jako téměř vše na světě i metody zobrazování se dají klasifikovat a hodnotit z mnoha úhlů pohledu. Z hlediska systému zpracování datového souboru rozpoznáváme algoritmy typu:

screen order, ve kterých je pořadí zpracování jednotlivých voxelů dáno vykreslovaným obrazem. Do této kategorie patří například metody sledování a vrhání paprsku.

object order, které procházejí prostorem dat a zpětně je transformují do prostoru obrazu. Zástupcem může být například *splatting*, nebo většina nepřímých metod pro extrakci isoploch (MC, MT,...).

2.3.1 *Screen Order* metody

Jemnější rozlišení metod si všímá způsobu procházení scénou podél paprsku. Dva základní přístupy mají označení:

Front To Back (FTB) postupující od pozorovatele směrem do scény a potom k pozadí. Jeho výhodou je, že může produkovat meziobraz, který je dále zpřesňován. Mnohé akcelerační metody jsou založeny právě na FTB průchodu daty.

Back To Front (BTF) algoritmus sleduje paprsek od nejvzdálenější části scény směrem k pozorovateli. Narozdíl od FTB se dá s úspěchem hardwarově urychlit například použitím *α-blendingu* a 3D textur v některé z implementací *OpenGL Extensions* [11].

2.3.2 *Object Order* metody

Object order znamená, že se primárně nevychází z obrazu a jednotlivých pixelů, ale procházíme volumetrickým souborem dat a pro každý návštívený voxel vypočítáme jeho příspěvek k celému obrazu, který zobrazíme. Nejznámější a zároveň velmi rychlou metodou je *splatting* [3]. Pro každý voxel je vypočítán tzv. *footprint*, neboli *otisk*, jenž odpovídá příspěvku daného voxelu k celkovému obrazu. Průchod daty a postupné kumulativní zobrazování otisků všech navštívených voxelů připomíná jejich „naplácávání“, jak zní překlad anglického názvu této metody, na sebe neboť jednotlivé otisky se často vzájemně překrývají.

2.4 Předpoklady zobrazování

Naměřená skalární data jsou často zatížena šumem a proto je třeba před vlastním zobrazováním aplikovat některou z metod filtrace. Většinou se však tento krok provádí vzápětí po nasnímání, takže datové soubory, které jsou běžně k dispozici, jsou již „čisté“. Výjimkou jsou ultrazvukové snímky. Šum je zde natolik silný, že pro jejich zobrazování jsou vyvíjeny zvláštní metody.

Zvláštní skupinou indexových dat jsou data dvouhodnotová (binární). Aplikujeme-li na ně stejné zobrazovací metody jako na data šedotónová, dostaneme z hlediska věrnosti realitě neuspokojivé výsledky. Binární informace nestačí k odhadu všech parametrů potřebných pro rekonstrukci povrchu a ten je potom plný nepříjemných artefaktů a to i v případě použití výkonnějších zobrazovacích postupů.

Nemáme-li tedy k dispozici jiná data, existují metody, které se více či méně úspěšně snaží tato „kostrbatá“ data vyhladit. Bývají založeny na nějaké formě *low-pass filtrace*[4].

Zajímavou variantou k filtračním postupům je například tzv. *metoda elastické povrchové sítě* (*Constrained Elastic Surface Net*)[6], která dokáže artefakty odstranit iterativním způsobem na základě minimalizace povrchové energie sítě, kopírující hranici binárně určeného objektu.

Kapitola 3

Metoda sledování paprsku

Metoda sledování paprsku je ideově relativně starou technikou a její kvality i omezení jsou poměrně dobře známy. Mezi jednoznačné výhody patří:

- implicitní řešení viditelnosti,
- schopnost modelovat lom (refrakci) i odraz (reflexi) světla nejenom od rovinných ploch, ale od každé, ke které můžeme vypočítat normálu k povrchu v průsečíku s paprskem,
- schopnost jednoduše zobrazovat nejen v paralelní a perspektivní projekci, ale také v projekcích cylindrických, sférických nebo jinak nestandardních,
- schopnost modelovat některé jevy spojené s fyzikální podstatou šíření světla: rozklad bílého světla (disperzi) při lomu, či průchodu materiálem, zakřivení světelných paprsků v gravitačním poli a další fyzikální úkazy spojené s teorií relativity,
- dosažitelná vysoká vizuální věrnost obrazu modelované situaci.

K nevýhodám se počítá zejména:

- obtížná schopnost modelování zdrojů měkkého světla (plošných zdrojů) a tedy modelování vícenásobných odrazů světla a z toho vyplývající komplikace při zobrazování realisticky vypadajících stínů,
- vysoká výpočetní náročnost,
- velmi omezená možnost preprocessingu.

Existují postupy, jak některé nevýhody eliminovat nebo aproximovat, ovšem za cenu dále zvýšené výpočetní náročnosti.

Vlastní raytracing obrazu je jednoduchý a sestává z následujících kroků, které jsou opakovány pro každý obrazový bod (*pixel*):

- zjistit počátek a směr paprsku,

- vlastní sledování (*tracing*) paprsku – průchod scénou podél jeho dráhy a nalezení objektů, které protíná,
- v případě zobrazování povrchu těles je zapotřebí zjistit polohu průsečíku a zjistit normálu k povrchu v daném místě (existují algoritmy, které se v tomto bodě obejdou bez zjišťování normálového vektoru),
- výběr průsečíku nejblíže pozorovateli,
- podle zvoleného zobrazovacího modelu určení jeho barvy.

Poslední uvedený bod může znamenat, že bude potřeba vyslat z pozice průsečíku sekundární paprsky, s jejichž pomocí bude možno vyhodnotit například polohu stínu nebo obraz odražený od lesklého povrchu.

3.1 Generování paprsku

Počátek paprsku je možno téměř vždy umístit do pozice pozorovatele, ale to je jen první krok. V druhém kroku je potřeba určit jeho směr podle úhlu pohledu a druhu použitého promítání. Souřadný systém používaný dále při výkladu je důsledně pravotočivý.

3.1.1 Orientace pozorovatele

K jednoznačnému určení orientace pozorovatele v prostoru je zapotřebí alespoň tří údajů:

- přesná poloha v prostoru
- orientace – vektor určující směr „nahoru“ nebo „doprava“ z hlediska pozorovatele
- směr pohledu – směrový vektor nebo souřadnice bodu, na který je pohled zaměřen

Pro zjednodušení všech výpočtů předpokládejme, že světová osa Y směřuje z pohledu pozorovatele vždy vzhůru. Vyhneme se tak potřebě zvláštní definice vektoru orientace. Za zjednodušení zaplatíme tím, že nejsme schopni orientaci jednoznačně určit, jestliže směr pohledu je rovnoběžný se světovou osou Y a proto se těmto krajním polohám vyhneme.

Znalost orientace pozorovatele usnadňuje další výpočet orientace obrazu („promítacího plátna“) v prostoru.

- osa X obrazu $ImageX$ je kolmá na světovou osu Y $WorldY = (0, 1, 0)$ i na vektor směru pohledu $View$ – vypočítáme ji tedy jejich vektorovým součinem.

$$ImageX = View \times WorldY$$

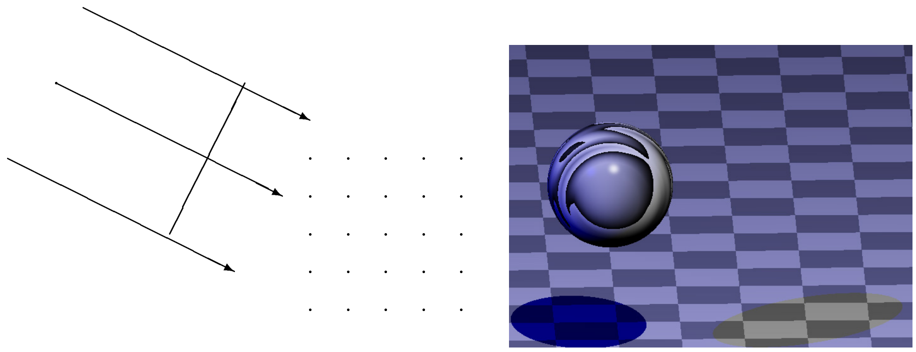
- osa Y obrazu je kolmá na obrazovou osu X a směr pohledu. Získáme ji opět jejich vektorovým součinem.

$$ImageY = View \times ImageX$$

3.1.2 Paralelní promítání

Všechny paprsky jsou rovnoběžné a jejich směr je shodný se směrem pohledu. Trochu vyjímečné je toto promítání tím, že paprsky nemohou mít počátek v pozici pozorovatele, byly by totiž všechny shodné.

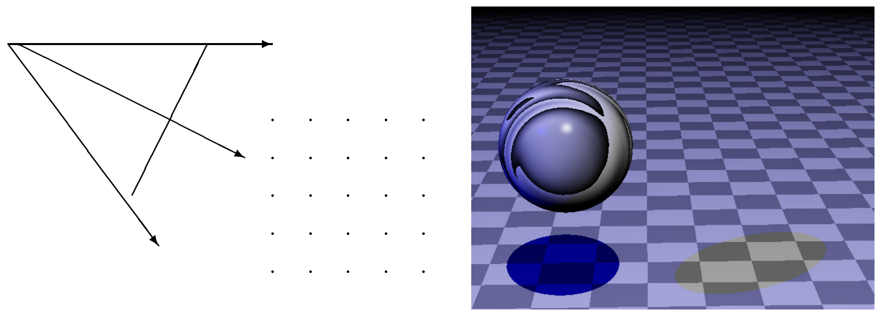
Počátky paprsků tedy posuneme podle os rovnoběžných s osami obrázku. Jestliže střed obrázku bude mít souřadnice [0,0], potom sousední bod vlevo bude mít souřadnice [-1,0] a to je zároveň vektor, o který je potřeba počátek paprsku posunout relativně ke středu obrazu.



Obrázek 3.1: Paralelní projekce

3.1.3 Perspektivní promítání

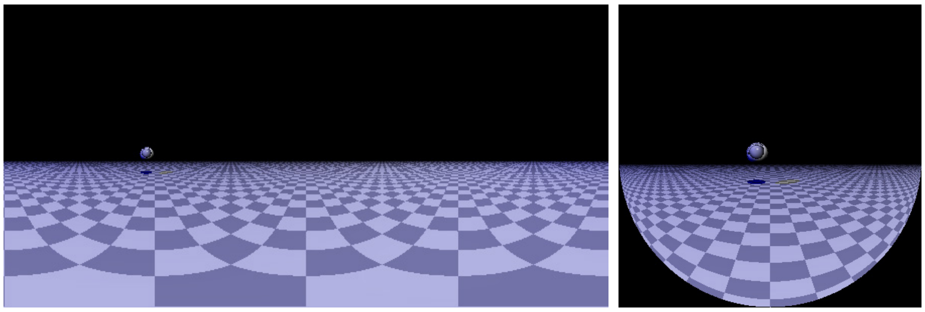
Tentokrát jsou počátky všech paprsků shodné s pozicí pozorovatele a liší se svým směrem. Každý paprsek je definován počátkem v místě pozorovatele a směrem, jenž je určen spojnicí počátku a toho bodu obrazu, pro který chceme paprsek vyslat. Střed obrázku má opět relativní souřadnice [0,0]. Skutečnou orientaci v prostoru již určit umíme.



Obrázek 3.2: Perspektivní projekce

3.1.4 Sférická projekce

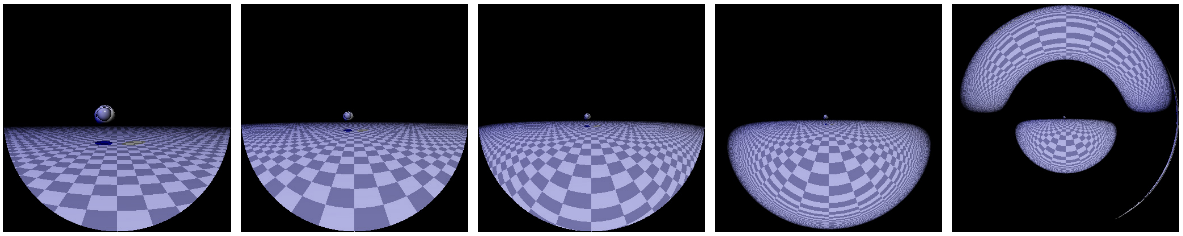
Existuje několik modifikací sférické projekce. Nejjednodušší modeluje ideální odraz paprsku od povrchu koule. Reálný zorný úhel je 180 stupňů a je podobný fotografickému zobrazení objektivem s velmi malou ohniskovou vzdáleností (*rybí oko*, *Fisheye*) (obr. 3.3 b).



a) b)

Obrázek 3.3: Sférická projekce a projekce rybí oko (Fisheye)

O něco málo složitější je sférická projekce, jež není omezena 180 stupni zorného úhlu. Prakticky tato projekce rozvine povrch koule do roviny tak, že myšlený severní pól je středem obrazu a jižní pól, jenž jako jediný bod celé koule nemá jednoznačný obraz, se rozvine do obvodové kružnice. Teoreticky je možné získat libovolně veliký zorný úhlel, nicméně prakticky dochází při úhlech větších než 360 stupňů k násobným obrazům jednotlivých bodů ve scéně.



Obrázek 3.4: Projekce Fisheye s volitelným zorným úhlem (90, 180, 270, 360, 720)

Třetí variantou je mapování, které osám X a Y přiřadí horizontální a vertikální úhel (otočení a elevaci, astronomicky řečeno rektascenzi a deklinaci, geograficky – zeměpisnou délku a šířku). Často se používá právě v kartografii pro projekci povrchu zemského glóbu do roviny (obr. 3.3 a).

3.1.5 Cylindrická projekce

Zjednodušeně řečeno je cylindrické promítání (promítání na povrch válce) kombinací perspektivní a sférické projekce. Ve směru shodném s osou X obrázku počítáme podle sférické a ve směru osy Y obrázku, která je rovnoběžná s podélnou osou válce, používáme projekci perspektivní.

