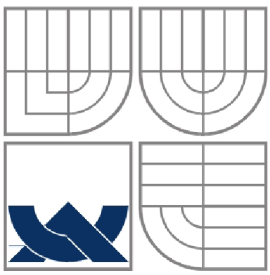
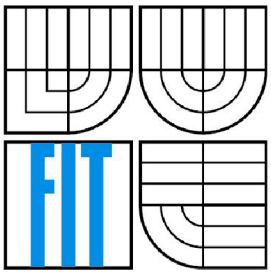


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PHOTON MAPPING

PHOTON MAPPING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ONDŘEJ NEČAS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2009

Abstrakt

V rámci této práce byla provedena praktická implementace algoritmu photon mapping. Pro dosažení kvalitnějšího výstupu byly zkoumány některé základní a pokročilejší metody globálního osvětlení. Tyto náročné algoritmy jsou často prakticky nepoužitelné a je nutná jejich optimalizace. Základem praktické implementace je optimalizace raytraceru. Vzorky nepřímého difuzního osvětlení počítané metodou Monte Carlo je možné mezi sebou interpolovat s použitím vhodné techniky.

Klíčová slova

globální osvětlení, photon mapping, sledování paprsku, světelné mapy, kd-strom, final gathering, irradiance caching, kaustiky

Abstract

This thesis deals with practical implementation of photon mapping algorithm. To achieve better results some basic and some more advanced methods of global illumination has been examined. These time demanding algorithms are often practically unusable and their further optimization is necessary. Optimized ray tracer is essential for practical implementation. Computing diffuse interreflection by Monte Carlo sampling is also very time demanding operation. Therefore it is appropriate to use it along with proper interpolation.

Keywords

global illumination, photon mapping, ray tracing, light maps, kd-tree, final gathering, irradiance caching, caustics

Citace

Ondřej Nečas: Photon mapping, diplomová práce, Brno, FIT VUT v Brně, 2009

Photon mapping

Prohlášení

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

……
Ondřej Nečas
26. května 2009

Poděkování

Děkuji vedoucímu Panu Adamu Heroutovi za odborné rady a ochotu při konzultacích.

Obsah

1 Úvod

Jeden z nejdůležitějších faktorů ovlivňujících celkový vjem počítačově generované scény je osvětlení. Osvětlení pomáhá pro lepší pochopení prostorového uspořádání scény. Vhodným modelem osvětlení je dodán výsledku realistický vzhled. Výpočet osvětlení je však poměrně náročná úloha a proto byla navržena řada algoritmů pro jeho efektivní výpočet. Metody se mohou dělit na stochastické a na metody konečných prvků. Všechny tyto algoritmy se snaží přiblížit reálnému výsledku zvýšením počtu náhodných vzorků, detailnějším dělením scény, vysláním více fotonů nebo sledováním paprsků do větší hloubky.

Tato práce se zabývá praktickou implementací algoritmu photon mapping. V první části této práce jsou shrnuty algoritmy globálního osvětlení. Existující raytracer není problém rozšířit tak, aby byl schopen otestovat některé základní schopnosti algoritmu photon mapping. Konstrukcí efektivního raytraceru se zabývá kapitola věnovaná rychlému nalezení nejbližšího průsečíku paprsku se scénou jakož to základního kamene efektivního raytraceru. Představeny jsou některé struktury pro urychlení nalezení průsečíku paprsku se scénou. Velká část je věnována právě kd-stromu, který patří mezi nejefektivnější struktury k rychlému nalezení průsečíku. Pro dosažení realistického vzhledu scény je nutné použití měkkých stínů. Toto souvisí s efektivním vzorkováním světelných zdrojů, které je popsáno v kapitole modelování světel. Samotný algoritmus photon mapping je vhodný spíše pro rychlou vizualizaci globálního osvětlení scény. Fotorealistické výsledky produkuje až ve spojení s některou pokročilou technikou jako například final gathering. Final gathering se pokouší obejít některé problémy algoritmu photon mapping. Tímto problémem je nepřesný odhad irradiance při přímé vizualizaci mapy fotonů. Tento algoritmus je však příliš pomalý. Na podobném principu funguje i algoritmus irradiance caching, kterému je věnována samostatná kapitola. Problémem jak tyto algoritmy skloubit dohromady se zabývá kapitola návrhu. Detailní informace o implementaci jsou přiloženy v programové dokumentaci vygenerované programem Doxygen.

