VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

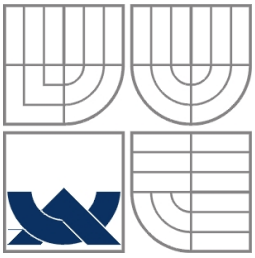
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFIKÉ INTRO 64KB S POUŽITÍM SLEDOVÁNÍ PAPRSKU

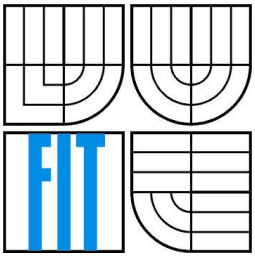
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE JIŘÍ DOČKAL
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFIKÉ INTRO 64KB S POUŽITÍM SLEDOVÁNÍ PAPRSKU

GRAPHICS INTRO 64KB USING RAY TRACING

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — JIŘÍ DOČKAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. ADAM HEROUT, Ph.D.
SUPERVISOR

BRNO 2007

Abstrakt

Tato práce se zabývá popisem tvorby aplikace určené zadáním této bakalářské práce. V dokumentu jsou popisovány problémy a principy související s danou tématikou práce. Dokument dále popisuje vlastní realizaci aplikace a dosažené výsledky práce. Závěr pak obsahuje zhodnocení a možné cesty pro pokračování práce na projektu.

Klíčová slova

intro, sledování paprsku, pixel, globální osvětlování, primární paprsek, sekundárních paprsek, odražený paprsek, lomený paprsek, Perlinova funkce, šum, průsečík, odraz světla, průhlednost, textura, algoritmus, objekt, světelný zdroj, bilineární interpolace, adaptivní vyhlazování, spekulární složka, ambientní složka, difůzní složka, material, rekurze, vektor.

Abstract

This thesis is concerned with the description of the aplication which was determined by the assignment of the bachelor thesis. The main focus are the problems and principles connected with the topic of the thesis. Further on, the thesis describes the actual realization of the aplication and the achieved results. The conclusion covers the evaluation of thesis and possible ways for further improvement of the project.

Keywords

Into, ray-tracing, pixel, global lighting, primary ray, secondary ray, reflected ray, refracted ray, Perlin´s funkcion, noise, intersection, light reflection, opacity, texture, algorithm, object, light source, billinear interpolation, adaptive sub-sampling, specular, difuse, ambient, material, recursion, vector

Citace

Dočkal Jiří: Grafické intro 64kB s použitím sledování paprsku. Brno, rok, bakalářská práce, FIT VUT v Brně.

Grafické intro 64kB s použitím sledování paprsku

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením p. Ing. Adama Herouta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Dočkal
15.5.2007

Poděkování

Rád bych na tomto místě poděkoval svému vedoucímu p. Ing. Adamu Heroutovi, Ph.D., jenž mne navedl na správnou cestu realizace bakalářského projektu.

Obsah

1 Úvod

Tvorba grafického intra je náročnou záležitostí. Už jen protože se jedná o grafickou aplikaci počítanou v reálném čase. Na autora aplikace jsou kladeny velké nároky. Musí prokázat znalost potřebných témat a v neposlední řadě značnou zručnost při implementaci návrhu aplikace. První krůčky jsou vždy těžké a před programátorem stojí velká výzva spočívající ve zvládnutí mnoha problémů souvisejících s tématem.

Při programování grafických aplikací je zapotřebí mít dostatečné znalosti a zkušenosti týkající se rozsáhlé problematiky počítačové grafiky. Spadají sem algoritmy, datové struktury, ale i pokročilé techniky programování. Proto chovám k lidem, kteří toto vše mistrně ovládají, veliký obdiv a úctu.

Následují stránky popisují mé počíná při řešení zadání bakalářské práce.

První kapitolou je úvod, který právě čtete. Měl by čtenáře seznámit se zevrubným obsahem dalšího textu.

Ve druhé kapitole popisuji své záměry a představy o konečném výsledku aplikace.

Třetí kapitolu věnuji popisu, charakteristikám a vyvozování důsledků problematik, jež mají přímý vztah k řešení mé práce. Je zde rozebrána metoda sledování paprsku, texturovací metody a techniky urychlení výpočtu pro metodu sledování paprsku.

Ve čtvrté kapitole se pak zaměřují na samotné řešení vystavivšího problému. Jsou popsány datové typy, jednotlivé funkce a rozebráno je jejich provedení.

Poslední kapitolou je závěr, kde se snažím o zhodnocení dosažených výsledků a hledání námětů pro další pokračování práce na projektu.

2 Cíl

Cíl práce na projektu nutně vychází ze zadání práce. Cílem tedy bylo vytvořit program, jež spustí a přehraje krátkou animaci s použitím dané technologie – sledování paprsku. Výsledná velikost spustitelné verze programu nesmí překročit 64kB.

Metou, které jsem chtěl při tvorbě aplikace již od začátku tvorby aplikace dosáhnout, bylo zajištění plynulého zobrazování grafického intra. Čili kladl jsem velký důraz na splnění požadavku vytvořit sledování paprsku počítané v reálném čase. Jako prioritu jsem zvolit technické zpracování na úkor uměleckých ambic, kterých může intro nabývat. Nešlo mi tedy ani tak o to co intro zobrazí, ale jak. Laťku jsem radši nenastavoval příliš vysoko, protože se jedná a můj první počin takovéhoto rozsahu v problematice počítačové grafiky

3 Teorie

3.1 Úvod

V této kapitole se zaměříme na principy a charakteristiky metod počítačové grafiky, které souvisí s řešeným zadáním mé práce. Dotkneme se však i jiných postupů a metod. Ty však poslouží spíše pro porovnání výsledků a pro lepší představu o diskutovaných problémech.

3.2 Sledování paprsku

Nejprve se zaměřím na jádro celého problému tj. na zobrazení dané scény. Sledování paprsku označuje celou skupinu metod globálního osvětlování. Přesto se v literatuře pojmem sledovaní paprsku často míní zpětné sledovaní paprsku. V následujícím textu, pokud to nebude explicitně uvedeno, budu pojmem sledování paprsku vždy označovat zpětné sledování paprsku.

3.2.1 Princip

Princip této zobrazovací metody vychází z metody vrhání paprsku. Před pozorovatelem je umístěna průmětna, která je rozdělena na určitý počet pixelů. Těmito pixely jsou pak od pozorovatele vrhány paprsky. Střetne-li se paprsek na své cestě s nějakým objektem, nastaví se barva pixeku na barvu místa střetu paprsku a objektu. Čili v místě střetu paprsku a objektu je vyhodnoceno jeho osvětlení a výsledná barva je přiřazena příslušnému pixelu. Výsledkem je dvourozměrný obraz, reprezentující trojrozměrnou scénu.