3.2 Hledání průsečíku paprsku s isoplochou

Analyticky definovaný paprsek prochází v prostoru mezi jednotlivými vzorky a v pravidelných intervalech kontroluje hodnoty dat podél dráhy paprsku, zda neprotnul isoplochu. Při průchodu objemovými daty a hledání průsečíku s případnou isoplochou hrají tedy zásadní roli dva faktory:

- interpolační metoda – určuje přesný průběh hodnot mezi voxely a tím i přesný tvar isoplochy
- vzorkovací frekvence – ovlivňuje interval na dráze paprsku, v němž se isoplocha nachází

3.2.1 Interpolace

interpolace nejbližším sousedem je rychlá, jednoduchá, ale prakticky použitelná pouze tehdy, jestliže velikost jednoho voxelu po zobrazení nepřekračuje velikost bodu rastrového výstupního zařízení.

lineární interpolace (respektive tri-lineární neboť lineárně interpolujeme hodnoty ve všech třech rozměrech) dává výsledky takřka o jeden řád lepší. Praktické pokusy ukázaly, že pro běžná lékařská data je tato interpolace naprosto dostačující.

interpolace kosinová je převzatá z algoritmů pro zpracování 2D obrazu a do systému byla zahrnuta pokusně jako případná jednodušší náhrada kubické interpolace. Experimentální výsledky ukazují na její nevhodnost pro praktické použití, neboť v detailech vytváří netolerovatelné artefakty.

kubická (trikubická) interpolace pracuje s polynomy třetího stupně. Výpočetně je výrazně náročnější než lineární, ale viditelný rozdíl poskytuje až při zobrazování detailů o velikosti řádově jednotek až desítek voxelů.

3.2.2 Vzorkování

Objemová data principiálně nemohou správně reprezentovat detaily, jejichž velikost je menší než dvojnásobek minimální vzdálenosti mezi jednotlivými vzorky. Vyplývá to z *Nyquistova kritéria*, které zároveň definuje minimální vzorkovací frekvenci zaručující eliminaci *aliasingu*, zdroje nepříjemných artefaktů.

Jestliže datový soubor splňuje uvedené kritérium, potom nám stačí kontrolovat hodnotu skalární veličiny podél paprsku s krokem, jenž odpovídá jednomu vzorku na voxel. Přesněji řečeno jednomu vzorku na každou buňku, kterou paprsek protne, neboť paprsek většinu času prochází prostorem mezi jednotlivými vzorky.

V literatuře [7] je rovněž popsán postup, kdy jsou vzorky odebírány na hranicích buněk, na místech, kde do buňky paprsek vstupuje a z ní vystupuje. Výhodou je jednodušší interpolace (vždy jen v některé osové rovině). Nevýhodou je pak nutnost analytického výpočtu průsečíku paprsku se stěnami buňky (v případě paralelní projekce existuje výrazná možnost preprocessingu pro všechny paprsky), mírná nekompatibilita s raycastingem, který často postupuje s kostantním krokem.

Je třeba brát také v úvahu nestejné vzdálenosti vzorků ve směru jednotlivých os. Teoreticky by se dala pro každý paprsek vypočítat ideální frekvence, která by zaručila právě jeden vzorek pro každou buňku, ale jelikož je tato hodnota závislá na směru paprsku, bylo by třeba tento výpočet provádět pro každý paprsek a to s sebou nese příliš velikou výpočetní zátěž v případech, kdy směry paprsků nejsou shodné, například při perspektivní projekci nebo při výpočtech spojených s odrazem světla.

Použijeme tedy převrácenou hodnotu minimální vzdálenosti mezi sousedními vzorky. Zaručí, že nedojde k podvzorkování, ale za cenu, že v některých případech budeme vzorkovat častěji, než je nezbytně nutné. Takto vzniklá výpočetní zátěž je vyvážena přesnějšími výsledky v místech, kde odebereme více vzorků.

3.3 Výpočet polohy průsečíku

Při průchodu podél paprsku odebíráme vzorky dat a v zásadě mohou nastat jen dvě možnosti. Paprsek buď protne nebo neprotne isoplochu dříve než opustí prostor souboru dat. Druhá z možností pro nás není příliš zajímavá a proto se soustředíme na hledání přesné polohy průsečíku.

Jestliže hodnoty dvou odebraných sousedních vzorků jsou rozdílné v tom smyslu, že hodnota jednoho z nich je větší než hodnota hledané isoplochy a hodnota druhého je menší, potom je více než pravděpodobné, že hledaný průsečík leží v oblasti mezi těmito dvěma vzorky.

V nejjednodušším případě se spokojíme s tím, že polohu jednoho z nich budeme pokládat za pozici průsečíku. Je to ta nejjednodušší a nejrychlejší aproximace průsečíku, jakýsi ekvivalent interpolace nejbližším sousedem. Její přesnost závisí pouze na vzorkovací frekvenci. Přesnějších výsledků dosáhneme jejím zvýšením, ale tak se ovšem také prodlouží doba zobrazování, neboť jsme nuceni odebrat a zpracovat úměrně větší počet vzorků.

Snažíme se tedy zachovat frekvenci jen nezbytně vysokou (co nejnižší), ale zpřesnit odhad polohy průsečíku jiným způsobem. Analytický výpočet přichází v úvahu, ale nepoužijeme jej, protože:

- kvůli modularitě se snažíme od sebe oddělit proces vzorkování a proces interpolace,
- je odvozen v [7] pro lineární interpolaci, ale my se nechceme omezit jen na ni.

Zvážíme možnosti numerického řešení. Podle použité interpolační metody může být průběh hodnot mezi našimi dvěma kritickými vzorky spojitý nebo nespojitý, vždy je však nerostoucí nebo neklesající a to nás opravňuje k použití některé z následujících metod.

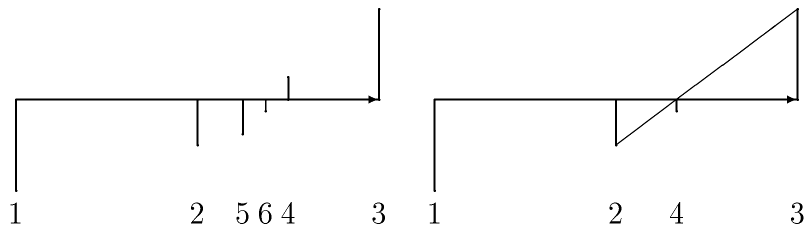
3.3.1 Půlení intervalu

První z uvažovaných metod je *půlení intervalu*, jež se běžně používá při vyhledávání v seřazených posloupnostech. Vzdálenost mezi vzorky rozdělíme na polovinu a interpolujeme hodnotu na nové pozici. Na její místo pak posuneme tu mez původního intervalu, jež leží na „stejné straně“ isoplochy. Je-li tedy první vzorek menší, druhý větší a hodnota mezi nimi menší než hledaná, potom zmenšíme kritický interval tak, že jeho dolní mez posuneme vzhůru na onu mezipozici. Postup můžeme opakovaně použít na nový, již o polovinu kratší interval a dále pokračovat, dokud nedosáhneme požadované přesnosti nebo mezního počtu iterací.

3.3.2 Metoda sečen

Metoda sečen (Newtonova metoda) spočívá v jednoduché úvaze: jestliže jedna z obou krajních hodnot je blíže hledané hodnotě, potom i její pozice bude zřejmě blíže pozici hledaného průsečíku s isoplochou. Interval se proto nerozdělí na dvě stejné části jako v případě půlení intervalu, ale části budou mít takový poměr velikostí, který odpovídá rozdílu krajních hodnot a hodnoty hledané.

I tento postup lze iterativně opakovat, nicméně experimentálně bylo ověřeno, že již první odhad je velmi dobře použitelný a vizuálně je ekvivalentní analytickému výpočtu. Protože však předpokládá spojitou změnu hodnoty mezi sousedními vzorky, nehodí se tato metoda pro použití ve spojení s interpolací nejbližším sousedem, jejíž výstup spojitý není.



Obrázek 3.5: Výpočet polohy průsečíku – půlení intervalu a metoda sečen – čísla udávají pořadí, ve kterém byly vzorky odebrány.

3.4 Výpočet normály v průsečíku s isoplochou

Nyní známe přesnou polohu průsečíku, ale pro správné zobrazení je potřeba zjistit také správnou orientaci plochy v daném bodě. Orientace je jednoznačně určena *normálovým vektorem* v místě průsečíku. Jelikož celá isoplocha je pouze odhadem a aproximací skutečnosti, nezbývá než odhadnout i normálový vektor. Poslouží nám k tomu *vektor gradientu*, jenž má směr největšího růstu skalární veličiny.

Vztah mezi gradientem a isoplochou je podobný vztahu mezi vrstevnicí a spádnicí. Vrstevnice spojuje body se stejnou nadmořskou výškou. Spádnice je na ní kolmá a má směr největšího klesání. Podobně je tomu i s gradientem. Je kolmý na isoplochu a jediný rozdíl je ten, že má směr největšího růstu a nikoliv největšího klesání. Vektor gradientu g odhadneme ve všech vrcholech buňky metodou *centrální (symetrické) diference* ze okolních voxelových hodnot v.

$$g(x,y,z) = \left(\frac{v(x+1,y,z)-v(x-1,y,z)}{2}, \frac{v(x,y+1,z)-v(x,y-1,z)}{2}, \frac{v(x,y,z+1)-v(x,y,z-1)}{2}\right)$$

Hodnotu gradientu v průsečíku s isoplochou určíme interpolací z gradientů všech osmi vrcholů buňky a místo normály v bodě průsečíku použijeme jeho opačnou hodnotu. Může nastat situace, kdy diferenční metodou nelze odhadnout gradient dat. V tomto případě, lze použít gradient *paměti hloubky* (*depth buffer*). Stejně jako do obrazu ukládáme do každého pixelu barvu příslušného průsečíku, paměť hloubky uchovává vzdálenosti těchto průsečíků od pozorovatele. Metoda nedává tak přesné výsledky, ale je rovněž velmi dobře použitelná. V některých případech může pomoci eliminovat některé artefakty (kap. 6).

3.5 Osvětlovací model

Pro úspěšné realistické zobrazování je potřeba mít odpovídající osvětlovací model. Těchto modelů bylo vytvořeno několik. Některé vycházejí z fyzikální podstaty světla a některé vychází z empirické zkušenosti. V praxi je nejčastěji používán empirický světelný model navržený Bui-Tuong Phongem již v roce 1975.

3.5.1 Difúzní a spekulární složka

Při pozorování světla odraženého od povrchu těles můžeme pozorovat dvě základní složky, z nichž se odražené světlo skládá:

difúzní – nesoucí zejména informaci o barvě povrchu a intenzitě jeho osvětlení

spekulární (zrcadlovou) – závislou na směru pohledu a asi nejlépe pojmenovatelnou „lesklostí“

Difúzní složka je nezávislá na směru pohledu. K jejímu určení potřebujeme znát normálový vektor k povrchu a polohu světelného zdroje. *Intenzita difúzního odrazu* I_d od daného místa na povrchu je maximální, jestliže normálový vektor N a vektor směřující ke zdroji světla L jsou shodné. Naopak jestliže svírají úhel větší nebo rovný 90 stupňům, povrch není přímo osvětlen. Tuto závislost nejlépe popisuje skalární součin obou vektorů, jenž má geometrický význam kosinu úhlu jimi sevřeného.

$$I_d = N \cdot L$$

Intenzita zrcadlové složky I_s závisí na úhlu odrazu světelného paprsku ze zdroje světla od povrchu. Opět je využit skalární součin, ale tentokrát násobíme vektor směru odraženého světla R_l a vektor směřující k pozorovateli V. Zde přichází ke slovu parametr h vyjadřující *ostrost zrcadlového odrazu*, v anglické terminologii nazývaný *shininess*, tedy

„vyleštěnost“ nebo „lesklost“ povrchu. Prakticky je to mocnitel, jímž se získaný skalární součin umocní. Barva spekulární složky není nezbytně závislá na barvě povrchu.

Odražený paprsek má počátek v místě průsečíku s plochou a směr vypočítáme podle předpisu:

$$R_l = 2 \cdot (L \cdot N) \cdot N - L$$

R_l je směrový vektor odraženého světelného paprsku
L je vektor směřující ke zdroji světla
N je povrchový normálový vektor

Intenzitu spekulárního odrazu I_s definuje vzorec:

$$I_s = (R_l \cdot V)^h$$

Jak již bylo uvedeno, jedná se o empirický osvětlovací model, který nemá příliš hlubokou souvislost s fyzikálně přesným šířením světla, nicméně je jeho velmi přesvědčivou aproximací.

3.5.2 Skládání barev

Skládáním barev je potřeba se zabývat prakticky neustále. Jestliže svítí červené světlo na bílý povrch, objekt se jeví červeně. Stejným způsobem osvětlený modrý objekt prakticky není vidět, protože veškeré dopadající červené světlo je pohlceno. Zrcadlová složka však není závislá na barvě povrchu tělesa a na modrém lesklém tělese zpozorujeme červený odlesk.

Difúzní složka světla (barva povrchu) D se získá vynásobením složek barevného RGB vektoru barvy světla C_l a RGB vektoru difúzní odrazivosti povrchu D_s. Spekulární složka S je určena složkovým součinem barvy světla a spekulární odrazivosti („barvy“) povrchu S_s, která se dá rovněž vyjádřit jako RGB vektor nezávislý na difúzní odrazivosti povrchu D_s. (I v praktickém životě si můžeme všimnout, že čočky brýlí s antireflexní úpravou, ačkoliv jsou samy čiré, odrážejí při bílém dopadajícím světle jen některé barvy, respektive vlnové délky světla.)

Difúzní a zrcadlové složky D aS se vynásobí difúzní (I_d) a spekulární (I_s) intenzitou, sečtou se dohromady a dále se k nim (viz dále) případně přičítá barva odraženého (*reflected*) paprsku C_f a paprsku lomeného C_r (*refracted*).

$$C = I_d \cdot D + I_s \cdot S + I_s \cdot C_f + C_r$$

Všechny výpočty se provádějí pro každý světelný zdroj zvlášť a nakonec se sečtou. Veškeré výpočty jsou prováděny v RGB barevném systému, který je implementovatelný bez složitých přepočtů a technicky přímo podporovaný.

3.5.3 Odrazy

Odrazy od povrchu těles nejsou například při zobrazování lékařských dat přímo užitečné, ale měly svůj význam pro analýzu a testování výpočtu normál k povrchu, neboť odrazy jsou jimi bezprostředně ovlivněny. Objemová data navíc nejsou omezena jen na data lékařská.

Praktický výpočet probíhá tak, že k získané barvě povrchu tělesa je přičtena barva odraženého paprsku (také ona musí být vynásobena RGB vektorem zrcadlové charakteristiky povrchu S_s). Jeho směr R_v se vypočítá stejným způsobem jako při určování spekulární složky osvětlení, pouze místo vektoru L použijeme vektor V směřující od průsečíku k pozici pozorovatele.

$$R_v = 2 \cdot (V \cdot N) \cdot N - V$$

3.5.4 Stíny

Někdy jsou stíny ve vizualizaci dat považovány za jistý nadstandard, ale pravdou je, že jejich přítomnost nebo nepřítomnost velkou měrou ovlivňuje prostorový vjem ze zobrazované scény.