2 Globální osvětlení

V následující kapitole je popsán princip základních algoritmů výpočtu globálního osvětlení. Algoritmu photon mapping je věnována samostatná kapitola. V poslední kapitole jsou popsány některé algoritmy, které jsou pro výpočet osvětlení klíčové. Patří sem například efektivní výpočet průsečíku paprsku se scénou, modelování a vzorkování světel. Je zde také popsán princip algoritmu irradiance caching.

2.1 Algoritmy globálního osvětlení

Soubor algoritmů pro výpočet realistického osvětlení scény se nazývá globální osvětlení. Objektové metody zobrazí pouze jednotlivé objekty, nejčastěji od nejvzdálenějších po nejbližší, bez jakékoli souvislosti mezi nimi. Obrazové metody pracují na úrovni pixelů. Každý pixel je zpracováván samostatně. Tuto třídu reprezentují metody na principu sledování paprsku. Simulují jevy jako jsou ostré stíny, odraz a lom světla. I když je zde patrná jistá interakce mezi objekty, tak se neoznačují jako globální, ale pro některé konkrétní scény mohou produkovat vysoce realistické výsledky.

V praxi se však mezi metody globálního osvětlení řadí metody schopné výpočtu nepřímých difuzních odrazů nebo výpočtu kaustik. Mezi tyto metody patří kromě photon mappingu také radiosita, path tracing, bidirectional path tracing, distributed raytracing a jako příklad nerealistické metody můžeme uvést ambient occlusion. Mnohé metody globálního osvětlení jsou schopny fyzikálně přesného modelu osvětlení a proto produkují fotorealistické výsledky. Avšak jsou mnohem výpočetně náročnější a výpočet některých scén v rozumné kvalitě by mohl být neúnosný i pro dnešní výkonné systémy.

2.1.1 Sledování paprsku (Ray tracing)

Sledování paprsku nepatří mezi metody globálního osvětlení, ale jeho základní principy se používají v mnohých níže popsaných algoritmech. Často se používá ve fázi finálního zobrazení scény, kdy se barva každého pixelu počítá samostatně. Tento algoritmus je snadno paralelizovatelný, proto je tu potenciál pro urychlení výpočtu na výkonných paralelních systémech. Zobrazení scény touto technikou je poměrně rychlé, avšak až na některé specializované výkonné systémy není použitelné pro zobrazení v reálném čase. Výsledkem zobrazení metodou sledování paprsku jsou ostré stíny. Metoda si také poradí se zrcadlovými odrazy a průhlednými objekty s různým indexem lomu.

Jak bylo řečeno algoritmus zpracovává obraz pixel po pixelu. Díky tomu je schopen zobrazit jakýkoli matematicky popsaný objekt u něhož lze určit průsečík s paprskem. Právě efektivní výpočet průsečíku paprsku s objektem je základem rychlého algoritmu - toto platí pro všechny algoritmy výpočtu osvětlení. Naivní přístup k nalezení nejbližšího průsečíku paprsku s objektem by spočíval ve výpočtu průsečíků se všemi objekty a následného určení toho nejbližšího. Takto se však provádí významné množství zbytečných výpočtů, proto bylo navrženo několik technik pro urychlení tohoto výpočtu.

2.1.1.1 Ray casting

Algoritmus představil poprvé Arthur Appel v roce 1968 [4]. Jedná se o nerekurzivní variantu sledování paprsku, kdy se pouze určí průsečík paprsku z kamery s nejbližším objektem a odhadne se

přímé osvětlení pomocí stínových paprsků.. Nesimulují se tedy jevy jako zrcadlové odrazy a průhlednost. Slouží k rychlému zobrazení scény s ostrými stíny.

2.1.1.2 Ray tracing

Pod tímto názvem bývá označován postup sledování paprsků z oka pozorovatele směrem do scény, tak dlouho dokud paprsek nenarazí na zcela difuzní materiál nebo neopustí scénu. Prakticky se však paprsek sleduje jen do nějaké úrovně. Tento algoritmus představil roku 1980 Turner Whitted [5].

```
Color TraceRay(ray)
{
        if(depth < maximal depth AND found intersection)
        {
                reflected component = 0.0
                transmitted component = 0.0
                diffuse component = 0.0
                if(R > 0.0)
                        reflected component = R * TraceRay(reflected ray)
                if(T > 0.0)
                        transmitted component = T * TraceRay(transmitted ray)
                if(D > 0.0)
                        for all lights do
                                if(intersection is not shaded in regard to the light)
                                        diffuse component  +=  D * light contribution
                return reflected component + transmitted component + diffuse component
        }
        else
                return background color
}
```

Algoritmus 1: Rekurzívní algoritmus sledování paprsku: R je reflexní koeficient, T je koeficient průhlednosti a D je difuzní koeficient.