Sledování paprsku obohacuje výše zmíněnou metodu o možnost rekurze. Nejprve se vrhají paprsky skrze průmětnu a pokud dojde ke střetu s nějakým tělesem ve scéně, je spočítán i odražený paprsek. Paprsek vržený od pozorovatele skrze průmětnu se označuje jako primární a paprsek odražený nebo lomený od do tělesa jako sekundární. Barevná hodnota pixelu se pak určí jako součet jednotlivých paprsků, tedy primárního a všech příslušných sekundárních paprsků. Pro lepší pochopení uvádím ještě základní algoritmus metody sledování paprsku:

SledujPaprsek(paprsek R, hloubka rekurze H)

1. Nalezni průsečík P paprsku R s nejbližším tělesem ve scéně
2. Pokud průsečík P neexistuje, přiřaď Paprsku R barvu pozadí a skonči
3. Ke každému světelnému zdroji vyšli z bodu P stínový paprsek a pokud k němu paprsek dorazí, označ světelný zdroj jako nezakrytý
4. Vyhodnoť příspěvky osvětlení v bodě P od všech nezakrytých světelných zdrojů
5. Pokud hloubka rekurze H nepřekročila maximální hloubku sledování , vyšli

1. odražený paprsek Rr voláním SledujPaprsek(Rr, H-1)
2. lomený paprsek Rt voláním SledujPaprsek(Rt, H-1)

6. Paprsku R přiřaď výslednou barvu jako součet příspěvků osvětlení, barvy odraženého paprsku Rr a barvy lomeného paprsku Rt

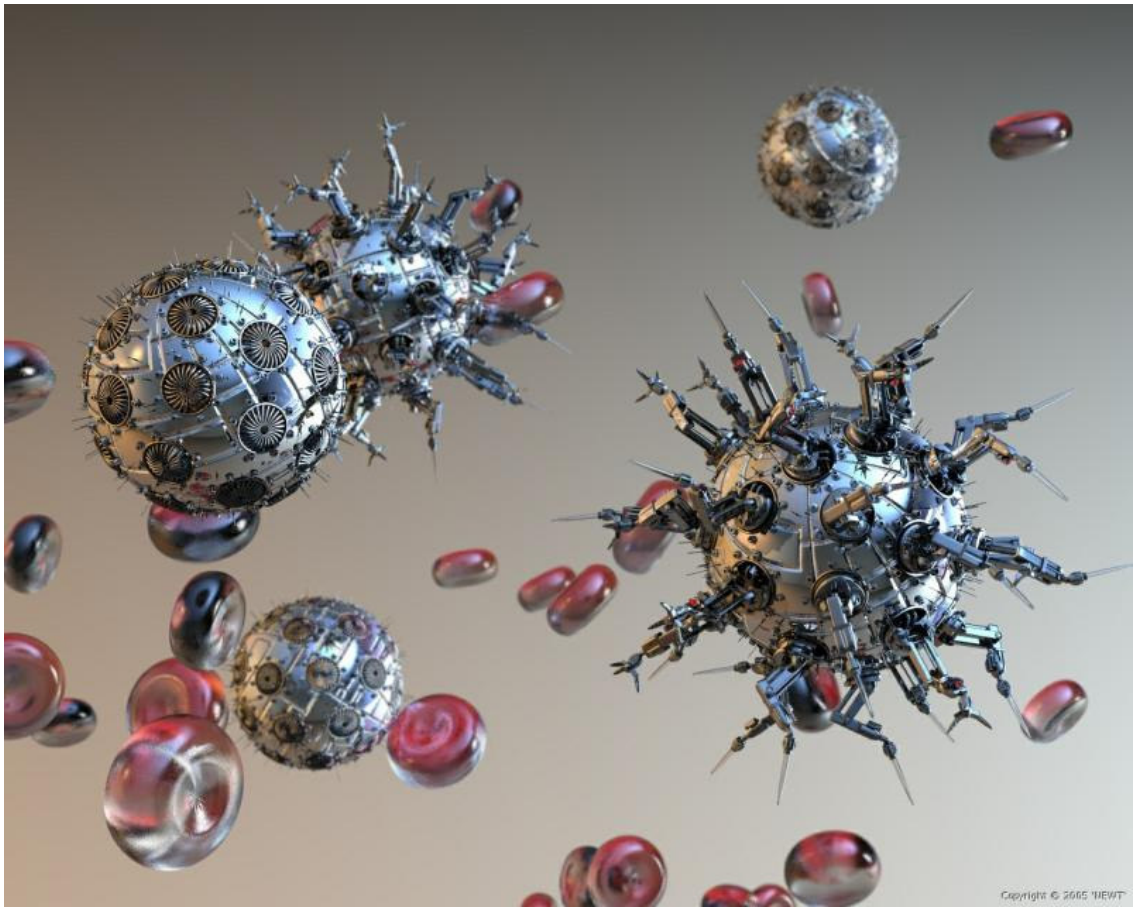
3.2.2 Charakteristika

Sledování paprsku je osvětlovací model. Vyhodnocuje tedy osvětlení v bodech, které jsou viditelné skrze danou průmětnu.

Z principu metody vyplívají její vlastnosti. Jednou z nejdůležitějších je kvalita výsledného obrazu. Ta je při použití této metody značná. Je to způsobeno samotným principem, kdy se vyhodnocují všechny pixely zvlášť a barva pixelu je pak případně ještě doplněna o barvy získané sekundárnímy paprsky. Ve výsledném obrazu jsou tedy zachyceny všechny viditelné objekty scény i s odrazy světla od ostatních těles ve scéně a to velice detailně. Metoda ve svém principu není schopna zachytit např. tzv. měkké stíny. Pokud totiž na průsečík paprsku a tělesa nedopadá světlo od světelného zdroje, je tomuto bodu přiřazena černá barva označující, že je bod ve stínu. Vedlejší bod však může být při extrému této funkce osvětlen a je mu přiřazena jiná hodnota. Vznikají tak ostré hranice stínu. To je dáno omezením základního algoritmu na bodové světelné zdroje. Dále jsou zanedbány vlnové vlastnosti světla. Není proto možné tímto postupem v obraze vykreslit ani další světelné jevy jako třeba difrakce světla na hraně nějakého tělesa. Metoda je tedy vhodná hlavně pro modelování odrazů světla a průhlednosti. Lze jí řadit k realistickým modelům osvětlení. K foto-realistickému zobrazování má však díky svým omezením daleko. Nicméně i tak poskytuje velice pro vhodnou situaci kvalitní obraz (viz obr.2.1).

Zmíněná relativní kvalita zobrazení za použití této metody je vyvážena její výpočtovou náročností. Kritickou oblastí výpočtu metodou sledování paprsku je určení průsečíku paprsku a tělesa. Základní algoritmus testuje na průsečík s paprskem všechny objekty ve scéně. Poté vyhodnocuje nejbližší průsečík. Tato operace se musí provést pro každý bod na průmětně. Počítání průsečíku zabere okolo 90% času, který je zapotřebí pro celý obraz. Pro představu nároků kladených na počítač uvádím příklad. Mějme scénu tvořenou pro demonstraci pouze jedním objektem. Vykreslení jednoho snímku bude na běžném počítači v závislosti na komplexnosti objektu při rozlišení 640 bodů na 480 bodů trvat v řádech sekund až dnů!