Při zobrazování metodou sledování paprsku se pro každý průsečík s povrchem vyšle takzvaný stínový paprsek ke každému zdroji světla. Jestliže stínový paprsek dříve, než dosáhne světelného zdroje, zasáhne jiný objekt, znamená to, že tento světelný zdroj není z místa průsečíku přímo viditelný, a bod na povrchu jím není přímo osvětlen. Při vyhodnocování osvětlení je světelný příspěvek tohoto zdroje zanedbán.

3.5.5 Depth cueing

Depth cueing je metoda zobrazování, která se obejde bez určování normály k povrchu tělesa. Rozhodujícím údajem je totiž vzdálenost průsečíku od pozorovatele. Konvenčně čím je průsečík s povrchem bližší, tím má světlejší barvu, a naopak nejvzdálenější oblasti jsou nejtmavší.

3.5.6 Chroma depth

Určitým zdokonalením předchozí metody je barevný způsob mapování hloubkové informace. Ve spojení se speciálními brýlemi získáme pseudo-stereoskopický obrázek. Tento mechanismus funguje i v kombinaci s výpočty osvětlení a stínů. Vnímaný prostorový efekt je potom velmi dobrý.

3.6 Následné zpracování získaného obrazu

V reálném světě je rozsah jasů, rozdíl mezi světlem a stínem mnohem větší, než můžeme zobrazit na monitoru počítače nebo jakémkoliv jiném výstupním zařízení. Můžeme-li reprodukovat pouze omezený rozsah jasů, nabízí se nám pouze dvě cesty:

- pečlivě vybrat polohu, barvu a intenzitu světelných zdrojů a vlastnosti materiálů tak, aby k přesvětlení nedocházelo, nebo

- použít mechanismus, který nám umožní modelově vypočítaný jas přepočítat do zobrazitelného rozsahu.

První varianta je velmi náročná a její princip ani získaný výsledek příliš neodpovídá úsilí o realistické zobrazování. Podívejme se blíže na druhý bod.

Jedním z mechanismů je určit nějaký horní práh a jestliže jej výpočtem získaný jas překročí, zobrazíme pouze onu maximální hodnotu. Tento způsob se nazývá *ořezávání* (*clamping*) a je mimo jiné také zahrnut ve standardu OpenGL. Takto zobrazené obrázky trpí nenapravitelným přesvětlením v těsné blízkosti jasných zdrojů světla nebo jejich zrcadlových odrazů.

Alternativou je *lineární mapování* rozsahu všech použitých jasů na interval zobrazitelných jasů. Ve velké většině případů je přitom důsledkem nezanedbatelná ztráta detailů ve stínech a celkově velmi tmavý výsledek přičemž zřetelné jsou pouze velmi jasné části snímku.

V praxi není intenzita světla téměř ničím omezena. Metoda sledování paprsku také funguje tak, že příspěvky osvětlení všech světelných zdrojů se stejně jako v reálném světě sčítají. Sčítají se jejich difúzní a spekulární složky, přičítají se k nim barevné příspěvky odražených paprsků a může se snadno stát, že celková intenzita světla překročí maximální zobrazitelný rozsah.

Kupodivu například obyčejná (resp. dobře vyvolaná) fotografie si s problémem velkého rozpětí jasů poradí velmi dobře a žádnou z popisovaných nectností netrpí i když fotografický papír má rozsah jasů omezený mnohem více než třeba monitor počítače. Pokusíme se využít mechanismů, které probíhají při expozici filmu pro naše účely.

3.6.1 Expoziční funkce

Míra zčernání vyvolaného negativu je ovlivněna dvěma hlavními faktory. Citlivostí filmové emulze a množstvím dopadajícího světla.

Dejme tomu, že dopadem určitého množství světla na povrch filmu se stane, že polovina chemických sloučenin emulze se rozpadne a místo na negativu bude světlejší. Jestliže na stejné místo dopadne shodné množství světla podruhé, opět bude reagovat polovina sloučenin, ale tentokrát už jenom těch zbylých. Rychlost zčernání tedy exponenciálně klesá, ačkoliv množství dopadajícího světla je stále stejné.

Tento proces lze ve zjednodušené formě matematicky modelovat funkcí

$$i = 1 - e^{-l_f},$$

kde l_f je množství světla dopadajícího na fiktivní film a i je výsledná intenzita.

Výpočtem získáme hodnotu v rozsahu $\langle 0, 1 \rangle$. V analogii s expozičním mechanismem fotografického přístroje zavedeme proměnnou k, která bude ovlivňovat množství světla dopadajícího na film a tedy

$$l_f = k \cdot l,$$

kde l je přímý výstup z raytraceru (světlo scény).

Popsanou transformaci je v případě barevného výstupu třeba provést pro každou barevnou složku RGB odděleně. Zde se přímo nabízí možnost použít pro každou barevnou složku jinou proměnnou k a tedy možnost případné barevné korekce, jestliže to bude výstupní zařízení vyžadovat.

3.6.2 Gamma korekce

Dalším faktorem ovlivňujícím kvalitu obrazového výstupu je nelinearita monitoru. Lineární změna napětí na elektrodách katodové trubice neodpovídá lineární změně jasu na stínítku obrazovky, ale tato změna je přibližně kvadratická. Důsledkem je výrazná ztráta kresby v tmavých částech obrazu.

Každému monitoru a vlastně každému výstupnímu zařízení přísluší určitá hodnota tzv. *gamma korekce*. Hodnota 1 se používá pro zařízení, která tuto korekci nepotřebují.

Matematická transformace korigující nelinearitu se řídí následujícím předpisem

$$i_p = i^{1/g},$$

kde i_p je přepočítaná hodnota jasu pixelu, i je požadovaný jas a g je hodnota korekce. Bývá uvedena v uživatelských příručkách k periferiím nebo se dá uživatelsky nastavit.

Každá RGB barevná složka musí být opět transformována zvlášť.

Kapitola 4

Modifikace základního algoritmu

Průzkum dostupné literatury ukázal, že vědecký svět se při vizualizaci objemových dat orientuje zejména na raycasting, zatímco pro zobrazování isoploch se běžně používají geometrické reprezentace zejména v kombinacích s metodami, které snižují složitost takto získaných povrchových sítí. Raytracing přitom zůstává poněkud na okraji zájmu. Nejpádnější příčiny tohoto stavu jsou zřejmě následující:

- při vhodné volbě parametrů je raycasting schopen do značné míry nahradit raytracing
- hardwarové akcelerátory přítomné i v levnějších grafických kartách velmi výrazně urychlují zpracování geometricky reprezentovaných isoploch

Odpovědí na první argument je kombinace obou přístupů. Raycasting i raytracing si jsou natolik podobné, že se jejich symbióza sama nabízí, zatímco rychlý nárůst výkonu pracovních stanic a zejména paralelní zpracování slibuje eliminovat druhý argument. V okamžiku, kdy bude možné interaktivní zobrazování přímými metodami v reálném čase (a nyní už je [7]), vynikne neobyčejná pružnost přímých metod ve srovnání s geometickými povrchovými reprezentacemi.

Ačkoliv je snaha o urychlení zkoumáním *koherence*, tedy závislostí a podobností v jednotlivých částech a mezi jednotlivými částmi zobrazovacího procesu, zaměřena primárně na raycasting, mnohé z níže uvedených algoritmů se dají adaptovat nebo mohou být inspirací pro další výzkum a použití v oblasti raytracingu isoploch.

4.1 Obrazová (pixelová) koherence

4.1.1 Postupné zjemňování

Anglický termín *progressive refinement* označuje metodu vykreslující obraz v několika průchodech.

Tradiční raytracing prochází obraz pixel za pixelem a pro každý vyšle jeden nebo více paprsků do scény. Adaptivní modifikace tohoto postupu prochází obraz s určitým krokem, například 4, a hodnoty mezi vzorky buď vyplní barvou jednoho ze sousedních vzorků nebo jejich barvy interpoluje.

Při druhém průchodu s polovičním krokem algoritmus dopočítá mezihodnoty. Třetí průchod dopočítá stejným způsobem poslední chybějící pixely.

Obraz po prvním průchodu je sice velmi hrubý a neobsahuje příliš mnoho detailů, ale jeho zobrazení trvá pouze šestnáctinu (v případě antialiasingu ještě mnohem méně) doby potřebné pro úplný výpočet.

Metoda postupného zjemňování se proto hodí zejména pro interaktivní práci. Uživatel se během prvního průchodu stačí zorientovat, případně upravit například polohu kamery a potom stačí pouze počkat tak dlouho, dokud obraz nedosáhne požadované úrovně podrobností.

4.1.2 Adaptivní vzorkování obrazu

Postřeh, že sousední pixely digitálního obrazu mají velmi často podobnou barvu, vede k metodě, jež se nazývá *adaptivní vzorkování obrazu* (*adaptive screen sampling*).

Vlastně jde o úpravu metody postupného zjemňování. Přívlastek adaptivní má proto, že při po sobě následujících průchodech zjemňuje (dopočítává mezihodnoty) pouze tam, kde rozdíl sousedních barev vypočítaných v předchozích krocích přesáhne určitou mez.

Nastane-li situace, že jas sousedních pixelů je i po posledním běhu příliš veliký, algoritmus může pokračovat dál a vyslat do kritické oblasti další paprsky, aby eliminoval aliasing.

Nevýhodou, která může být někdy závažná, je nebezpečí zanedbání malých detailů v jednolitých plochách. Ideální použití je zřejmě v oblasti zobrazování náhledu například při hledání vhodné pozice kamery. Je-li uživatel s nastavením kamery spokojen, může se vypočítat obrázek v plném rozlišení.

4.2 Paprsková koherence

Dva základní přístupy definují dva základní způsoby generování dráhy paprsku scénou:

- diskrétní způsob detekuje paprskem protnuté buňky pomocí algoritmů pro generování čar v diskrétním třírozměrném prostoru a potom zpracovává data na úrovni buněk
- druhý způsob využívá analytické reprezentace paprsku a o buňky se příliš nestará, jedná jen s jednotlivými vzorky

4.2.1 Paralelní projekce

Buňky, jimiž paprsek prochází tvoří jeho diskrétní aproximaci. V případě paralelní projekce jsou tyto paprsky rovnoběžné. Díky tomu, že jsou shodné a mají pouze posunuté počátky, stačí vytvořit *šablonu* (*template*), podle které budou jednotlivé paprsky procházet objemem. Ta může obsahovat například rozdíl indexů k následující protnuté buňce.

Stejně jako v případě podobnosti paprsků, je-li zachována stejná vzorkovací frekvence, dochází ke vzorkování na stejných (nebo po určité době se opakujících místech v buňkách) a je tedy možné vytvořit tzv. *sampling point templates*, tedy *šablony vzorkovacích* *míst* a podle způsobu použité interpolace lze některé hodnoty pro konkrétní oblast dat předpočítat.

Jiný způsob urychlení zavislý na této projekci používají metody, jež nejprve přetransformují data tak, aby je mohly procházet ve směru os. Šablona paprsku tak degraduje na přímku rovnoběžnou s některou z os a průchod podél něj je realizován pouhou změnou jediné souřadnice.

4.2.2 Ostatní projekce

Narozdíl od paralelní projekce, ve všech ostatních projekcích má každý paprsek obecně různý směr. Jelikož zde již tvorba šablon pro jednotlivé paprsky není příliš opodstatněná, přicházejí ke slovu algoritmy pro rychlé generování přímých čar v diskrétním prostředí. Algoritmů bylo vyvinuto několik na různých principech a dosáhnutá urychlení ve srovnání s původním Bressenhamovým algoritmem se počítají v řádech desítek procent [8].

Další záludností, se kterou je třeba počítat, je rozbíhavost paprsků, která může způsobit vynechání některých důležitých vzorků objemových dat (dochází zde k tzv. *podvzorkování*, spojenému s *aliasingem*).

Řešením může být:

- vyslat více paprsků na jeden pixel. V bližších oblastech dochází k *nadvzorkování* (*oversampling*), tedy k nadbytečnému množství vzorků a oblast podvzorkování je posunuta do větší vzdálenosti od pozorovatele,
- rozdvojit paprsek, jestliže divergence sousedních paprsků překročí vzdálenost mezi sousedními vzorky. Jednoznačnou výhodou je to, že počet paprsků je úměrný charakteru dat a k dělení dochází až v případě, že je to skutečně zapotřebí, že to má praktický význam, například při odrazu od konvexní plochy. Nevýhodou je nutnost neustále udržovat přehled o divergenci sousedních paprsků,
- vyslat paprsky do rohů pixelů a procházet a integrovat celou pyramidu mezi rohy každého pixelu (metoda vhodná pro raycasting, kdy se dají integrovat hodnoty pohltivosti v takto vymezeném objemu).

4.3 Ray rendering

Lze urychlovat i samotný průchod podél paprsku. Ne všechny odebrané vzorky jsou totiž stejně důležité a je tedy možné:

- v závislosti na vzdálenosti od pozorovatele nebo akumulované hodnotě nebo velikosti gradientu lze vzorkovat s měnící se frekvencí (*adaptive sampling*)
- vzdálené oblasti scény zanedbat jestliže akumulovaná hodnota průhlednosti dosáhne nějakého prahu a příspěvky dalších vzorků nejsou pozorovatelné (*adaptive ray termination*)

Větší homogenní oblasti mohou být úspěšně předpočítány a:

- do každého voxelu lze uložit hodnotu předpočítanou pro dané okolí a velikost předpočítané oblasti (o tuto vzdálenost můžeme postoupit pro další vzorek)
- paprsek lze rozdělit na segmenty, které prochází homogenními oblastmi, a ty je možno optimalizovat a například přidělit ke zpracování různým procesorům.

K detekci oblastí homogenity lze dobře využít v další části zmíněné hierarchické stromové struktury.

4.4 Koherence dat

4.4.1 Pomocné datové reprezentace

Pomocné datové reprezentace mají zpravidla za úkol eliminovat nezajímavé oblasti dat, urychlit nějakým způsobem průchod datovým objemem, případně umožňovat adaptivní změnu úrovně podrobností (*Level Of Detail*, LOD). Zhruba se dají rozdělit na metody, které využívají dělení datového prostoru pomocí stromové struktury, a ty ostatní. Často umožňují v případě raycastingu eliminovat nepříjemnou rozbíhavost paprsků při perspektivním promítání.