Paprsek vyslaný z oka pozorovatele přes matici pixelů v rovině kamery se rekurzivně dělí na paprsek zrcadlově odražený a parsek procházejí skrz průhledný materiál. Pokud nenastane průsečík, bude mít pixel barvu pozadí, jinak bude výsledná barva součtem příspěvků s těchto paprsků a difuzního osvětlení. Z místa průsečíku je vyslán ještě tzv. stínový paprsek směrem do světla pro odhad přímého difuzního osvětlení v tomto bodě. Jestliže je tento bod zastíněn vzhledem k vyšetřovanému světlu, je příspěvek energie ze světla nulový. Difuzní složka bude tvořena příspěvky všech světel ve scéně, proto bude výpočet se zvětšujícím se počtem světel náročnější.

Výsledná barva bude dána vztahem:

$$pixelColor = R * reflectedColor + T * transmittedColor + D * diffuseColor$$

Aby rovnice vyhovovala zákonu o zachování energie musí také platit:

$$(R + T + D) \leq 1.0$$

Hodnota $1.0 - (R + T + D)$ odpovídá množství energie pohlcené tímto povrchem.

Často se však používá model doplněný spekulární složkou světla, která napodobuje zrcadlový odraz světelného zdroje. Její výpočet je však podstatně méně náročný než výpočet skutečné reflexe a výsledek je mnohdy dostačující.

2.1.1.3 Distributed ray tracing

Samotný ray tracing používá jeden paprsek na jeden pixel obrázku. U distribuovaného ray tracingu se paprsků používá více. Směr paprsků je dán specifickou distribuční funkcí pro každý efekt. Není docíleno lepšího modelu osvětlení, ale výstupní obrázky lépe odpovídají reálným efektům.

- Paprsky z kamery
  - Supersampling (antialiasing)
  - Pohybové rozostření (distribuce paprsků v čase)
  - Hloubka ostrosti (distribuce paprsků v prostoru čočky)
- Stínové paprsky
  - Měkké stíny (vzorkování pozice světla)
- Odražené nebo lomené paprsky
  - Matné povrchy
  - Rozklad světla (distribuce ve frekvenčním spektru)

2.1.2 Path tracing

Metodu představil v roce 1986 James T. Kajiya v dokumentu *The rendering equation* [15]. Metoda je velice fyzikálně přesná a používá se pro generování referenčních obrázků. Algoritmus je schopen simulovat široké spektrum optických jevů. Hlavní nevýhodou je dlouhá doba výpočtu. Mezi další rysy patří produkce obrázků s typickým vysokofrekvenčním šumem, který se dá potlačit dodatečnou filtrací.

Princip algoritmu je podobný jako sledování paprsku. Z oka pozorovatele se vyšle paprsek, jehož cesta se sleduje přes scénu dokud nedorazí do nějakého světelného zdroje. Na rozdíl od sledování paprsku se při průsečíku paprsku s difuzním materiálem nezanedbává nepřímé difuzní osvětlení, ale vygeneruje se náhodný paprsek, který pokračuje dále dokud nedorazí do světla. Tento princip lze snadno rozšířit na ostatní BRDF modely materiálu. Světlo v této scéně nemůže být bodové, protože je nulová pravděpodobnost, že jej nějaký paprsek protne. Světla musí tedy být výhradně plošná. Nevhodná pro tento algoritmus bude například scéna s místností osvětlenou žárovkou. Pravděpodobnost, že náhodný paprsek protne objekt žárovky je malá a proto výpočet bude extrémně náročný.

Na podobném principu jako path tracing pracuje metoda zvaná light tracing. Zde jsou zase paprsky náhodně vysílány ze světelných zdrojů a sledovány přes scénu dokud se neprotnou s rovinou kamery. Tento přístup se však samotný nepoužívá.