Další nevýhodou při použití sledování paprsku je neexistující podpora této metody na grafických procesorech. Jinými slovy, celý výpočet provádí pouze hlavní výpočetní jednotka počítače. Aby bylo možné počítat hodnoty pixelů v reálném čase je nutné použít akceleračních technik viz. dále.



Obr. 3.3.1

3.2.3 Světelný model

Pro realizaci výpočtů osvětlení daného bodu na tělese bývá velice často použit tzv. Phongův empirický osvětlovací model. Důvodem je snadná realizace a rychlost algoritmu při dobré kvalitě výsledků. Tento model navrhl v roce 1977 Bui-Thong Phong. Jedná se model pro výpočet odraženého světla. Phongův osvětlovací model rozlišuje tři druhy odrazu světla od materiálu tělesa. Odraz je rozdělen na zrcadlový (spekulární), difůzní a ambientní.

Zrcadlová složka je vyjádřena jako

$$I_s = I_L \cdot r_s (\vec{v} \cdot \vec{r})^h ,$$

kde I_L reprezentuje barevné složení dopadajícího paprsku, $\vec{v}$ je normalizovaný vektor pohledu a vektor $\vec{r}$ vyjadřuje směr ideálního zrcadlového odrazu. Koeficient zrcadlového odrazu $0 \le r_s \le 1$ určuje míru zastoupení oražené zrcadlové složky světla v celkově odraženém světle.

Koeficient h udává ostrost zrcadlového odrazu. Může nabývat hodnot v rozmezí $\langle 1, \infty \rangle$.

Difůzní složka odrazu je definována vztahem

$$I_d = I_L \cdot r_d\left(\vec{l} \cdot \vec{n}\right),$$

kde r_d je koeficient difúzního odrazu. Jedná se o trojsložkový vektor. Ten udává míru zastoupení jednotlivých složek barvy. Množství světla tím větší, čím je směr dopadu blíže k normále.To je Lambertův zákon. Je vyjádřen touto rovnicí: $I_d = I \cdot \cos\alpha$ .Difůzní složka může být chápana jako barva tělesa.

Ambientní složka určuje příspěvek okolního odraženého světla. Je to světlo vniklé mnohonásobnými odrazy světla od okolních objektů a molekulární strukturou prostředí. Je vzjádřené jednoduchým vztahem: $I_a = I_A \cdot r_a$.

Výsledná barva je pak součtem jednotlivých složek Phongova osvětlovacího modelu. Tedy takto: $I_V = I_s + I_d + I_a$

Pro použití v technice sledování paprsku se tento model rozšiřuje ještě o příspěvek barvy odraženého a lomeného paprsku.

3.2.4 Akcelerační techniky

Jak jsem zmínil již výše je metoda sledování paprsku velice náročná na výpočetní výkon stroje. Jedním z hlavních požadavků na můj projekt je však vytvoření animace, která bude používat osvětlovací model sledování paprsku. Výsledná animace by měla vykreslovat alespoň 15 snímků za sekundu, aby uživateli na běžném počítači připadala alespoň částečně plynulá. Toho lze dosáhnout za pomoci akceleračních technik. Těmto technikám se věnují mnohé výzkumné týmy po celém světě a problém urychlení výpočtu klasického algoritmu sledování paprsku je po dlouhou dobu předmětem zájmu mnoha špičkových odborníků. Důvod je prostý. Již současné existující techniky totiž mohou urychlit výpočet o 2 až 3 řády!

Akcelerační techniky můžeme rozdělit na 2 hlavní skupiny: techniky, které urychlují výpočet průsečíků s objekty scény a techniky, které snižují počet vypočítávaných paprsků viz Tab. 2.1.

.

	Klasifikace akceleračních metod pro sledování paprsku		
Zástupci	**Urychlení výpočtu průsečíků**		**Menší počet počítaných paprsků**
	Zvláštní výpočet pro každý typ objektu	Dělení prostoru	Adaptivní vyhlazování
	Jednoduché obálky	Hierarchie obálek	Adaptivní řízení rekurze

Tab 2.1

Skupinu technik urychlujících výpočet průsečíků s tělesy, můžeme rozdělit ještě na dvě menší podskupiny

Tou první je urychlování výpočtu pro každý typ objektu zvlášť. Existuje celá řada velice rychlých podprogramů pro každý typ objektu. Jejich důležitou součástí je určení zda je testovaný objekt zasažen paprskem či nikoli. V tomto přístupu je ukryta jejich výsledná rychlost. Např. pro kouli je možné testovat, zda jí paprsek mine nebo trefí ještě před řešením kvadratické rovnice pomocí několika násobení a jednoho porovnání.

Druhou podskupinou tvoří techniky, které umožňují snížit počet testovaných objektů vůči paprsku. Techniky jsou založeny na principu uchování dalších informací o vykreslované scéně. Technika dělení ohraničí všechny objekty do co nejmenšího kvádru. Ten je následně rozdělen na menší části a těm je přiřazeno patřičné těleso, jež se v části nachází. Při hledání průsečíku s nejbližším tělesem scény, pak algoritmus vyhodnotí aktuální pozici paprsku, tj. ve které části scény se paprsek nachází, a otestuje pouze objekty, které se v této části nacházejí. Pokud se ve scéně nalézají objekty, které pro nalezení průsečíku vyžadují mnoho času (nejlepším příkladem je polygonální model) je časová úspora obrovská. Uvádí se více než desetinásobné urychlení výpočtu scény. Pokud však scéna obsahuje jednoduché objekty, kde je výpočet průsečíku relativně rychlý (např. koule), projeví se urychlení jen při velkém počtu těles. Lze jí tedy použít v rozsáhlých a komplikovaných scénách. Výsledný obraz je pak stejně kvalitní a je rychleji vykreslen. Další možností, která snižuje počet testovaných objektů vůči paprsku je použití hierarchie obálek. Obálka je ve své podstatě další těleso , které se přidává do scény v průběhu jejího vykreslování. Obálka je objekt jehož tetování je velice rychlé. Touto obálkou, nejčastěji tvaru koule, je „obaleno“ každé těleso scény. Jednotlivé obálky jsou pak uspořádány do hierarchie podle umístění v prostoru. Algoritmus po té pro primární paprsky prohledává celou strukturu obálek a testuje zda by mohlo dojit k průsečíku. Pokud ano, je ještě nutné otestovat objekt, který je uvnitř obálky. Rychlost této metody se projeví výrazně až u sekundárních paprsků. Pro ně totiž testování objektů začíná od listu stromu, kde byl nalezen průsečík. Tento princip lze opět s výhodou použít pro komplikované scény. Výsledné urychlení se pro takovéto prostory opět uvádí více než desetinásobné.