Stromové struktury mohou obsahovat mnoho nejrůznější informace. Nejenom přímo hodnotu nebo průměrnou hodnotu náležející k podoblasti příslušného listu, ale také minimum, maximum nebo interval hodnot příslušejících dané podoblasti (struktury jsou potom nazvané *minmax tree*, *average tree*, *range tree*,...).

Zobrazování dat potom může probíhat přímo z těchto struktur, nebo mohou být použity pro rychlé určení polohy významných oblastí ve skutečných datech.

Jednou z možností akcelerace je tedy procházet strom hodnot v hloubce odpovídající požadované úrovni zobrazených detailů. Úroveň podrobností se takto dá měnit i v závislosti na vzdálenosti odebíraných vzorků od pozorovatele.

Shear-warp [9] transformace pomocí několika zřetězených operací zkosení transformuje datový soubor tak, že jej lze procházet ve směru os a pro úsporné zakódování polohy zobrazované isoplochy používá variantu kódování délky běhu (*Run Length Encoding*, RLE). Je to jedna z metod, jež umožňují interaktivní práci bez HW podpory, ale zobrazování je omezeno pouze na isoplochy a paralelní projekci a v základní verzi neumožňuje implementaci a použití složitějších osvětlovacích modelů.

4.4.2 Eliminace nezajímavých oblastí

Jelikož často poměrně velkou část dat tvoří prázdný prostor, respektive oblasti, které nejsou pro zobrazování významné nebo zajímavé, velká skupina metod je založena právě na urychlení zobrazování či přeskočení takovýchto oblastí.

K nestromovým strukturám můžeme počtat například PARC (*Polygon Assisted Raycasting*). PARC vytvoří polygonální obálku zajímavé oblasti a pro každý vyslaný paprsek nejprve testuje průsečíky s touto obálkou, případně začíná sledovat vržený paprsek až od hranice této oblasti.

Ke stejnému účelu se dá použít *oct-tree* nebo nějaká jeho modifikace.

Velmi podobné postupy se používají i při urychlení zobrazování homogenních oblastí dat. Hodnoty pro celé homogenní oblasti mohu být předpočítány a tím celý proces výrazně urychlen. Stromy jsou pouze jednou z mnoha možností, kam předpočítaná data uložit. Zajímavou možností jsou tzv. *proximity-clouds* [10]. Pomocná datová struktura má shodnou velikost jako zobrazovaný datový soubor, ale každý voxel obsahuje největší vzdálenost, o kterou je možno z daného bodu postoupit kterýmkoliv směrem bez nebezpečí, že mineme nějaký zajímavý detail. Spolu se vzdálenostními je opět možno ukládat předpočítanou hodnotu pro dané homogenní okolí.

4.5 Speciální metody

4.5.1 3D Raster Raytracing

Spíše než o raytracing volumetrických dat se jedná o způsob dělení prostoru, který je rozdělen na buňky a každá z nich obsahuje seznam všech těles (včetně objemových dat), jež ji protínají. Jednotlivé buňky scény jsou potom pro každý vržený paprsek procházeny diskrétním způsobem a průsečíky jsou analyticky vypočítány pouze u těch těles, které se nacházejí v příslušných buňkách na dráze paprsku. Výrazné urychlení je pozorovatelné zejména při zobrazování složitých scén s velkým množstvím objektů.

4.5.2 Volumetric Radiosity

V procesu zkoumání jsou i metody volumetrické radiozity. Radiozitní metoda je založena především na dělení prostoru a výpočtu vzájemných energetických interakcí (vyzáření, odraz, absorpce). Klasické algoritmy jsou omezeny na rovinné plošky a způsob jejich dělení je zpravidla počítán adaptivně. Volumetrická data jsou rozdělena již v samotném principu, umožňují reprezentaci ploch a proto se použití radiozitních algoritmů samo nabízí.

Mezi získané výhody patří hlavně realističtější způsob výpočtu osvětlení. Matematická zátěž se dá do značné míry přesunout do předzpracování a vlastní zobrazení se tím výrazně urychlí.

4.5.3 LightField Rendering

Z úplně jiného konce se na interaktivní zobrazování objemových dat dívají autoři, kteří implementovali tzv. *lightfield rendering* neboli *zobrazování světelného pole*. Představme si celou scénu uzavřenou do myšlené koule. Povrch této koule rovnoměrně pokryjeme trojúhelníkovou sítí. Postupně do každého uzlu této sítě umístíme kameru a některou z běžných metod vypočítáme obraz scény uvnitř koule a ten uložíme do paměti.

Po předpočítání všech pohledů se zobrazení z libovolného bodu na povrchu koule vypočítá interpolací ze tří nejbližších předpočítaných pohledů.

4.5.4 Paralelní zpracování

Po již relativně důkladně zpracované oblasti základních algoritmů se pozornost v posledních letech zaměřuje na výzkum paralelismu a paralelizace při interaktivním zobrazování velkých objemů dat. Zatím je to stále jediný způsob jak dosáhnout interaktivních zobrazovacích časů bez nějakých zásadních omezení týkajících se velikosti dat, projekce nebo podobných parametrů zobrazování.

Kapitola 5

Distance transformation

Výběr z mnoha výše uvedených akceleračních technik nebyl jednoduchý. To, že volba padla na *proximity-clouds*, je důsledkem jejích několika zajímavých vlastností:

- metoda je směrově nezávislá – je možné jednoduše implementovat pokročilejší zobrazovací techniky jako stíny nebo odrazy
- zachovává rozlišení dat
- víceúčelovost – dá se totiž použít nejenom pro akceleraci, ale zároveň i pro reprezentaci isoplochy (vystačíme jen s jedním datovým souborem)
- dá se využít i pro případnou kombinaci raytracingu s raycastingem

Základem použité metody je tzv. *distance transformation*, neboli *vzdálenostní transformace*. Matematicky je definována jako funkce jedné, dvou nebo tří nezávislých proměnných podle dimenze prostoru a její hodnota vyjadřuje minimální vzdálenost od nejbližšího bodu z předem dané množiny. Množina, jejíž transformaci hledáme, může obsahovat prakticky libovolný počet bodů a nemusí být nutně spojitá. Body množiny mohou být rozptýlené, mohou tvořit křivky, plochy nebo tělesa.

Výsledkem transformace ve spojitém prostoru je spojitá funkce, která obecně není všude diferencovatelná, laicky řečeno může obsahovat zlomy. Konkrétní průběh je jednoznačně určen transformovanou množinou bodů a použitou *metrikou prostoru*.

Pro měření vzdálenosti se nejčastěji používá *euklidovská metrika*, jež definuje to, co si běžně představíme pod pojmem „vzdálenost dvou bodů“, tedy délku úsečky, která tyto body spojuje. Protože prostor objemových dat je diskrétní a má některé specifické vlastnosti, používají se v praxi i jiné než euklidovské definice vzdálenosti zpravidla vycházející z různých definicí *sousednosti*, resp. *diskrétního okolí* bodu.

5.1 Diskrétní okolí bodu

V třírozměrné mřížce lze definovat několik typů sousednosti. Představme si prostor složený z krychlových buněk. Ke každé buňce diskrétního prostoru nalezneme šest sousedů, kteří s ní mají společnou celou stěnu. Těchto šest nejbližších sousedů nazveme podle jejich

	1	
1	0	1
	1	

1	1	1
1	0	1
1	1	1

Obrázek 5.1: Dvě definice 2D diskrétního okolí (čtyřokolí a osmiokolí)

počtu *šestiokolím*. Rozšíříme toto okolí dále a zahrneme do něj navíc všechny buňky, které s danou buňkou sdílejí hranu. Krychle má dvanáct hran a šestiokolí tak naroste na *osmnáctiokolí*. K poslední běžně používané definici patří *dvacetišestiokolí*. Obsahuje všechny buňky, které se zkoumanou buňkou přímo sousedí přes stěnu, hranu nebo vrchol. Na obrázku 5.1 je uvedena 2D analogie diskrétního okolí *čtyřokolí* a *osmiokolí*.

Obrázek 5.6 znázorňuje geometrické vztahy mezi jednotlivými diskrétními okolími. Vyobrazené plochy mají vzdálenost 10 od středu. Sousední buňky jsou obarveny rozdílnými barvami.

Tato okolí jsou základem některých diskrétních transformací včetně transformace vzdálenostní, jejíž varianty budou popsány dále.

5.2 Varianty transformace

5.2.1 Euclidean Distance Transformation

Euklidovská vzdálenost d dvou bodů o souřadnicích $[x_0, y_0, z_0]$ a $[x_1, y_1, z_1]$ se obejde bez definice diskrétního okolí a vypočítá se klasicky přes Pythagorovu větu z rozdílu souřadnic.

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$$

Přesný výpočet celé transformace tímto způsobem je velmi náročný neboť vzdálenost je potřeba vypočítat pro každou dvojici bodů datového prostoru. Řešení brutální silou předpokládá u každého voxelu, který není v transformované množině výpočet vzdálenosti ke každému bodu, jenž do ní patří. Po jejich vzájemném porovnání je hodnotou vzdálenostní funkce ta nejmenší z nich. Poněkud méně naivní implementace by usela použít nějakou heuristiku k určení nejbližšího bodu transformované množiny.

Ačkoliv se dostupná literatura zmiňuje o používání euklidovské vzdálenosti, nepodařilo se zjistit v praxi používaný způsob jejího výpočtu. Rovněž bývá zmíněna i *diskrétní euklidovská transformace*, která získané hodnoty vzdálenosti ukládá jako celá čísla (zřejmě zaokrouhleně), ale podrobnosti rovněž nejsou k dispozici.

5.2.2 Discrete Distance Transformation

Ať už si vybereme kterékoliv diskrétní okolí, všechny do něj zahrnuté buňky mají z definice vzdálenost od centrálního (vztažného) bodu (buňky) rovnou jedné, a může měnit pouze po násobcích jedné (obr. 5.2). Chceme-li určit vzdálenost buňky, která do okolí nespadá, postupujeme inkrementálně:

	2	2	2	2	2	2	2	2	2
a)	1	1	2	1	1	2	2	2	2
	0	1	2	1	1	2	2	2	2

	2	2	2	2	2	3	2	3	4
b)	1	1	2	1	2	2	2	3	3
	0	1	2	1	1	2	2	2	2

	2	3	4	3	4	5	4	5	6
c)	1	2	3	2	3	4	3	4	5
	0	1	2	1	2	3	2	3	4

Obrázek 5.2: Výseky 3D diskrétních okolí. Vrstvy jsou paralelní a vztažný bod má hodnotu 0. a) dvacetišestiokolí, b) osmnáctiokolí, c) šestiokolí

- prvním krokem je určení vzdáleností buněk ve vybraném jednotkovém okolí
- v dalším kroku zpracujeme buňky, které mají vzdálenost jedna od těch z okolí z předchozího kroku, a dosud nebyly zpracovány; jejich vzdálenost je o jedničku větší

Takto pokračujeme dál, dokud stále se rozšiřující okolí nezasáhne to místo, jehož vzdálenost chceme určit, nebo dokud nejsou zpracovány všechny buňky datového souboru.

Jiná varianta výpočtu principiálně vychází z metody v počítačové grafice známé jako *semínkové plnění* (*seed fill*), z níž převzala způsob procházení prostorem. Jen se v našem případě nejedná o přebarvování uzavřené oblasti. Teoreticky by se jako počátek algoritmu („semínko“) měl použít každý bod transformované množiny.

Zde může nastat situace, že pro jeden bod prostoru vyjdou dvě různé vzdálenosti. Jedná se o vzdálenosti k různým bodům množiny, nebo o jinou cestu ke stejnému bodu. Použijeme podle definice samozřejmě tu menší vzdálenost. Při výpočtu nesmíme zapomenout zkontrolovat, jestli právě počítaná cesta k bodům vypočítaným v předchozích krocích není kratší. Nestačí tedy zjistit, zda body byly zpracovány, ale je třeba kontrolovat, jestli aktuální cesta nemá menší délku a případně je přepočítat.

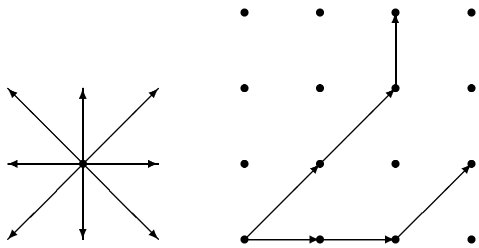
Posledně jmenovaný postup je náročnější než první popsaná varianta a je zde uveden pouze pro úplnost.

5.2.3 Chamfer Distance Transformation

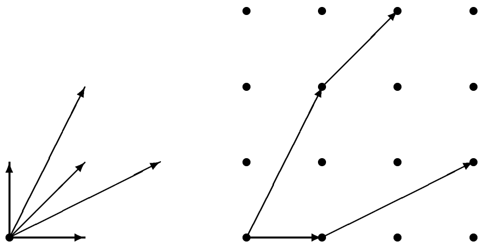
Chamferová vzdálenost je aproximací euklidovské vzdálenosti. Definuje rozsáhlejší okolí a přímá spojnice je nahrazena lineární kombinací několika vektorů předem daných orientací a délek.

V nejjednodušším případě se jedná o vektory (0,0,1), (0,1,1), (1,1,1) včetně jejich symetrických ekvivalentů (obr. 5.3). Jako vzdálenost je brán součet jejich euklidovských délek, tedy 1, $\sqrt{2}$, a $\sqrt{3}$. Například vektor (1,1,2) skutečné délky 2,45 je nahrazen vektory (1,1,1) a (0,0,1) o celkové délce $1+\sqrt{3} = 2,732$. Diskrétní transformace by určila jeho délku jako 4, 3 nebo 2 podle použitého okolí.

Potřebujeme-li přesnější aproximaci, rozšíříme množinu vektorů například na (0,0,1), (0,1,1), (1,1,1), (0,1,2), (1,1,2), (1,2,2), (1,2,3), (0,0,2), (0,0,3), (1,1,3), (0,2,3), (2,2,3), (2,3,3) a jejich symetrické obrazy (obr. 5.4). Výběr množiny vektorů může být přizpůsoben aktuálním potřebám.



Obrázek 5.3: Sada chamferových vektorů a způsob konstrukce vzdálenosti v rovině



Obrázek 5.4: Rozšířená sada chamferových vektorů

Výsledkem použití chamferové vzdálenosti je mnohem přesnější aproximace euklidovské vzdálenosti než při použití diskrétní transformace při zachování inkrementálního postupu výpočtu. Nevýhodou je nutnost uchování vzdálenosti v zobrazení s pohyblivou řádovou čárkou.