Kombinací těchto dvou přístupů dostaneme metodu zvanou bidirectional path tracing. Paprsky jsou vysílány směrem ze světla. Paralelně jsou vysílány paprsky z oka pozorovatele. Náhodným propojením průsečíků paprsků ze světla a paprsků z oka pozorovatele dostáváme kompletní cestu. Propojení musí respektovat předpoklad, že paprsky ze světla nesou zářivý tok a paprsky z oka sbírají radianci. Algoritmus bidirectional path tracing může být pomalejší než samotný path tracing, zato však podává dobré výsledky při simulaci reflexních povrchů.

2.1.3 Radiosita

Fyzikálně přesná metoda pro výpočet globálního osvětlení. Počítá pouze difuzní složku světla. Je odvozená od fyzikálního přenosu tepla mezi povrchy. Její výpočet je poměrně časově náročný a výsledek je pohledově nezávislý. Difuzní osvětlení pro statickou scénu stačí spočítat jednou a později

stačí pouze používat předpočítaný model osvětlení pro průlet touto scénou. Výpočet osvětlení pro dynamické scény touto metodou se musí opakovat pro každý snímek. Metoda se také stává značně neefektivní pro scény s komplexní geometrií.

Pro výpočet osvětlení touto metodou je nutné rozdělit povrchy objektů scény na malé plošky (patches). Potom se postupně pro každou plošku vypočítá energie touto ploškou přijímaná což odpovídá součtu energie vyzářené ze všech ostatních plošek vzhledem k této plošce. Kolik si tyto plošky vzájemně předají energie závisí na jejich vzdálenosti, ploše a vzájemnému úhlu natočení. Pokud jsou plošky vzájemně zastíněny je příspěvek energie nulový.

Rovnice radiosity:

$$B_i = E_i + R_i \sum_{j=1}^{n} B_j F_{ij}$$

B_i - Celková energie vyzařovaná z povrchu (radiosita)

E_i - Vyzařovaná energie samovolně z plošky i .

R_i - Odrazivost povrchu nebo také difuzní faktor.

$B_j F ij$ - Část energie vyzařované z plošky B_j přijímané ploškou B_i .

F_{ij} - Form faktor, konstanta závislá na vzájemné viditelnosti dvou plošek. Určuje jakou část energie vyzářené ploškou j přijme ploška i .

Výpočet radiosity potom vede k systému mnoha lineárních rovnic. Analytické řešení této soustavy je však velmi náročné. Proto se používá řada různých vylepšení jak výpočet urychlit. Například progresivní radiosita je iterativní algoritmus. V každé iteraci se přičte k výsledku další odraz světelné energie a pomalu se konverguje k úplnému řešení. Pro zlepšení výsledku se používá adaptivní dělení modelu, kdy v místech se v místech s velkými rozdíly intenzity osvětlení použije jemnější dělení ploch.

2.1.4 Ambient occlusion

Nerealistická technika simulace osvětlení. Odhaduje osvětlení pouze podle míry zastínění vyšetřovaného bodu. Algoritmus lze snadno implementovat v grafických akcelerátorech, proto je také vhodný jako efekt globálního osvětlení počítaný v reálném čase.

Výpočet se může dělat buď stochastickou metodou, kdy se z vyšetřovaného bodu náhodně vysílají paprsky s cosinem váženou distribuční funkcí. Podle poměru počtu paprsků, které naleznou průsečík s objektem a paprsků, které průsečík nemají, se odhadne míra zastínění. Jiný přístup může vypadat tak, že se do vyšetřovaného bodu umístí kamera s úhlem pohledu 180°, objekt se vyrenderuje v černé barvě proti bílému pozadí a z obrazu se určí cosinem vážený průměr intenzit pixelů, který představuje hodnotu osvětlení.

2.2 Photon mapping

Algoritmus výpočtu globálního osvětlení, který v roce 1996 prezentoval Henrik Wann Jensen [3]. Algoritmus je navržen velmi obecně a proto umí realisticky simulovat téměř jakýkoli viditelný optický jev. Kromě nepřímého osvětlení, které dobře simulují zejména radiozitní metody, zvládá bez problémů kaustiky, rozklad světla, rozptyl světla pod povrchem objektů nebo simulaci volumetrických materiálů. Scéna může také obsahovat procedurálně definované objekty. Výhodou algoritmu je oddělení informace o osvětlení od geometrie objektů, proto algoritmus efektivněji

pracuje se složitými scénami. Některé jevy lze sice efektivněji simulovat jinými algoritmy, ale velkou výhodou photon mappingu je, že tyto metody sdružuje a umožňuje k nim přistupovat jednotně.