Z technik, které snižují počet vypočítávaných paprsku, uvádím nejprve metodu adaptivního řízení počtu rekurzí. Princip spočívá ve vyhodnocení příspěvku sekundárního paprsku k primárnímu. Tedy pokud je barva přinášená sekundárním paprskem nižší než je stanovená hranice, pak se provádění rekurzí zastaví i přesto, že se paprsek setkal s nějakým objektem v prostoru. Metoda snižuje adaptivně počet sekundárních paprsků a přináší jen velmi malé, většinou okem nerozeznatelné, zkreslení. Urychlení však není příliš značné pokud scéna neobsahuje mnoho navzájem odrážejících se těles. Další metodou je adaptivní vyhlazování. Princip spočívá v rozdělení průmětny na čtverce podle zvoleného měřítka. Pro každý čtverec je pak určena výsledná barva pomocí klasického algoritmu sledování paprsku ve všech jeho rozích. Algoritmus také ukládá krom výsledné barvy i další potřebná data. To mohou být informace o zasažených objektech, příspěvky

sekundárních paprsků apod. Kritickým místem je pak vyhodnocení všech rohů aktuálního čtverce. Pokud jsou všechny rohy shodné, jsou zbylé pixely interpolovány zvolenou metodou. Pokud se však alespoň jeden roh liší od ostatních, je původní čtverec rekurzivně dělen na další čtyři čtverce. Ty jsou opět vyhodnocovány. Jsou-li rohy stále rozdílné a zároveň již nelze čtverec rozdělit, jsou zbylé body dopočítány sledováním paprsku.

Jako vhodná metoda interpolační technika se zde nabízí bilineární filtrace. Ta je obecně určena pro interpolaci hodnot konvexního mnohoúhelníka. Pro nás je útvar ještě jednodušší, měla by tak být zajíštěna dostatěná výkonnostní výhoda.

Technika adaptivního vyhlazování velice efektivně snižuje počet pixelů, které je nutno vykreslit použitím sledování paprsku při zanechání detailů kvůli nimž sledování paprsků používáme. Nevýhodou je samozřejmě ztráta některých informací. Hlavně na objektech samotných, pokud nedochází k odrazům paprsku na další objekty apod.

Pro urychlení výpočtu obrazu je také možné zvolit jiné cesty. Ty jsou však založené na technickém vybavení jež máme k dispozici. Např. je možné zpracovávat scénu paralelně na více procesorech či obdobně použít specializovaný procesor pro urychlení nejnáročnějších výpočtů jako průsečík paprsku s polygonem.

Pro svou práci jsem zvolil techniky adaptivního vyhlazování a adaptivního řízení rekurze. Důvod je nasnadě. Techniky snižování počtu testovaných objektů na průsečík s paprskem totiž pro jednoduchou scénu nepřinášejí požadovaný efekt urychlení na takovou míru, aby bylo možné vykreslovat více snímků za jednu sekundu. Snižováním počtu paprsků je možné toto dosáhnout pro jednoduchou scénu s jednoduchými geometrickými tvary, i když za jisté snížení výsledné kvality snímků. V praxi se pak často jednotlivé techniky kombinují pro dosažaní co největšího výkonu dle zadaných kritérií.

3.3 Textury

Textura definuje povrchové a barevné vlastnosti objektu. Tedy z jakého strukturu a povrchové vlastnosti objektu. Textury lze při nejjednodušším pohledu na problematiku buď načítat ze souborů jako barevné mapy čí je generovat za běhu programu. Ve druhém případě mluvíme o procedurálních texturách. V zadání mé práce je však požadavek omezující velikost výsledného spustitelného souboru na 64kB. Je tak jasné, že nemohu použít mapy načítané ze souboru. Proto se budu v následujícím textu věnovat texturám procedurálním. Pokud v následujícím textu uvedu termín textura, bude tím myšlena textura procedurální.

Textura je ve své podstatě krátký algoritmus , který pro zadané souřadnice bodu na povrchu objektu vytváří hodnotu, jež je pak použita k reprezentaci barvy daného bodu. Při tvorbě takovýchto algoritmů se s výhodou používají šumové funkce. V následujícím textu se přidržím popisu z [1].

Optimální šumová funkce by měla poskytovat bílý šum. Takový šum lze libovolně zvětšovat, protože obsahuje detaily pro všechny úrovně. Jeho použití však není nutné. Takové množství detailů není prostě zapotřebí (např. pro velkou vzdálenost) a navíc je tento šum výpočetně velice náročný. Šum, který by bylo možné použít pro generovaní textur musí mít následující parametry:

- Musí být staticky invariantní vzhledem k otáčení,
- staticky invariantní vzhledem k posunutí,
- spojitý,
- s omezeným frekvenčním spektrem a
- opakovatelný

Takový algoritmus navrhl p. Perlin a je základem pro tvorbu textur. Později se tato funkce začala označovat jako Perlinova funkce. Perlinova funkce splňuje všechny výše uvedené požadavky bna šumovou funkci. Nezáleží tedy, kde začneme texturu mapovat a jak bude natočená. Funkce je spojitá, poskytuje tedy hodnoty pro jakékoliv vstupní argumenty. Má omezené frekvenční spektrum, je tedy snadno a rychle vypočitatelná. A konečně pro stejné argumenty generuje vždy stejnou hodnotu šumu. Pro zadané dvojnásobné vstupní argumenty, generuje tato funkce šum s dvojnásobnou frekvencí. Je také možné generovat hodnoty pro libovolnou dimenzi. Toho lze s výhodou využít pro metodu sledovaní paprsku, kdy nemusíme pro průsečík paprsku na tělese počítat ještě výsledek mapovací funkce pro dané těleso, ale může bod průsečíku rovnou předat trojrozměrné šumové funkci. Ta vrátí hodnoty textury pro daný bod v prostoru. Základní myšlenkou Perlinovi funkce je rozdělení prostoru do pravidelné mřížky. Vrcholy mřížky označíme [i, j, k]. V každém z těchto vrcholů je definována trojrozměrná pseudonáhodná funkce označovaná jako „vlnka“. Postup výpočtu Perlinovy funkce pro reálný bod [x, y, z] je pak následující:

1. Urči, ve které buňce mřížky se bod [x, y, z] nachází.
2. Pro každý z osmi sousedních vrcholů vypočítej tvar vlnky
3. Sečti hodnoty vlnek odpovídající poloze [x, y, z]

Perlinova šumová funkce je základem algoritmů pro skládání šumových funkcí. Myšlenka je následující.

Složení šumových funkcí je součet funkcí generujících šum, každá s se změněnou amplitudou a frekvencí. Obecně je takto možné skládat libovolné šumové funkce. Většinou se však jedná o Perlinovy funkce.