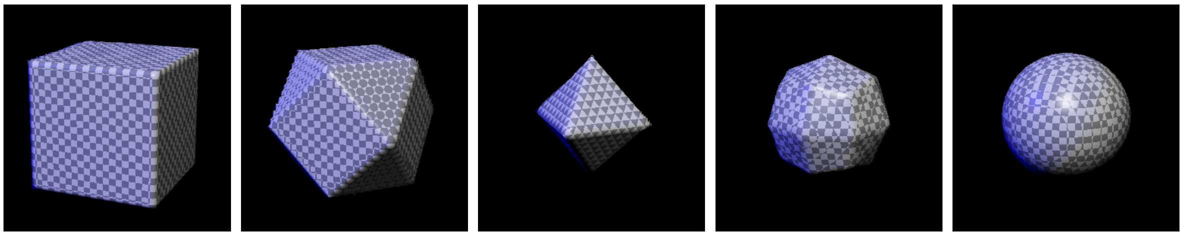
5.2.4 ⟨3,4,5⟩ Weighted Distance Transformation

⟨3,4,5⟩ Weighted Distance Transform (*vážená DT*) je kompromisem mezi chamferovou a vzdáleností, neboť zachovává diskrétní charakter uchovávaných vzdáleností, ale každá vzdálenost je vážená faktorem tři. Jedná se tedy o formu zobrazení v pevné řádové čárce.

Buňka sousedící přes stěnu má vzdálenost tři, soused přes hranu má vzdálenost čtyři a buňka sdílející pouze vrchol má vzdálenost pět (obr. 5.5). V této formě jsou vypočítány všechny vzdálenosti, ale při vlastním použití je vzdálenost vydělena faktorem tří. Skutečné hodnoty vzdálenosti všech tří typů sousedních buněk jsou 3/3 (přes stěnu), 4/3 (přes hranu) a 5/3 (přes vrchol).

5	4	5	4	3	4	5	4	5
4	3	4	3	0	3	4	3	4
5	4	5	4	3	4	5	4	5

Obrázek 5.5: ⟨3,4,5⟩ vážené vzdálenosti v okolí $3 \times 3 \times 3$



a) b) c) d) e)

Obrázek 5.6: Plochy tvořené body s konstantní vzdáleností od centrálního bodu se pro jednotlivá okolí liší. a) 26-okolí, b) 18-okolí, c) 6-okolí, d) základní chamferová vzdálenost, e) odpovídající euklidovská vzdálenost.

5.3 Výpočet transformace

Možnost inkrementálního výpočtu celé transformace je značnou výhodou oproti použití brutální síly, ale ani způsob popsaný v odstavci o diskrétní transformaci není nenáročný, protože pro efektivní výpočet je potřeba uchovat informaci o všech buňkách zpracovaných v předchozím kroku. Pro inkrementální výpočet vzdálenostní transformace byl proto vyvinut dvouprůchodový algoritmus podobný postupu při diskrétních transformacích obrazu.

Máme definovanou masku o rozměrech, které odpovídají maximálním použitým vektorům. V případě diskrétní transformace a $\langle 3, 4, 5 \rangle$ vážené nebo základní chamferové transformace má maska rozměry 3×3 a vztažný bod, jenž je zároveň počátkem všech vektorů, je v jejím středu. V případě použití větších vektorů je musí maska do sebe pojmout všechny.

Na místech, které odpovídají koncům jednotlivých vektorů se nachází hodnoty jejich délek, se kterými bude při transformaci počítáno.

5.3.1 Inicializace

V inicializační části je nejprvě potřeba alokovat v paměti objem dat shodné velikosti jako transformovaný objem dat, nazvěme jej objemem vzdáleností. Celý objem vzdáleností je inicializován tak, aby na pozicích bodů, které patří do transformované množiny (v našem případě to mohou být například uzly v těsném sousedství isoplochy), byly uloženy nulové hodnoty. Vzdálenosti všech ostatních bodů jsou nastaveny na maximální možnou (požadovanou) hodnotu.

5.3.2 Dopředný běh

Při dopředném běhu posunujeme masku ve vzdálenostním souboru tak, aby její vztažný bod postupně prošel všemi jeho body. Pojmem *počátek* budeme dále rozumět vztažný bod masky. Pod pojmem *hodnota masky* rozumíme hodnotu uloženou v masce (má význam vzdálenosti bodu z těsného sousedství). Pojem *hodnota pozice* se vztahuje k hodnotě uložené ve vzdálenostním souboru.

Pro každou pozici opakujeme následující postup:

1. do pomocné proměnné *minimum* ulož hodnotu pozice počátku

	1
1	0

a)

0	1
1	

b)

1	1	1
1	0	

c)

	0	1
1	1	1

d)

Obrázek 5.7: Masky pro dopředný a), c) a zpětný b), d) běh 2D vzdálenostní transformace pro čtyřokolí a), b) a osmiokolí c), d). Dopředný běh postupuje zleva doprava a shora dolů, zpětný opačně.

2. pro každý bod masky, který nepřesahuje hranice datového prostoru, proveď následující činnosti:
   (a) sečti hodnotu pozice a hodnotu masky
   (b) je-li výsledek menší než *minimum*, potom jej ulož do proměnné *minimum*
3. do souboru vzdáleností na pozici počátku ulož hodnotu nalezeného minima

Jelikož se jedná o inkrementální postup, je nezbytné, aby byly zpracovány postupně:

- všechny voxely v každé řádce
- všechny řádky v každém řezu
- všechny řezy v celém objemu

5.3.3 Zpětný běh

Zpětný běh je naprosto shodný s dopředným pouze je potřeba obrátit pořadí řezů, řádek i voxelů. Pokud tedy byl soubor v prvním běhu procházen cyklem:

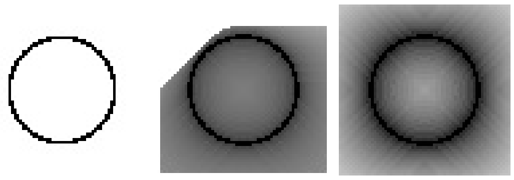
```
for (z = 0; z < max_z; z++)
  for (y = 0; y < max_y; y++)
    for (x = 0; x < max_x; x++)
```

zpětný chod bude korektní použitím cyklu:

```
for (z = max_z - 1; z >= 0; z--)
  for (y = max_y - 1; y >= 0; y--)
    for (x = max_x - 1; x >= 0; x--)
```

5.3.4 Optimalizace

Celý výpočet se dá jednoduše urychlit, když si uvědomíme, že při dopředném běhu není třeba kontrolovat pozice, které jsme ještě nezpracovali („leží před námi“). Při zpětném běhu platí stejný postřeh. Z tohoto důvodu je postup implementován tak, že pro dopředný a pro zpětný běh jsou definovány odlišné masky (obr. 5.7).



a) b) c)

Obrázek 5.8: Tři kroky vzdálenostní transformace (diskrétní 18 okolí). a) inicializace isoplochy, b) dopředný běh, c) zpětný běh

5.4 Kombinace existujících objemů

Máme-li k dispozici dva objemy vzdáleností a každý z nich se vztahuje k jiné isoploše, není za dodržení určitých podmínek problém tyto objemy zkombinovat do jediného souboru. Nejdůležitější podmínkou je nutnost, aby oba soubory měly shodné rozlišení (absolutní hodnota měřítka jednotlivých os). Není-li tomu tak, je potřeba objemová data převzorkovat.

Vlastní kombinace probíhá tak, že do výsledného souboru bude uloženo vždy minimum ze stejnolehlých hodnot obou souborů.

5.5 Použití pro akceleraci raytracingu

Tradiční verze raytracingu postupuje prostorem a s předem danou frekvencí vzorkuje hodnoty datového souboru. Jestliže v inicializační fázi transformace zahrneme všechna místa, kterými probíhá zobrazovaná isoplocha, do transformované množiny, po transformaci udávají získané hodnoty vzdálenost k nejbližšímu bodu isoplochy. Celou tuto vzdálenost můžeme přeskočit bez obav, že mineme důležitou oblast dat. Vzorkování začne probíhat až v těsné blízkosti isoplochy.

Nejlépe lze zakódovat jednu isoplochu. Pokud do jednoho vzdálenostního souboru zahrneme údaje pro více isoploch, metoda ztrácí efektivitu. Paprsek se i při průchodu v blízkosti nezobrazovaných isoploch zpomaluje, probíhá vzorkování a poté opět urychluje, aniž by se cokoliv zobrazilo. Krajním (a přesto stále velmi dobře použitelným) případem je situace, kdy v datech je zahrnuto tolik isoploch, že průchod paprsku je urychlován pouze v oblasti prázdného prostoru. Určíme-li jako hraniční isoplochu v lékařských datech například hodnotu hranice kůže, potom je paprsek urychlován pouze k této isoploše, a už zde začíná vzorkovat s plnou frekvencí, ačkoliv bude zobrazena třeba kost ve velké hloubce. Této technice se říka *space-leaping*, neboli *přeskakování prázdného prostoru*

Nevýhodou je paměťová náročnost. Ta je sice srovnatelná s tzv. *pyramidálními objemy dat* [12] obsahujícími datový soubor v několika exemplářích s postupně se snižujícím rozlišením, ale má možnost reprezentovat pouze omezené množství ploch.

5.6 Použití pro reprezentaci isoploch

Pro přesnou reprezentaci isoploch je potřeba, aby v místě, kudy prochází isoplocha, byl zachován gradient vzdálenostních dat. Tak tomu ovšem při použití klasické vzdálenostní transformace není. Vzdálenosti od isoplochy jsou ze všech směrů vždy kladné. Aby byl zachován gradient dat, vzdálenosti uvnitř isoplochy, tedy na místech, která obsahovala data větší než je hledaná isohodnota, necháme kladné, ale vzdálenosti vně isoplochy budeme ukládat záporné.

Z hlediska paprsku protínajícího isoplochu mohou mít hodnoty vzdáleností například následující průběh:

```
...-2, -1, 0, 1, 2...
```

Gradient zůstává zachován narozdíl od původní varianty:

```
...2, 1, 0, 1, 2...,
```

kdy je v místě isoplochy (hodnota 0) použitím metody centrální diference nemožné určit směr gradientu často nezbytného pro zobrazení.

Další podmínkou je dostatečná přesnost vzdálenostních hodnot. Dobrým řešením je uchovávat vzdálenosti v zobrazení s plovoucí řádovou čárkou. Experimentální výsledky ukazují, že stačí celočíselné hodnoty vážené faktorem řádu jednotek.

V dostupné literatuře nebyl nalezen způsob, jak konvertovat do vzdálenostní podoby isoplochu z existujících například CT snímků tak, aby v jednom objemu byly zachovány informace o gradientu a přesném průběhu isoplochy. Pro její exaktní reprezentaci je totiž potřeba subvoxelové přesnosti a v případě konverze z existujících dat je tato přesnost silně závislá na použité interpolační metodě. Proto se vzdálenostní objem používá pro reprezentaci objektů zejména při tvorbě syntetických objemů z geometrických reprezentací.

Mezi výhody jistě patří zahrnutí definice povrchu a akcelerační informace do jediného souboru dat. Rozměry a rozlišení mohou být přizpůsobeny charakteru obsažených objektů.

Zahrneme-li do jednoho objemu navzájem disjunktní objekty, problém nenastává. Je ale potřeba řešit situace, kdy je jeden objekt uvnitř jiného nebo kdy se jejich povrchy přibližují, dotýkají nebo protínají.

Kapitola 6

Výsledky

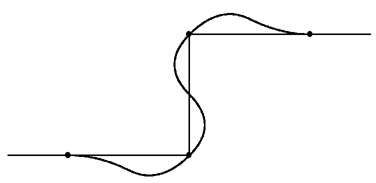
Prakticky všechny zde uvedené výsledky (vyjma akcelerační metody) mají význam až při zobrazování detailů velikosti řádově jednotek voxelů a v dostupné literatuře podobný výzkum dosud nebyl publikován.