3.4 Fenomén grafického intra s omezenou velikostí

Grafické intro je jedním z nejzajímavějších fenoménů digitálního umění. Vznik kultury okolo grafických inter se datuje do období osmdesátých let. Tehdy začali vznikat první prográmky pro prolomení ochrany na kopírování her. Byl to zároveň počátek počítavého pirátství. Tehdy první crackeři umísťovali před hru samotnou svoje vlastní grafické prezentace, pokud se jim podařilo ochranu ze hry odstranit. Tato prezentace byla důkazem, že právě dotyčný cracker prolomil ochranu hry proti kopírování jako první. Grafické intro mělo v té době občas vyšší technickou úroveň než hra samotná.

Grafické intro jako takové, by mělo předvádět technickou zručnost tvůrců. Zároveň je kladen velký důraz na uměleckou stránky provedení.

Dnes se k nám intra většinou nedostávají touto cestou, ale jsou volně stažitelné z internetu, jako samostatná díla, prezentují se na festivalech nových medií, a často je uvidíte jako součást VJ vystoupení apod. Díky velikosti inter, je jejich přenositelnost a šíření dál velice jednoduché. Velikosti spustitelných souborů se pohybují od 4kB po řádově desítky až stovky kB. Přesto, že by se mohlo zdát, že výsledná animace nemůže nijak kvalitně vypadat, je právě opak pravdou. I do animace, jejíž velikost je 4kB, může být velice poutavá, zajímavá a sdělovat i silnou myšlenku.

4 Implementace

4.1 Úvod

Vlastní implementace jakéhokoliv zadaného problému je vždy kritickým místem při tvorbě aplikace. Tedy převést požadavky zadání do programové podoby, lze vždy prakticky neomezeným počtem způsobů. Při návrhu samotné aplikace je nejdůležitější samotný návrh programu. Každý programátor

začínal od chyb. Po strastiplných začátcích, kdy si programátor uvědomí, že měl program psát jinak a přitom je již v půlce řešení, přijde časem kolik práce ušetří kvalitní a detailní návrh. I já jsem při tvorbě programu realizujícího zadání mé bakalářské práce takto postupoval

4.2 Návrh aplikace

Při návrhu jakékoli aplikace je nutné nejprve analyzovat požadavky kladené na výsledný program. V prvé řadě je tedy nutné ujasnit si alespoň celkovou funkčnost programu. Často se uplatňuje metoda návrhu zhora dolů. Při samotném návrhu postupujeme od nejhrubších základních částí jako je např. již zmiňovaná celková představa o funkčnosti celého systému. Je velice nutné postupovat důsledně a strukturovaně. Návrh, který v sobě obsahuje logické či jiné chyby, by nás pak při implementaci problému příliš nepotěšil. Musíme mít stále na paměti čeho chceme vlastně dosáhnout při realizaci práce. Tedy např. pro jaký konečný účel bude naše aplikace používána (jistě je rozdíl v návrhu databázového systému nebo třeba grafického intra), kdo bude naši aplikaci používat (vhodné ovládací prostředí), a na jakém technickém vybavení bude aplikace spouště.

Toto je samozřejmě jen pouhé zjednodušení celé problematiky návrhu aplikací. Nutno také podotknout, že kvalitní návrh je základem kvalitní aplikace. Vytvořit takový návrh však pro sloužitější a rozsáhlejší aplikace je často velice obtížné. Proto vzniklo i odvětví informatiky, jež se touto problematikou zabývá. Řeč je o softwarové inženýrství. Toliko úvodem. Nyní již samotný návrh mé aplikace.

4.2.1 Požadavky

Ze zadání mé práce vyplívají základní požadavky na výsledný program. Připomínám zadání mé práce: grafické intro 64kB se sledováním paprsku. Může se zdát, že se po řešiteli příliš mnoho nepožaduje a že jde vlastně jen o malý prográmek. Toto by však mohlo napadnout pouze člověka, který se o programování a technické záležitosti s tím spojené příliš nezajímá.

Základními požadavky kladené na funkčnost programu jsou následující:

1. program má přehrávat statickou či neměnnou animaci tzv. intro
2. program musí používat technologii sledování paprsku
3. velikost výsledného spustitelného souboru nesmí přesáhnout 64kB

4.2.2 Specifikace

Program má přehrávat grafickou animaci - intro. Intro je neměnné a neumožňuje iterakci s uživatelem. Není tedy nutné implementovat různé grafické ovládací prvky či reagovat na množství

vstupů přicházejících od uživatele. Je však nutné zajistit alespoň reakce na klasické vstupy. Tím je v tomto případě snad pouze možnost předčasného ukončení zobrazování intra.

V této souvislosti můžeme specifikovat koncového uživatele spustitelného souboru. Grafické intro je velkým fenoménem v oblasti počítačové grafiky. Lze tedy předpokládat veliký zájem o tyto prezentace. Díky své limitované velikosti, jej lze snadno umístit na veřejnou počítačovou síť a nabídnou ke shlédnutí vlastně komukoli. Aplikace je tedy určena pro široké masy obdivovatelů tohoto umění, zvědavce, jakožto i pro tvůrce demo scén .

Jelikož se jedná o grafický program, je nutné také zhodnotit nároky na technické vybavení uživatele. Musíme však se však vždy podřídit právě koncovému uživateli. Toho dozajista nebude příliš lákat program, u kterého musí na vše čekat a nic jiného už na jeho stanici nepůjde ve stejnou dobu provádět. Pro grafické intro platí, že bude s největší pravděpodobností spouštěno na osobní počítačích. To nám poskytuje jistou představu o nárocích jež můžeme klást na výkon sytému. Grafické intro se většinou spouští v celoobrazovém módu.Za ideálních podmínek, má tedy k dispozici celý výkon systému pro provádění svých operací. Při ladění náročnosti programu na systém jsem se držel této myšlenky.

Velice důležitým bodem ze specifikace požadavků je bod č. dvě. A totiž povinné použití osvětlovací techniky sledování paprsku. Jak jsem již uvedl v předcházející kapitole, přináší tato metoda kvalitní výsledky, ale bohužel za vysokou cenu náročnosti výpočtu scény. Intro má však být renderováno v reálném čase. Lidské oko dokáže zachytit 25 snímků za jednu sekundu. I to je při použití osvětlovacího modelu sledováním paprsku značné ne-li obrovské množství výpočtů pro osobní počítače. Algoritmus bude tedy nutno optimalizovat pomocí akceleračních technik. Celý výpočet se musí odehrávat pouze na centrální procesorové jednotce. To dále zvyšuje nároky na program jako takový. Proto je důležité zvolit ty správné akcelerační techniky k urychlení celého výpočtu. Asi nejpoužívanější je pro toto oblast tvory programů, adaptivní vyhlazování. Jeho charakteristika je uvedena v předcházející kapitole. I já pro svou raci zvolil tuto metodu. Jelikož se jedná o časové opravdu náročnou metodu osvětlení scény, jen akcelerační techniky pro provoz na běžném domácím počítači nestačí. Vyvstává nutnost přizpůsobit i samotnou vykreslovanou scénu. Určit např. maximální množství objektů, jež reflektují jiná tělesa scény. Dále pak můžeme stanovit i počet zároveň zobrazených těles v prostoru. Každé takové rozhodnutí je velmi podstatné pro konečnou rychlost výpočtu. Nastavení prostoru scény je i přes svoji značnou důležitost záležitostí spíše zkušeností s tvorbou a testování aplikace. Dalším aspektem ovlivňujícím efektivitu implementace, je použitý programovací jazyk. V tuto chvíli přicházejí do boje neoblíbené nízkoúrovňové jazyky. Nejznámějším, ale také nejpoužívanějším jazykem, je pro tvorbu intra jistě asembler. Umožňuje vytvořit velice rychlý a efektivní kód. Možností je i použití programovacího jazyka vyšší úrovně. V hodným kandidátem je zde jazyk C. Za větší pohodlí při programování v tomto jazyce se poněkud ztrácí rychlost výsledného kódu. Jazyk C je vhodný zejména pro svou práci s pamětí a vcelku dobré možnosti ladění kódu. Pro svůj program jsem se rozhodl použít jazyk C.

A konečně velikost spustitelné verze programu nesmí přesáhnout požadovaných 64kB. Tímto je jasně určen přístup pro texturování objektů. V úvahu přicházejí pouze procedurální textury (viz kapitola 2.2). Velikostí je také do jisté míry limitována délka trvaní animace. Pro dynamickou a často se měnící scénu, je totiž zapotřebí důmyslnějších a obsáhlejších algoritmů. Vzhled intra zůstává pouze na autorovi a je limitován pouze jeho představivostí.

Lze tedy říci, že naším úkolem je vytvořit co nejzajímavější grafické intro, které bude pro své zobrazení používat postup sledování paprsku. Je možné ho přehrávat na běžném osobním počítači a jeho velikost by neměla překročit 64kB.

4.3 Vlastní implementace

4.3.1 Datové typy

Při vlastní tvorbě aplikace jsou postupoval systémem zdola nahoru. Nejprve bylo tedy důležité ujasnit si hlavní datové typy, jichž budu v aplikaci využívat. Stanovil jsem si proto nejprve popis dat, se kterými bude aplikace pracovat a jež bude ke svému chodu potřebovat. Základem celého programu je algoritmus sledování paprsku. Ten z principu pro svou činnost musí znát veškerá nastavení scény. Algoritmus vyžaduje tedy informace o právě pořítaném paprsku čili jeho směr a velikost, informace o umístění všech objektů ve scéně, u objektů pak ještě další informace o velikosti, parametry pro odraz a lom světla a informace o materiálu. Dále musí mít přehled o všech světelných zdrojích jako pozice a barva světla.

Pro použití metody adaptivního vyhlazování je pak nutno uchovávat další informace, jež budou při vyhodnocování sloužit jako kritéria pro určení zda se má zadané pole mřížky dále dělit či je možné rovnou zbylé barvy interpolovat mezi jednotlivými rohy pole.

Datové typy byla tedy zvoleny tak, aby vyhovovaly těmto požadavkům. Nyní popíši jen ty nejzákladnější z nichž jsou dále odvozeny další. Těm se budu podrobněji věnovat při popisu jednotlivých funkčních bloků programu. Datové typy jsou k nahlédnutí v příloze č. 2.

Stavebním kamenem typů jsou struktury pro uchování informace o poloze bodu, směru vektoru a hodnot barev. Jedná se o struktury složené ze tří hodnot základního datového typu float. Tento typ je použit jako základ z důvodu urychlení výpočtů oproti provádění výpočtů pomocí datového typu double. Tato možnost spočívá v požívání specializované jednotky procesoru FPU. V programu jsou tyto struktury pojmenovány Point, sloužící pro uchování pozice nějakého bodu, Vector uchovává směr nějakého vektoru a Color pro uchování barevné hodnoty RGB.

4.3.2 Vektorové operace

Metoda sledování paprsku nejčastěji realizuje právě operace s vektory. Každý program implementující metodu sledování paprsku musí tyto operace umožňovat. Program realizuje klasickou sbírku používaných operací nad vektory. Jsou to tedy: součet a rozdíl dvou vektorů, skalární a vektorový součin, operace normalizování vektoru, zjištění délky a otočení směru vektoru. Každá z těchto operací je realizována funkcí vracející příslušné požadované hodnoty. Pro přesnou implementaci odkazuji čtenáře opět do přílohy č. 2.

4.3.3 Algoritmus sledování paprsku

Základem je funkce hledající nejbližší průsečík paprsku ve scéně. Paprsek samotný je reprezentován datovým typem Ray. Ten v sobě obsahuje informaci o místě počátku p polopřímky ve formě typu Point a směrový vektor d. Hodnoty paprsku generuje funkce ComputeRay, jež na základě informací o nastavení kamery a určeným souřadnicím průmětny, vrazí hodnoty bodu počátku paprsku a směrového vektoru. Směrový vektor má normalizovanou hodnotu. Bod počátku je pro všechny paprsky vrhané v průběhu pořizování jednoho snímku stejný. Je přesně roven umístění kamery. Ta uchovává informaci o místě pozorovatele a směru pohledu pozorovatele. Funkce CamLookAt nastaví, podle zadané pozice a bodu pozorování, své vnitrní vektory. Datový typ Camera poskytuje informace o směru jaký je obloha, o směru umístění průmětny a na oba tyto směry kolmý vektor určující směr do prava. Tyto informace používá právě funkce ComputeRay k získání směru paprsku. Informace o paprsku a objektech scény je předána již ymíněné funkci pro nalezení nejbližšího průsečíku paprsku a tělesa. Funkce je pojmenována Intersect. Ta pro každý objekt scény provede test průsečíku paprsku a daného objektu. Pokud je nalezeno více průsečíku s tělesy, vybere se pouze ten nejblíže kameře. Ten je vrácen ve přes parametr předávaný formou ukazatele na typ Isect. Funkce návratovou hodnotou v jedničce oznamuje, že došlo k průsečíku s nějakým objektem. Pokud funkce skončí s návratovou hodnotou rovnu nule, pak k průsečíku s nedošlo s žádným tělesem. Předávaný typ Isect v sobě nese údaje o násobku směrového paprsku pro výpočet bodu průsečíku a odkaz na objekt jehož se průsečík týká.