6.1 Interpolace

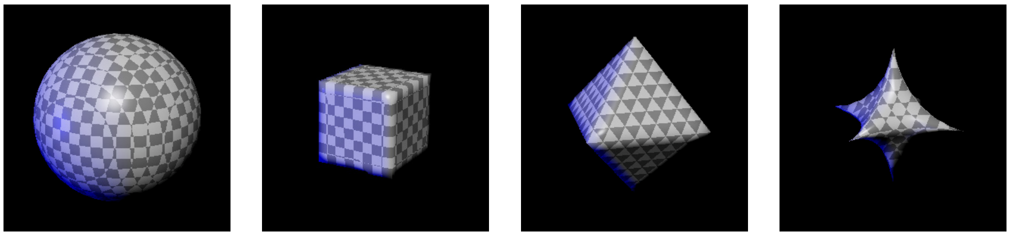
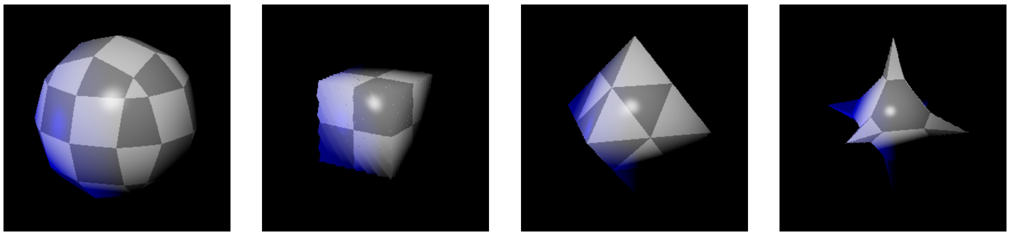
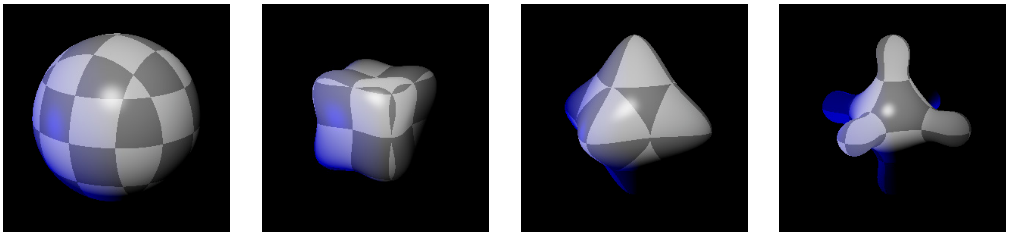
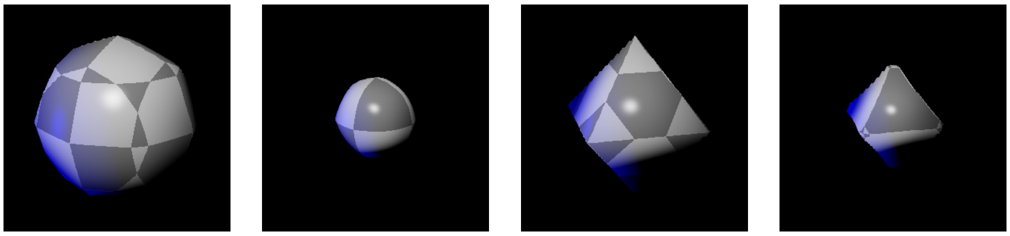
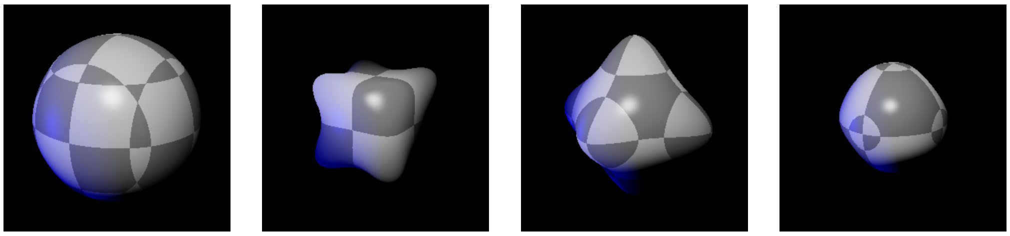
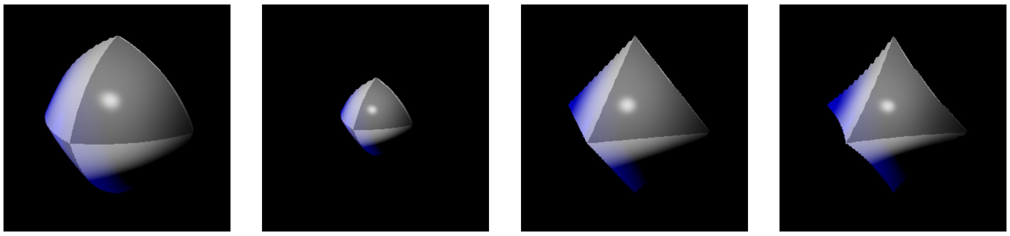
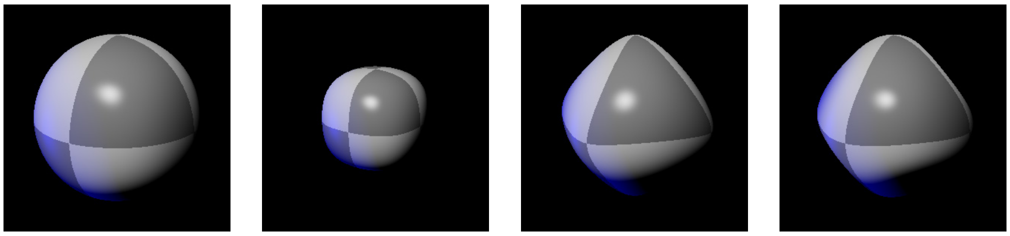
Interpolace má rozhodující vliv na tvar zobrazované isoplochy, jak je patrno z tabulky 6.1. Rozdíl mezi lineární a kubickou interpolací je patrný až při velkém zvětšení. Při kubické interpolaci je bráno v úvahu i okolí interpolovaného bodu, respektive gradient v nejbližších uzlech. V případech, kdy je změna gradientu spojitá v okolí bodu (koule), výsledky jsou velmi dobré, ale pro nespojitý průběh (těleso obsahuje hrany) je skutečný tvar deformován. Aproximace průběhu kubické interpolace je na obrázku 6.1. V těchto případech dává (při velkém zvětšení) lineární interpolace geometricky přijatelnější výsledky.

Objekty na obrázku 6.1 jsou obarveny tak, aby sousední buňky měly odlišné barvy. Objekty v první řádce jsou uvedeny pro srovnání. Díky většímu rozlišení velmi dobře aproximují tvary zakódovaných těles.

Při zobrazování větších objemů jsou jednotlivé voxely tak blízko sebe, že mezi lineární a kubickou interpolací není žádný optický rozdíl. Volba proto při běžném zobrazování oprávněně padá na lineární interpolaci. Kubická zřejmě bude mít místo pouze ve speciálních případech, zejména díky své výrazně větší výpočetní náročnosti.



Obrázek 6.1: Přibližný průběh kubické křivky podél hrany v pravidelné mřížce

zvětšení 2 rozměr 8 lineární	
zvětšení 8 rozměr 2 lineární	
zvětšení 8 rozměr 2 kubická	
zvětšení 10 rozměr 1,6 lineární	
zvětšení 10 rozměr 1,6 kubická	
zvětšení 16 rozměr 1 lineární	
zvětšení 16 rozměr 1 kubická	

Tabulka 6.1: Vliv interpolace na tvar isoplochy

6.2 Výpočet průsečíku

Vypočítaná poloha průsečíku závisí na interpolační funkci, vzorkovací frekvenci a způsobu numerické aproximace. Srovnání interpolačních funkcí je uvedeno v předchozí části. Ukázku vlivu zbývajících dvou faktorů můžeme vidět v tabulce 6.2.

Nízká vzorkovací frekvence bez aproximace průsečíku působí to, že objekt se zdá poskládán z řezů rovnoběžných s rovinou obrazu. Jejich vzájemná vzdálenost je rovna vzorkovacímu intervalu. K úplné redukci tohoto jevu je potřeba použít (u námi použitých syntetických dat) alespoň čtyři iterace binárního hledání nebo první odhad pomocí Newtonovy metody, který je s ní srovnatelný.

Při generování všech ostatních obrázků pro tuto práci byla použita Newtonova metoda a vzorkovací frekvence = 1. Výsledky jsou uspokojující.

6.3 Výpočty normály

Pozornou prohlídkou obrázků v tabulce 6.1 můžeme zjistit drobné nesrovnalosti v osvětlení jednotlivých objektů. Všimněme si blíže druhé řádky miniatur. Ačkoliv se jedná o objekty složené díky lineární interpolaci z rovinných plošek, tak u těch plošek, které jsou blízko hranice stínu, dochází ke světelnému přechodu. U rovinných plošek má normála ve všech bodech stejný směr a tedy by měla být na celé plošce stejná i intenzita odraženého světla, podobně jako je tomu na první řádce. Tělesa, kamera i zdroje světel zůstaly na stejném místě a přesto se osvětlení stěn liší.

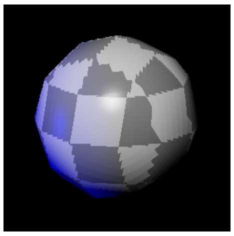
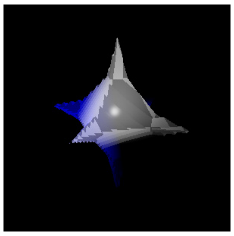
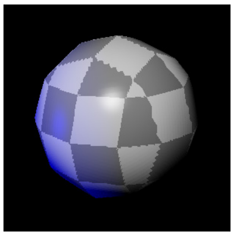
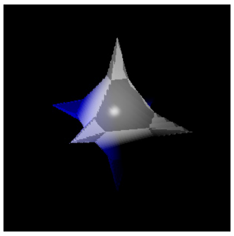
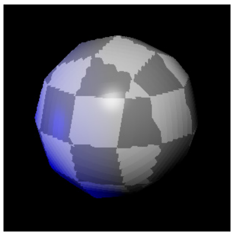
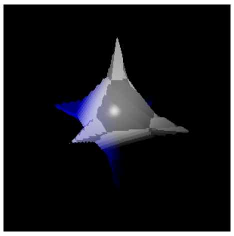
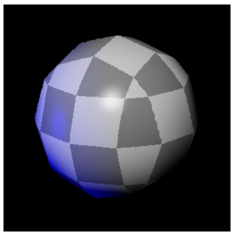
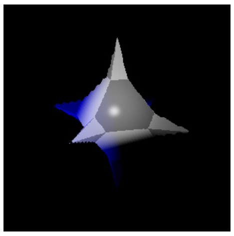
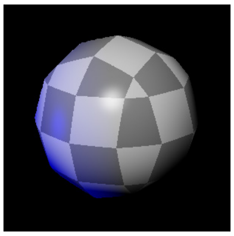
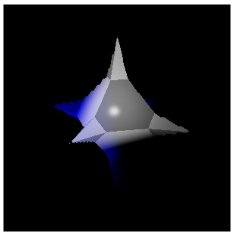
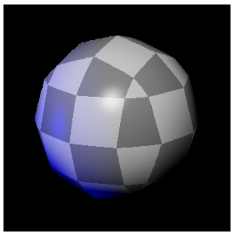
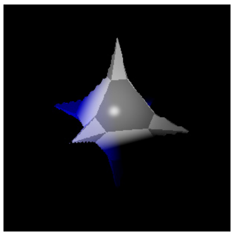
Důvod je podobný jako při deformaci některých těles kubickou interpolací. Plochy jsou interpolovány lineárně a tedy jsou rovinné, protože lineární interpolace vychází pouze z osmi vrcholů interpolované buňky. S normálovými vektory to tak ale není. Hodnoty normál ve vrcholech jsou symetrickou diferencí počítány z voxelů, které s vrcholem buňky sousedí. Stejně jako hodnotu uvnitř buňky interpolujeme z jejích vrcholů, tak také normálu interpolujeme z osmi normál ve vrcholech. Jenže každá normála je vypočítána ze svého (jiného) šestiokolí a tedy máme osm šestiokolí, ze kterých určíme její směr v daném bodě. Vyloučíme-li násobné voxely, dostaneme 28 různých ovlivňujících hodnot a právě to je příčinou toho, že se normály neshodují s reálným tvarem isoplochy – rovinná stěna se díky osvětlení jeví vyboulené. Tyto údaje platí zejména v blízkosti hran těles.

Řešením by mohl být výpočet normály ne interpolací z hodnot gradientu ve vrcholech buňky, ale symetrickou diferencí přímo z nějakého předem určeného malého δ-okolí interpolovaného bodu.

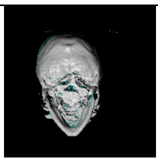
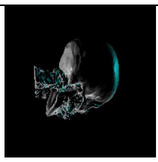
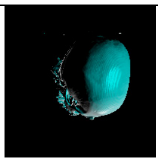
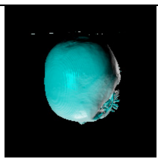
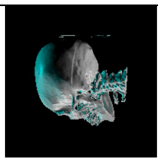
$$g(x,y,z) = \left(\frac{v(x+\delta,y,z)-v(x-\delta,y,z)}{2\delta}, \frac{v(x,y+\delta,z)-v(x,y-\delta,z)}{2\delta}, \frac{v(x,y,z+\delta)-v(x,y,z-\delta)}{2\delta}\right)$$

Velikost δ-okolí by potom záležela na aktuální potřebě.

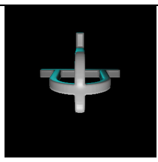
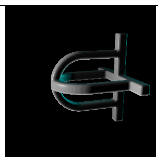
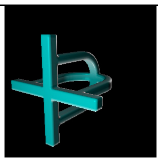
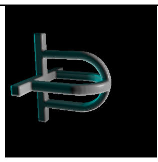
Jinou možností je použít pro odhad normály v průsečíku gradient paměti hloubky. Není totiž závislá na rozlišení dat, ale pouze na rozlišení výsledného obrazu.

Full f = 1		
Full f = 2		
Binary f = 1 i = 0		
Binary f = 1 i = 2		
Binary f = 1 i = 4		
Newton f = 1		

Tabulka 6.2: Porovnání způsobů aproximace průsečíků. (f – frekvence, i – počet iterací)

						
metoda	pre	1	2	3	4	5
Full 1		45,578	34,968	38,937	34,016	30,172
Full 2		88,078	69,031	76,188	66,062	58,500
Full 3		130,610	102,469	112,750	97,828	86,703
Dist 1	26,063	7,750	5,515	6,547	6,735	6,204
Dist 2	26,109	9,031	6,719	7,594	7,812	7,328
Dist 3	26,109	10,360	7,687	8,656	8,954	8,515

časy v jsou v sekundách, soubor: cthead.vol ($256 \times 256 \times 113$), iso: 100, okolí: 18, krátká animace: 5 snímků (300×300), vzorkovací frekvence: číslo v označení metody

						
metoda	pre	1	2	3	4	5
Full 1		12,922	12,047	10,828	10,735	12,156
Full 2		24,641	23,281	20,547	20,328	23,141
Full 3		35,907	33,344	29,250	28,937	33,171
Dist 1	0,891	3,875	3,969	4,047	4,063	4,000
Dist 2	0,890	4,844	4,656	4,750	4,750	4.719
Dist 3	0,891	5,500	5,343	5,516	5.500	5.406

časy v jsou v sekundách, soubor: syn_64.vol ($64 \times 64 \times 64$), iso: 100, okolí: 18, krátká animace: 5 snímků (300×300), vzorkovací frekvence: číslo v označení metody

Tabulka 6.3: Akcelerace pomocí *distance transformation*.

6.4 Akcelerace

Časové poměry jsou uvedeny v tabulce 6.3. Doba vzdálenostní transformace je zahrnuta do preprocessingu a uvedena ve sloupci „pre“. Její doba je závislá pouze na velikosti dat (lineárně) a nikoliv na složitosti objemové scény.

Metoda „Full“ vzorkuje podél celé dráhy paprsku plnou frekvencí. Metoda „Dist“ je urychlená pomocí vzdálenostní transformace a s plnou frekvencí vzorkuje pouze v intervalu vzdáleností $\langle -1, 1\rangle$ v okolí isoplochy.

Distance transformation urychluje průchod scénou 4–10 krát. Zda se vyplatí počítat transformaci pro jediný obrázek je třeba zvážit. Pro dva a více je to již výhodné.

Kapitola 7

Poznámky k implementaci

Všechna rozhodnutí ohledně implementace se řídila podle několika následujích bodů:

- při zobrazování preferovat kvalitu povrchu, jako doklad toho, že je možná poměrně přesná reprezentace isoploch
- umožnit zobrazování jak geometrických, tak volumetrických dat pro názornější srovnání
- ze zkoumaných metod pro urychlení vybrat pokud možno jednoduchou směrově nezávislou metodu, s možností vícenásobného využití
- zachovat vysokou modularitu systému pro snadné testování, úpravu a případné budoucí rozšíření zkoumaných postupů

7.1 Objektový návrh

Testování různých metod vyžaduje modulární návrh celého programového systému. S výhodou byl použit objektově orientovaný programovací styl, který již v samotném svém základu tuto modularitu předpokládá, umožňuje a podporuje. Základní principy tohoto programovacího stylu (zapouzdření a dědičnost) jsou těžištěm programového řešení této práce.

Použitým jazykem je C++, které nabízí všechny základní objektové nástroje a již je de facto standardem mezi OO jazyky a to i přes mnohou kritiku „nečistoty" jazykových konstrukcí. Při dodržení několika jednoduchých pravidel je možno se vyhnout všem kritizovaným nectnostem a dosáhnout elegance, která je zatím přisuzována jen Javě nebo „čistě objektovým jazykům" typu Eiffel či SmallTalk.

Principy objektového programování již přešly do obecného povědomí zejména s rozšířením prostředků typů Delphi, C++ Builder, Visual C++ a zejména díky popularitě jazyka Java. Existuje však jeden nástroj, kterým si C++ stále udržuje vyjímečné postavení mezi ostatními populárními jazyky. Touto předností jsou *šablony* (*templates*). Umožňují tvorbu typově bezpečných generických modulů. Jsou například základem implementovaného *garbage collectoru*. Dalším příkladem mohou být interpolační třídy. Použitím šablon lze jednoduše použít jeden zdrojový kód pro implementaci interpolace skalárních hodnot objemových dat nebo vektorů gradientu. Při definici šablony je použit symbolický název typu a překladač sám během kompilace programu dosadí místo symbolického typu skutečný typ, který uvedeme jako parametr šablony, a zkompiluje pro každý dosazený typ zvláštní jednotku. Při programování klasickým způsobem by bylo potřeba mít rozdílné zdrojové texty pro každý interpolovaný typ.

Jako jednoduchý příklad uveďme šablonu pro ukazatel, který se používá ve spojení s již zmíněným garbage collectorem.

```
template <class T> class Ptr
{
    T* p;
    // definice metod byly vypuštěny...
};
```

Symbolickým parametrem je `T` a při použití v programu je nahrazen typem uvedeným v lomených závorkách.

```
Ptr<int>
```

definuje ukazatel na typ `int`. Napíšeme-li šablonu pro třídu reprezentující objem dat obecného typu

```
template<class DATA>VolumeData{...};
```

potom kompilátor podlé jedné šablony vytvoří pro každý takto dosazený typ modul přeložený do strojového kódu. Efekt šablon tedy není v tom, že by zmenšily velikost přeloženého programu, ale v tom, že zmenšují zdrojový text.

`VolumeData<short>` bude typ pro ukládání šestnáctibitových znaménkových dat a `VolumeData<float>` bude objemem hodnot v plovoucí řádové čárce, například vzdálenostních dat.

Šablona s dosazeným konkrétním typem má vlastnosti jakéhokoliv jiného běžného typu a lze ji proto použít jako parametr jiné šablony.

```
Ptr<VolumeData<RGBColor<unsigned char> > >  pointer;
```

je již poněkud extrémním případem definice ukazatele na objem RGB hodnot. Může se jednat například o ukazatel na prostorovou texturu. Chceme-li použít barvy o šestnáctibitové hloubce, stačí místo typu `unsigned char` uvést jako parametr typ `unsigned short`.

7.2 Struktura knihovny a filosofie návrhu

Všechny základní třídy lze rozdělit do dvou skupin:

- třídy, které zapouzdřují data a základní metody pro přístup k nim (`DataSource`, `DataDestination`, `VolumeData`, `Image`, `Vector`, `Color`, ...)
- třídy jež s datovými třídami manipulují (`VolumeReader`, `SaveImage`, `SliceLoader`, `GammaCorrection`, `ImageTracer`, ...)

Pro názornost budou způsoby použití vysvětleny na příkladech.

7.3 Příklad konverze souboru

Příkladem budiž zpracování jednoho souboru s volumetrickými daty. Chceme například extrahovat všechny řezy ze zadaného datového souboru a uložit je jako šedotónové obrázky do zvláštních souborů.

Algoritmus je jednoduchý:

1. načti datový soubor
2. existují-li nepřečtená data přečti jeden řez dat a jdi na bod 3, jinak skonči
3. ulož řez do souboru
4. vrať se k bodu 2

O načtení souboru se stará třída `VolumeReader`

```
VolumeReader reader;
reader.read_file("filename");
```

Používané soubory mohou obsahovat maximálně dva druhy informace (vlastní data a segmentová informace) a proto jsou ukazatele na dynamicky vytvořené datové soubory vráceny dvěma funkcemi: `reader.data_result()` a `reader.segm_result()`. V případě, že některá z informací chybí, funkce vrátí hodnotu NULL.

Vrácený ukazatel je typu `DataDestination`, jenž je rozšířením typu `DataSource`.

Třída `DataSource` implementuje rozhraní pro přístup k volumetrickým datům. Přístup je omezen na čtení dat z konkrétních souřadnic a zjištění základních informací o datovém souboru, konkrétně rozměry a měřítko v jednotlivých osách. V případě úspěšného čtení může následovat tento kód:

```
DataDestination<int> *data = reader.data_result();
```

Dále je zapotřebí paměť pro jeden řez. Vyhradíme ji následujícím způsobem.

```
DataDestination<int> *data_slice =
new DataVolume<int,int>(data->Size.X, data->Size.Y, 1);
```

Třída `DataVolume` reprezentuje objemová data načtená a fyzicky přítomná v paměti. Je rozšířením třídy `DataDestination` a do dat tedy lze zapisovat.

Paměť je připravena a zbývá provést načtení řezu a uložení do souboru. Jak již bylo výše zmíněno pro manipulaci s daty jsou vytvořeny zvláštní třídy, odvozené od třídy `Transform`. Tato třída obsahuje deklarace vlastností společných pro všechny transformace. Obsahuje časovač pro měření doby výpočtu transformace a metody `transform()` a `action()`. V metodě `action()` jsou všechny aktuální příkazy příslušné transformace.

Metoda `transform()` je společná všem transformacím; obsluhuje časovač, volá metodu `action()` a zajišťuje tisk doby výpočtu na obrazovku, takže při programování vlastních transformací stačí pouze předefinovat metodu `action()` a není třeba se dále zabývat statistikami. V případě, že potřebujeme změnit její chování, nic nebrání její redefinici.

V našem příkladu použijeme dvě transformace: `SliceLoader` a `SaveSlice`. Metoda `transform()` je bez parametrů a proto je potřeba všechny údaje předat přes konstruktor (případně vlastními metodami). Třída `SliceLoader` potřebuje pouze ukazatele na zdrojový a cílový paměťový prostor (pomocí ukazatelů `DataDestination`) a třída `SaveSlice8` musí znát zdroj dat a jméno souboru, do kterého chceme data uložit.

```
Ptr<Transform> load_data_slice =
    new SliceLoader(&data, &data_slice);
Ptr<Transform> save_slice =
    new SaveSlice8(&data_slice, "data");
```

Je třeba objasnit, že třída `Ptr` implementuje ukazatel spolu s takzvaným *garbage collectorem*. Každý dynamicky vytvořený objekt odvozený od třídy `RefCount` eviduje počet odkazů na sebe samotného, který se mění vždy při přiřazení ukazatele do proměnné tapu `Ptr<>`. V okamžiku, kdy tento počet klesne na nulu, objekt sám sebe dealokuje. Řešení je jednoduché, pouze s několika málo omezeními a výrazně usnadňuje práci s dynamickou pamětí. Použití šablon je navíc typově naprosto bezpečné.

Osmička v názvu třídy `SaveSlice8` znamená ukládání v osmibitové hloubce.

Obě třídy jsou navrženy tak, aby si samy udržovaly přehled o tom, kolik řezů již načetly či uložily a při opakovaném volání čtou nebo ukládají další řez v pořadí do souboru, ke kterému je připojeno číslo (`data001`, `data002`, `data003`,...). V případě, že došlo k chybě, návratová hodnota `action()` musí být rovna nule a tuto hodnotu přečte a vrátí také metoda `transform()`. Neúspěchem může být například dosažení konce souboru nebo nedostatek místa na disku.

Je tedy jednoduché vytvořit `TransformLoop`, tedy cyklus, který iteruje obě činnosti, dokud to lze.

```
Ptr<TransformLoop> loop = new TransformLoop;

loop->add(load_data_slice);
loop->add(save_slice);
```

První transformace má podobnou funkci jako řídící proměnná cyklu. Řez je tedy uložen pouze v případě, že se jej podařilo v pořádku přečíst z datového souboru. Cyklus se spustí následujícím příkazem.

```
loop->transform();
```

Takto vypadá funkční kód převzatý ze zdrojových textů. Za pozornost stojí, že celý cyklus lze zopakovat pro segmentovou informaci načtenou z datového souboru pouze nastavením jediného ukazatele a opětovným vyvoláním cyklu.

```
int volume_reader_test(String filename)
{
    DataDestination<int> *data = 0;
    DataDestination<int> *data_slice = 0;
```

```
    VolumeReader<int> reader;

    int result = 0;

    if (!reader.read_file(filename))
    {
     cerr << "error reading file '" << filename << "'\n";
    }
    else
    {
      data = reader.data_result();
      result = 1;

      cout << "file '" << filename << "' read OK\n";

      data_slice =
        new DataVolume<int,int>(data->Size.X, data->Size.Y, 1);

      Ptr<Transform> save_slice =
        new SaveSlice8(&data_slice, "data");
      Ptr<Transform> load_data_slice =
        new SliceLoader(&data, &data_slice);

      Ptr<TransformLoop> loop = new TransformLoop;

      loop->add(load_data_slice);
      loop->add(save_slice);

      loop->transform();

      data = reader.segment_result();

      loop->transform();
    }

    delete data_slice;
    return result;
}
```

7.4 Příklad raytracingu

Předchozí příklad ilustroval základní filosofii řazení a kombinace transformací. V tomto duchu lze pokládat i samotný raytracing za transformaci, která změní obraz.

Řetěz transformací vedoucí k sérii obrazů uložené na disk může pak vypadat následovně.

```
Ptr<TransformLoop> animate = new TransformLoop;

animate->add(animator);
animate->add(tracer);
animate->add(expose_image);
animate->add(gamma_correction);
animate->add(save_image);

animate->transform();
```

`Animator` mění pozici a směr pohledu pozorovatele, `tracer` provádí vlastní vykreslení obrázku, `expose_image` a `gamma_correction` upravují získaný obrázek a `save_image` jej uloží do souboru na disk. Celý cyklus je řízen `animator` a může být přerušen chybou v kterémkoliv modulu cyklu.

Následující příklad je rovněž převzat téměř beze změny ze zdrojových textů a byl využit při generování tabulky 6.1.

```
int interpolator_test(String filename)
{
  Index size(21,21,21); // velikost generovaneho souboru
  Vector scale(2,2,2); // meritko
  float radius = 8; // polomer generovanych objektu
  float sampling_rate = 1; // vzorkovaci frekvence
  Point center(scale*(size/2)); // focus point
  Point camera_pos(60,40,100); // pozice kamery
//--------------------------------------------------------
  // generovane teleso je koule...
  Ptr<DataSource<float> >data =
  new FloatGreyBall(radius, size, scale);

  // hodnoty jsou linearne interpolovany
  Ptr<Interpolator<float> >idata =
    new DataLinInterpolator<float,float>(&data);

  // vypocet gradientu sym. diferenci
  Ptr<DataSource<Vector> >grad =   // pocitac gradientu
    new GradientSource<float, Vector>(&data);

  // interpolace gradientu
  Ptr<Interpolator<Vector> >igrad =  // interpolator gradientu
    new DataLinInterpolator<Vector, Vector>(&grad);
```

```
  // tracer pro objemova data
  Ptr<Tracer> tracer =
    new VolumeNewtonIsoTracer(&idata, &igrad, 0, sampling_rate);

  // seznam shaderu
  Ptr<Shaders> shaders = new Shaders;

  // zobrazovany objekt
  Ptr<Object> object = new SingleObject(&tracer, &shaders);
  //------------------------------------------------------------
  // seznam svetelnych zdroju
  Lights lights;
  lights.add(new Light(Point(-400, 0,400),
    RGBA(0.0, 0.0, 1.0, 1.0)));
  lights.add(new Light(Point( 200, 400,400),
   RGBA(1.0, 1.0, 1.0, 1.0)));

  shaders->add(new ChessShader(
    RGBA(1.0f, 1.0f, 1.0f, 1.0f),   // diffuse1
    RGBA(1.0f, 1.0f, 1.0f, 0.9f), 32.0f, // specular1, shininess 1
    RGBA(0.5f, 0.5f, 0.5f, 1.0f),   // diffuse2
    RGBA(1.0f, 1.0f, 1.0f, 0.9f), 32.0f, // specular2, shininess 2
    1.0f) // velikost policek
  );

  shaders->add(new PhongShader(lights));
//------------------------------------------------------------------
  // parametrem je zorny uhel
  Ptr<Camera>camera =
    new PerspectiveCamera(20*PI/180);

  Ptr<RGBAImage>image =
    new RGBAImage(300, 300);

  Ptr<Transform> save_image =
    new SaveTGA24ImageTransform(&image, "inter");
  Ptr<Transform> expose_image =
    new ExposeImageTransform(&image, 1.5f);

  Ptr<Transform> image_tracer =
    new ImageTracer(&image, &camera, &object);

  camera->move_to(center+camera_pos);
  camera->point_to(center);
```

```
  // retezec transformaci
  Ptr<TransformPipe> pipe = new TransformPipe();

  pipe->add(image_tracer);
  pipe->add(expose_image);
  pipe->add(save_image);

//---------------------------------
  pipe->transform();

  // vygenerujeme obrazy nekolika teles...

  data = new Cube(radius/2, size, scale);
  pipe->transform();

  data = new Octahedron(radius, size, scale);
  pipe->transform();

  data = new InvertedBall(radius, size, scale);
  pipe->transform();

  \\ ...zkraceno
  return 1;
}
```

Středem dalšího výkladu bude zejména modul `tracer`, jenž je typu `ImageTracer` a také je „srdcem" celého zobrazovacího systému.

7.5 ImageTracer

`ImageTracer` plně respektuje postup popsaný v kapitole o metodě sledování paprsku. Každý jednotlivý krok je implementován zvláštním modulem, aby byl snadno modifikovatelný. Samotný `ImageTracer` má na starosti výběr obrazových bodů, pro které bude vysílán paprsek, a koordinaci všech ostatních kroků.