Hlavní funkcí implementující metodu sledování paprsku je funkce pojmenovaná jako TraceRay. Jejím úkolem je samozřejmě výpočet barvy pro daný pixel průmětny. Vstupem pro tuto funkci jsou informace o všech objektech, světlech a maximálního počtu rekurzí výpočtu. Objekty jsou do funkce předány ukazatelem na pole ukazatelů na objekty. Stejné je to i pro zdroje světla. Funkce nejprve za pomoci funkce Intersect vyhodnotí možný průsečík. Pokud je pro paprsek nalezen, pak přistupuje k vyhodnocení osvětlení bodu. Z místa bodu průsečíku je vyslán nový paprsek ke každému zdroji světla. Daný bod je ve stínu pokud je skalární součin primárního paprsku vysílaného do scény a paprsku pro světelný zdroj větší než nula. Je-li záporný, nemůže světlo dopadat na zkoumaný bod a vyhodnocování světla se ihned přeruší a je testován další světelný zdroj. Pokud však je možné, aby

byl bod osvícen, je ještě nutné zjistit, zda není světelný paprsek zastíněn jiným tělesem. Pro této účel je opět volána funkce Intersect, ale již pro nový paprsek. Pokud je nakonec zkoumaný bod přeci jen osvícen jsou generovány hodnoty povrchu tělesa (viz níže) a znásobeny podle Phongova osvětlovacího modelu (viz kapitola 2.2.3). Je-li Aktuální hloubka rekurze větší než nula je provedeno další rekurzivní volání této funkce. Možnost rekurze je také podmíněna hodnotou koeficientu odrazivosti tělesa. Implementovaná verze algoritmu sledování paprsku přináší ještě další informace využité jako kritéria rozhodování o dělení v adaptivním vyhlazování. Blíže samostatná podkapitola zabývající se implementací této metody.

Pro objekty je definován vlastní datový typ Object. Ten udržuje odkaz na strukturu tělesa např. kouli nebo plochu. Typ Object dále obsahuje ukazatel na množinu funkcí, jež lze s tělesem provadět. Tato množina je nazvaná ObjectProcs. Udržuje odkazy na jednotlivé funkce příslušející typu objektu. Funkcemi jsou výpočet průsečíku a vyjádření normály. Díky tomuto přístupu je možné pro jakýkoli typ objektu (koule apod.) volat funkce příslušné objektu vždy shodným zápisem aniž bychom předem věděli o jaký druh objektu se jedná (viz [2]). Rovněž typ Object uchovává informace o materiálu tělesa v podobě odkazu na typ surf. Tento typ obsahuje koeficienty odráživosti světla od tělesa, koeficient velikosti odlesku na tělese a ukazatel na funkci pro generování šumu použitého pro výpočet koeficientu difůzní složky materialu.

4.3.4 Textury

Textury jsou implementovány použitím Perlinovi funkce pro generování šumu. Jde o základní verzi, která generuje čísla mezi nulou a jedničkou. V programu jsou rovněž umístěny statické barevné tabulky. Tato paleta může být použita pro rychlé barevné generování textury.

4.3.5 Adaptivní vyhlazování

Základní myšlenka metody byla popsána v kapitole 2.2.5.Nyní k samotné implementaci.

Základním typem je Square jež zastupuje políčko mřížky. Tento typ obsahuje pole čtyř ukazatelů na typ SqPointColor. Ta schraňuje pozici rohu políčka relativně k obrazovce a výslednou barvu pixelu. Dále obsahuje pole ukazatelů na objekty jež naplní funkce TraceRay. Pole uschovává reference postupně na všechny objekty, které byly předmětem průsečíku jednotlivých rekurzí funkce TraceRay. Kněm jsou jsou rovněž uchovány barvy vniklé vzhodnocením osvětlení na průsečíku s tělesem. Tyto informace pak poslouží pro rozhodnutí o nutnosti Prvotní ustavení célé struktury mřížky přes obrayovku provádí funkce preRender. Jejím vstupem je staticky alokované pole položek typu Square. Statické pole bylo použito, protože zůstává neměnné po celou dobu renderování animace. Funkce proto pouze dynamicky nealokuje paměť pro jednotlivé rohy čtverců. Čtverce se částečně překrývají. Tím je dosaženo minimálního počtu bodů, jež je nutné vykreslovat pomocí

sledování paprsku. Ostatním rohům se přiřadí adresa odpovídajícího shodného rohu. Počáteční rozdělení scény je po osmi pixelech.

Funkce Render je hlavní funkcí programu, ze které jsou volány další často důležitější funkce. Tato funkce nejprve naplní všechny rohy každého čtverce hodnotami výsledné barvy a již zmíněnými informacemi o dílčím vyhodnocení sekundárnách paprsků. Pokud nedojde při volání funkce Trace Ray je pole ukazatelů nastaveno na hodnotu NULL. Tedy takové pole je vždy ukončeno touto hodnotou.Po té co jsou naplněny všechny rohy každého čtverce, je spuštěn rozhodovací algoritmus. Ten prochází a zpracovává všechny čtverce scény. Funkce je pojmenována CalculateTexel. Nejdůležitější částí je rozhodnutí o případném jemnějším rozdělení prováděného čtverce na čtyři další. Rozhodování provádí funkce Compare. Ta kontroluje shodnost všech rohů čtverce. Nejprve je porovnána reference na objekt, který byl zasažen primárním paprskem. V případě, že se alespoň jedna reference liší od zbylých, je funkce ukončena s návratovou hodnotou jedna. Ta indikuje nutnost dalšího dělení. Tímto způsobem je možné detekovat hrany jednotlivých objektů a ty pak jemněji vzorkovat. Jsou-li reference na objekt primárního paprsku stejné čili celý čtverec se nalézá uvnitř jednoho objektu, příjde ke slovu kontrola podle stavu jednotlivých světel. Tyto stavy opět generuje Funkce TraceRay a ukládá je do pole prvků typu int. Světlo může nabývat těchto stavů:

- Světelný zdroj zkoumaný bod neosvětluje – v tomto případě je hodnoto stavu zdroje nastavena na nulu.
- Zkoumaný bod je zdrojem světla osvícen – tomuto stavu odpovídá hodnota jedna
- Bod je osvícen a zároveň dochází ke speculárnímu odrazu světla – pak má ydroj světla přiřazenu hodnotu dva

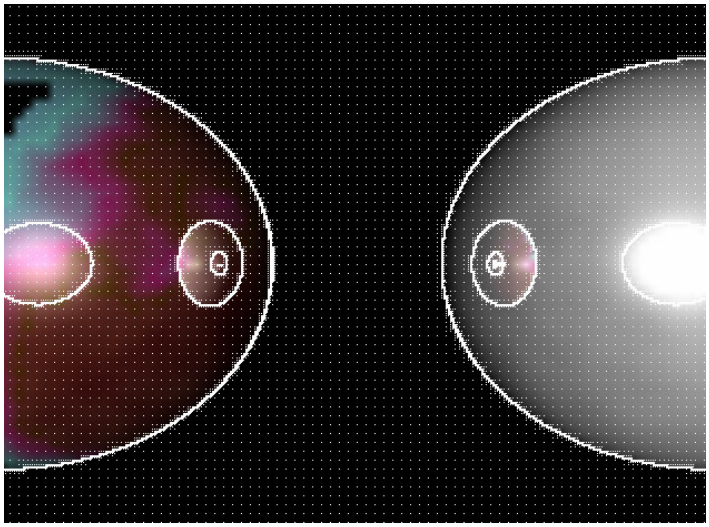
Opšt jsou porovnána všechna světla v každém rohu. Dělit není nutné pokud jsou hodnoty stavů všech světel stejné pro všechny rohy zpracovávaného čtverce. Tímto způsobem můžeme ve scéně odhalit hrany stínů a výraznější spekulární odrazy světla.