Sám existuje v několika modifikacích:

- `ImageTracer` implementuje nejjednodušší postup obrázkem. Pro každý pixel je vyslán právě jeden paprsek.
- `ImageAATracer` navíc přidává možnost antialiasingu, který výrazně zvýší kvalitu obrazu, ale také úměrně prodlouží dobu výpočtu.

Další modifikace by mohla implementovat například adaptivní vzorkování obrazu. Výkonná část poměrně jasně ukazuje návaznost mezi jednotlivými moduly.

```
int ImageTracer::action()
{
  Index limit=Image->size(); // ukazatel na generovany obraz
  RaySegment ray;    // data pro generovany paprsek

  for (int y=0; y<limit.Y; y++)
    for (int x=0; x<limit.X; x++)
    {
      if(Camera->get_ray(limit, Vector((float)x,(float)y), ray))
      {
        if (!Object->trace(ray)) // neexistuje-li prusecik
          ray.Color.set(0,0,0,0); // barva pozadi
        Img->set(x, y, ray.Color); // vykresleni yiskane barvy
      }
    }

  return 1;
}
```

7.6 RaySegment

Kromě ukazatelů na podřízené moduly a na paměťový prostor vykreslovaného obrazu obsahuje `ImageTracer` také datovou strukturu `RaySegment`. Ta je používána jako sdílená paměť pro komunikaci mezi moduly a ukládání mezivýsledků při zpracování jednoho paprsku. Jmenujme některé z datových položek:

- `Point Origin` – počátek paprsku
- `Vector Direction` – směr paprsku
- `int Depth` – hloubka rekurze při výpočtech několikanásobných odrazů
- `RGBA Color` – výsledná barva
- `float T` – hloubka obrazu (vzdálenost nalezeného průsečíku)
- `Point Local` – souřadnice průsečíku v tělesových souřadnicích
- `Point World` – světové souřadnice průsečíku
- `Vector Normal` – normála k povrchu
- `RGBA Diffuse` – difuzní charakteristika povrchu
- `RGBA Specular` – zrcadlová charakteristika povrchu
- `float Shininess` – parametr „lesklosti“

7.7 Camera

O generování parametrů paprsku, který přísluší danému obrazovému bodu, se stará modul nazvaný `Camera`. Uchovává pozici pozorovatele a parametry použité projekce. Byly implementovány následující varianty:

- `ParallelCamera`
- `PerspectiveCamera`
- `FishEyeCamera` – implementuje sférickou projekci simulací ideálního odrazu od povrchu koule. Teoretický zorný úhel je 360 stupňů, ale v okrajových částech je obraz příliš zkreslený. Reálný úhel činí zhruba 180 stupňů.
- `FishEyeAngleCamera` – zpřístupňuje jiný model sférické projekce, kdy si lze explicitně určit zorný úhel, který je teoreticky neomezený.
- `SphericalCamera` – mapuje zorný úhel 360 stupňů tak, že ose X obrazu přiřadí horizontální úhel $\langle -180, 180 \rangle$ stupňů a ose Y vertikální úhel $\langle -90, 90 \rangle$ stupňů.

Vstupním parametrem pro určení paprsku jsou souřadnice pixelu relativně ke středu obrazu. Výstup je uložen do struktury `RaySegment`.

7.8 Object

`Object` je třída, která definuje komunikační rozhraní mezi modulem `ImageTracer` a objekty v zobrazované scéně. Toto rozhraní je implementováno ve dvou dalších modulech:

- `SingleObject` – reprezentuje scénu, které se skládá z jediného objektu
- `ObjectList` – zastupuje složitější scénu z více objektů typu `SingleObject`

Z hlediska modulu `ImageTracer` mezi nimi rozdíl není a obě se chovají shodně.
Rozhraní se skládá ze dvou základních metod:

- `advance` – zajistí průchod podél paprsku a výpočet geometrických charakteristik případného průsečíku (normály k povrchu a vzdálenosti od pozorovatele) (třída `ObjectList` v tomto případě zjistí průsečíky se všemi tělesy, vzájemně je porovná a jako výsledek vrátí ten, který je nejblíže),
- `shade` – vypočítá podle vypočítaných charakteristik barvu průsečíku (`ObjectList` volá pouze u tělesa, kterému patří nejbližší průsečík – z toho vyplývá, že `shade` lze volat až po volání `trace`).

Jelikož každé těleso má tyto metody své vlastní, mohou být plně přizpůsobeny charakteru příslušného objektu. Průsečík analyticky definované roviny se vypočítá jinak, než průsečík s isoplochou ve volumetrických datech. Jak již bylo dříve řečeno, z hlediska nadřízeného modulu není žádný rozdíl znát.

Každý definovaný objekt má zároveň možnost definice vlastního způsobu stínování, tedy výpočtu barvy v bodě průsečíku. Dává nám to možnost mnohých kombinací. Například objekty, které budou vždy v pozadí, mohou být vykresleny jednoduše, zatímco na těch důležitých si dáme záležet a budeme se zabývat zrcadlovými odrazy, prostorovými texturami, stíny a dalšími věcmi, které u ostatních zanedbáme.

Jelikož `Object` je pouze rozhraní, obsahuje ukazatele na dva další objekty, které obsahují definice obou metod:

- `Tracer`
- `Shader`

Důvod pro jejich oddělení je ten, že pro výpočet průsečíku s isoplochou ve volumetrických datech existuje několik postupů. Při jejich testování je výhodné použít stejný způsob stínování – stejný modul `Shader`, ale zvláštní moduly `Tracer` pro každou variantu hledání průsečíku. Navíc tento systém umožňuje použití shodného stínování pro objemová i geometrická data. I to je nezanedbatelná výhoda.

7.9 Tracers

Každý typ zobrazovaných těles má svůj nebo své vlastní tracery. Jejich názvy jsou většinou samovysvětlující:

- `SphereTracer` – výpočet průsečíku s koulí
- `PlaneTracer` – výpočet průsečíku s rovinou
- `BoxTracer` – urychluje výpočet průsečíku s hranicí volumetrických dat – ta je tvořena kvádrem, jehož osy jsou zarovnány se světovými osami.
- `VolumeTracer` – má poněkud zvláštní postavení neboť obsahuje kód, který je společný všem objemovým tracerům, kteří předefinují jen rozdílné části
- `VolumeFullIsoTracer` – základní raytracer
- `VolumeBinaryIsoTracer` – hledání průsečíku půlením intervalu
- `VolumeNewtonIsoTracer` – tracing pomocí metody sečen
- `VolumeNewtonDistanceIsoTracer` – medoda sečen kombinovaná s urychlením pomocí vzdálenostní transformace

7.10 Shaders

Třída `Shaders` implementuje tzv. *shading pipeline*, neboli stínovací řetězec. Jednotlivé kroky procesu stínování se mohou velmi výrazně lišit v závislosti na efektu, který chceme vytvořit. Dynamicky vytvářený řetězec jednotlivých shaderů umožňuje zahrnout jen ty kroky, které budem aktuálně potřebovat a ostatní vynechat.

Uveďme jako příklad několik způsobů stínování stejného objektu (každý objekt může mít svůj vlastní stínovací řetězec).
Depth cueing:

1. zjisti vzdálenost průsečíku od pozorovatele
2. urči jeho relativní polohu vzhledem k zobrazitelným mezím
3. namapuj tuto relativní vzdálenost do barevné palety
4. jako výsledek vrať příslušnou barvu

Jednoduchý osvětlovací model:

1. zjisti barvu materiálu v místě průsečíku
2. pro každý zdroj světla urči jeho světelný příspěvek vzhledem k barvě povrchu a orientaci příslušné normály
3. součet všech příspěvků vrať jako výsledek

Komplexnější osvětlovací model:

1. zjisti barvu materiálu v místě průsečíku
2. urči viditelnost jednotlivých zdrojů světla
3. pro každý zdroj světla urči jeho světelný příspěvek vzhledem k jeho viditelnosti, barvě povrchu a orientaci příslušné normály
4. vypočítej směr odraženého paprsku
5. sleduj tento paprsek a vyhodnoť jeho barvu
6. součet všech příspěvků světelných zdrojů a barvy odraženého paprsku ulož jako barvu původního paprsku
7. modifikuj získanou barvu průsečíku v závislosti na vzdálenosti od zdroje tak, aby modelovala pokles jasu s rostoucí vzdáleností (varianta depth-cueing)
8. uprav barvu dále tak, aby při použití brýli ChromaDepth vznikl prostorový dojem

První dva postupy nemají společný žádný krok kromě uložení výsledku. Třetí algoritmus je oba dva kombinuje a přidává výpočty odrazů a stínů (viditelnosti zdrojů světla z místa průsečíku). Pokud bychom chtěli vytvořit komplexní proceduru, která by zahrnoval všechny kroky a podle stavových proměnných by se rozhodovala, které kroky zahrnout a které ne, byl by výsledný efekt velmi neefektivní a nepřehledný. Výhodou dynamického řetězce je právě jeho modularita a přizpůsobitelnost. Umožňuje prakticky libovolnou kombinaci jednotlivých kroků, přičemž na programátorovi zůstává zodpovědnost za takovou jejich kombinaci, která dává rozumné výsledky.

Jednotlivé shadery můžeme podle jejich funkcí rozdělit do několika skupin.

7.10.1 DirectShaders

Slouží pro jednoduché přímé zobrazení povrchu tělesa. Patří sem zejména:

- `DistanceLinearShader` implementující depth-cueing z prvního příkladu

Jejich významnou vlastností je možnost použít je samostatně, nezávisle na ostatních.

7.10.2 MaterialShaders

V pořadí první se musí vyskytovat některý materiálový shader. Jeho úkolem je zjistit materiálové charakteristiky povrchu, to znamená jeho difúzní a spekulární barevné vektory. V tomto místě se provádí například mapování textur.

- `MaterialShader` – umožňuje definovat vlastní parametry povrchu
- `ChessShader` – aplikuje na objekt šachovnicovou texturu
- `WhiteShader` – bílý objekt
- `GrayTextureShader` – generuje barevný přechod (ve spojení s generátorem Perlinova šumu může vzniknout fraktální textura)

Ačkoliv již jsou definovány vlastnosti povrchu, zbývá vyhodnotit osvětlení.

7.10.3 LightShaders

`LightShaders` vyhodnocují osvětlení daného místa v prostoru a určují jeho výslednou barvu. Vlastnosti materiálu musí být nastaveny některým z materiálových shaderů.

- `PhongShader` – určí difuzní a spekulární příspěvek od každého zdroje světla
- `DiffuseShader` – vyhodnocuje pouze difúzní složku
- `ShaddowShader` – pracuje jako PhongShader, ale navíc provádí výpočet stínů
- `ColorShader` – je odlišný od předchozích, neboť nebere v úvahu žádný světelný zdroj, ale pouze výsledné barvě paprsku přiřadí difúzní charakteristiku povrchu

7.10.4 ModifyingShaders

Provádí závěrečnou modifikaci barvy určené předchozími moduly. Typickými představiteli této skupiny jsou:

- `ChromaDepthRGBShader` – zajišťující Chroma Depth barevné kódování hloubky
- `DistanceModifyShader` – modifikuje jas v závislosti na vzdálenosti od pozorovatele

Toto byla jen stručná ukázka principů použitých při implementaci přímého zobrazování v rovině. Případná další rozšíření a úpravy popsané knihovny se neobejdou bez obeznámení se se zdrojovými texty.

Závěr

Ukázali jsme, že pravidelná pravoúhlá třírozměrná mřížka voxelových dat o dostatečném rozlišení dokáže ve spojení s vhodnou reprezentací isoplochy reprezentovat povrch objektů s dostatečnou přesností na to, aby mohla být prakticky používána pro modelování a vizualizaci.

Nicméně při *nízkém rozlišení* voxelové mřížky (vzhledem k charakteru reprezentovaného objektu) vznikají při nevhodně zvolených interpolačních metodách nežádoucí defekty a deformace.

Využití vzdálenostní transformace k reprezentaci syntetických objektů zároveň pomáhá akcelerovat raytracing bez jakýchkoliv dalších paměťových nároků. V závislosti na charakteru dat je urychlení zobrazování 3 – 9 násobné.

Daní za modularitu systému je větší časová režie spojená s objektovým programováním. Je jisté, že výzkum zaměřený na rychlost výpočtu by dokázal absolutní časy ještě dále výrazně snížit.

Literatura

[1] Lorensen, W., Cline, H. *Marching Cubes: a high resolution 3d surface construction algorithm*, Computer Graphics Proceedings SIGGRAPH 87, ACM SIGGRAPH, 1989

[2] Cline, H. E., Lorensen, W. E., Ludke, S.,*Two agorithms for the three-dimensional reconstruction of tomograms*, 1996

[3] Laur, D., Hanrahan, P., *Hierarchical Splatting: A progressive refinement algorithm for volume rendering*, SIGGRAPH'91 proceedings, vol. 25, 1991

[4] Kadosh, A., Cohen, D., Shibolet O., Yagel R., *Rendering Discrete Surfaces from Close-up Views*, Tel-Aviv University, 1999

[5] Gibson, S., *Using distance maps for accurate surface representation in sampled volumes*, IEEE Visualisation proceedings, 1998

[6] Gibson, S., *Constrained Elastic Surface Nets*, proceedings Medical Image Computation and Computer Assisted Surgery, MICCAI'98, 1998

[7] Parker, S., Parker, M., Livnat, Y., Sloan, P., Hansen, C., *Interactive Ray Tracing for Volume Visualisation*, IEEE Transactions on Visualisation And Computer Graphics, vol. 5, no 3., 1999

[8] Boyer, V., Bourdin, J.J., *Fast Lines: a Span by Span Method*, EUROGRAPHICS '99 proceedings, vol.18, n.3., 1999

[9] Lacroute, P.G., Levoy, M., *Fast Volume Rendering using a Shear-Warp factorization of the Viewing Transformation*, Computer Graphics Proceedings, 1994

[10] Cohen, D., Sheffer, Z., *Proximity Clouds – An Acceleration Technique for 3D Grid Traversal*, Technical report FC 93-01, Math & Computer Science, Ben Gurion University, Beer-Sheva, 1993

[11] Grzeszczuk, R., Henn, C., Yagel, R., *Advanced Geometric Techniques for Ray Casting Volumes*, SIGGRAPH'98 Proceedings, 1998

[12] Danskin J., Hanrahan P., *Fast Algorithms For Volume Ray Tracing*, Workshop Volume Visualisation, Boston, MA, 1992

[13] Zuiderveld, K.J., *Visualisation of Multimodality Medical Volume Data Using Object-oriented Methods*, Ph.D. dissertation, Utrecht University, 1995

Seznam obrázků