Skončí-li i tento test úspěchem tj. nedošlo k odhalení nějakého rozdílu, přihází na pomyslné kolbiště informace o zasažených objektech sekundárními paprsky. Znovu poprcháváme celá pole referencí na objekty pro všechny rohy čtverce. Jakmile je nalezena neshoda, třeba že až na položce odpovídající třetí rekurzi volání funkce sledování paprsku, je test neúspěšný a daný čtverec se bude dále dělit. Tímto odhalíme obrysy reflexí a to i několikanásobných.

Při úspěchu tohoto testu máme v rukávu schované ještě jedno eso. Tím jsou jednotlivé barevné příspěvky pro provedené rekurze paprsku. Princip porovnávání je shodný s předešlými testy. Tímto testem určíme i odražené odlesky ostatních těles.

Případné dělení čtverce při neúspěchu některého z testů vyvstane požadavek na rodělení zpracovávaného čtverce na čtyři menší. Dělení lze provádět dokud máme jak dělit. Implementovaný algoritmus poskytuje možnost dělení až do úrovně 1x1, kdy jsou všechny body čtverce dopočítány klasicky za použití sledování paprsku. Vzniklé čtyři nové čtverce pak vyhodnotíme rekurzivním voláním funkce ComputeTexel.

Při lepe testu je oblast mezi rohy interpolována funkcí Interpol. Ta pro proložení bodů používá bilineární interpolaci popsanou textu výše. Závěrem uvádím obrázek demonstrující dosažený výsledek. Bílé body označují místa, kde bude použito metody sledování paprsku k získání barevné informace.



Obr. 4.1

4.3.6 Vlastní zobrazení snímku

Programem je alokováno pole pixmap, do kterého jsou postupně ukládány výsledné barvy. Obsah tohoto pole je pak pomocí příkazů knihovny OpenGl zapsán do framu buferu grafické karty a je obsah předního a zadního buferu je pak prohozen. Využívá se tedy postupu doublebuffering.

5 Závěr

Dosažený výsledek je vcelku přijatelný. Podařilo se imlementovat akcelerační techniku adaptivního vyhlazování, jež urychlila základní algoritmus sledování paprsku až 7krát. Podařilo se tak dosáhnout plynulé animační scény. Dosažení plynulosti je však podmíněno technickým vybavením, které máme k dispozici. Záleží hlavně na použitém procesoru. Doporučenou konfigurací je Athlon 64 3000+ a jemu podobné. Možnost plynulého vykreslování, bych viděl jako zásadní plus mé práce. Jsem velice rád, že se tohoto cíle podařilo obstojně dosáhnout. Zároveň zůstala přiměřeně zachována kvalita výsledku s ohledem na optické jevy, které přináší použitá technologie. Algoritmus adaptivního vyhlazování jsem sám jak navrhl, tak implementoval.

Textury objektů jsou navrženy vcelku jednoduše, stejně tak i výsledná animace. Právě v těchto dvou bodech bych viděl další možnosti pro pokračování práce. Animaci by bylo vhodné kvalitně klíčovat. Možnosti dalšího vývoje zde však zdaleka nekončí. Bylo by jistě zajímavé implementovat podporu zpracování na paralelních systémech s více procesory či navrhnout např. FPGA chip, který by realizoval nejnáročnější operace. Mohl by být použit kupříkladu pro výpočet průsečíku paprsku a polygonu. Pak by bylo možné umísťovat do scény i komplexní a složité objekty.

Práce na bakalářském projektu byla pro mne osobně velkým přínosem. Díky jejímu vypracování jsem prohloubil své dosavadní znalosti s oblasti informačních technologií, která mne nejvíce zajímá. Měl sem možnost důvěrně se seznámit s použitým programovacím jazykem. Získané zkušenosti a znalosti mě jistě posunuly o mílový krok dopředu ve snaze zvládnutí rozsáhlé problematiky počítačové grafiky.

Literatura

[1] Žára, J., Beneš, B., Sochor, J., Felkel, P. *Moderní počítačová grafika.* 2, Brno, Computer Press

[2] Glassner, A., S., An introduction to ray tracing, 9, San Francisco, Califonia, Morgan Kaufmannn Publisher, Inc.

[3] Lengyel, E. Mathematics for 3D game progamming and computer graphics. 2, Hingham, Massachusetts, Charles River media, Inc.

Seznam příloh

Příloha č. 2 – datové typy

```
//typ pro uchov8n9 a yapod5en9 dat objektu
typedef struct Object {
        char * info;                            //ukazatel na specifick0 informace týkající se objektu
        struct ObjectProcs  * procs;
        struct Surf * surf;
}Object;



//struktura pro používané funkce nad objekty
typedef struct ObjectProcs {
        int (* Intersection) ();        //ukazatel na funkci, která vypočítá průsečík s objektem
        Vector (* Normal) ();           //ukazatel na funkci, která vrací normálový vektor
}ObjectProcs;



//typ pro uchování informací o nalezeném průsečíku
typedef struct Isect {
        Flt t;                  //násobek jednotkového směrového vektoru paprsku pro určení průsečíku
        Object * object;    //reference na objekt jehož se průsečík týká
}Isect;



//typ pro reprezentaci materiálu objektu
typedef struct Surf {
   Vector (* GetDiffuse) ();     //ukazatel na funkci generující difůzní vlastnosti materiálu
   Vector r;
        Flt rf;
        int hb;
}Surf;
```

```
//typ pro informace o umístění pixelu
typedef struct SqPoint {
   int x,y;
}SqPoint;


//typ pro uchování objektu a příspěvku barvy
typedef struct ObjectColor {
   Object * object;
   Color* col;
}ObjectColor;

//typ pro rohy čtverce
typedef struct SqPointColor{
   SqPoint p;                                   //umístění rohu
   Color c;                                     //výsledná barva
   ObjectColor *objectColor[RECURSION+1];       //uchování referencí na objekty a jejích
                                                //barevné příspěvky
   int shadowSpec[MAX_LIGHTS+1];                //stavy světel
}SqPointColor;

//základní typ pro metodu adaptivního vyhlazování
typedef struct Square{
   SqPointColor* corner[4];       //pole rohů čtverce
   struct Square* square[4];      //pole možných vnitřních čtvercu čtverce
}Square;
```