Výpočet probíhá ve dvou fázích. Nejprve je vytvořena mapa fotonů nesoucí informaci o zářivém toku ve scéně. Následně se sledováním paprsků vytvoří výsledný obraz scény. Využití mapy fotonů však není omezeno na pouhou vizualizaci scény sledováním paprsku, ale může se využít některých pokročilejších technik jako například kombinace mapy fotonů a algoritmu path tracing nebo použití mapy fotonů v kombinaci s algoritmem irradiance caching produkuje kvalitní výstup v rozumném čase. Fotony tedy určují množství dopadající energie na dané místo (radiant flux). Oproti tomu paprsek vyslaný z oka pozorovatele sbírá radianci.

2.2.1 Photon tracing

V první fázi algoritmu se vytvoří mapa fotonů. Fotony jsou vysílány ze všech světel náhodně po scéně a je sledována jejich cesta dokud nedojde k jejich pohlcení. Vlastnosti materiálu určují jestli bude foton pohlcen nebo jaký bude jeho směr po interakci s materiálem. Přesnost a detailnost osvětlení určuje počet vyslaných fotonů. Mapa fotonů může být později uložena do souboru pro opětovné použití. Při uložení informace o světle ze kterého byl foton vyslán, lze snadno měnit charakter osvětlení pouhou transformací energie fotonů daného světelného zdroje.

2.2.1.1 Počáteční energie fotonů

Každé světlo ve scéně má danou energii pro jednotlivé barevné složky RGB. Energie jednoho fotonu je zlomek energie světla:

$$\Phi_P = \frac{\Phi_S}{n}$$

Kde Φ_S je energie světla a n je počet fotonů ze světla vyslaných. Pokud se ve scéně nachází více světelných zdrojů je vhodné vyslat ze světla počet fotonů úměrný jeho energii, protože každý foton zabírá v paměti stejné místo, proto je značně neefektivní vysílat ze slabého světelného zdroje stejné množství fotonů jako ze zdroje podstatně silnějšího. Silnější světlo se větší mírou podílí na osvětlení scény a proto je vhodné jeho vliv modelovat přesněji – vysílat z něj více fotonů. Při vysílání většího počtu fotonů by však počáteční energie byla příliš malá a v průběhu sledování by docházelo k velkým nepřesnostem, proto je výhodnější vysílat fotony s plnou energií světelného zdroje a až následně podělit jejich energii počtem vyslaných fotonů ze zdroje.

Fotony je možné generovat dvěma základními způsoby. Pokud bude zadán cílový počet vysílaných fotonů, rozpočítá se počet fotonů vysílaných jednotlivými světly. Protože dopředu známe kolik fotonů bude z jakého světla vysláno, je tento přístup vhodný pro paralelizaci. Problémem je, že paměťové nároky a kvalitu výstupu určuje počet fotonů uložených do fotonové mapy, který se může podstatně lišit od počtu vyslaných fotonů. Tento problém je možno vyřešit vysíláním fotonů z náhodně vybíraného světla dokud nebude mapa fotonů naplněna. Pro optimální algoritmus bude nutné tyto přístupy kombinovat.

2.2.1.2 Interakce fotonů s objekty

Po nalezení průsečíku dráhy fotonu s nejbližším objektem je nutné rozhodnout o dalším směru fotonu. Pokud počítáme pouze s difuzním odrazem, tak máme dvě možnosti. Buď se všechny fotony odrazí náhodným směrem v poloprostoru normály a jejich energie se vynásobí s difuzním faktorem nebo se odrazí jen část fotonů úměrná difuznímu faktoru a jejich energie se nezmění. Druhá možnost

je výhodnější, protože jinak se energie fotonů bude rychle snižovat. Směr difuzně odraženého fotonu se stanoví stejně jako při generování paprsku s difuzního zdroje světla.

Podobně se postup rozšíří o reflexní složku a průhlednost. Materiál povrchu je popsán difuzním faktorem, faktorem průhlednosti a zrcadlového odrazu. Každý faktor určuje pravděpodobnost, že nastane právě daný typ odrazu paprsku, proto součet těchto faktorů musí být menší nebo roven jedné. Zbytek světla se pohltí materiálem. Realističtější model některých materiálů by se musel odvodit z Fresnelových rovnic, které určují vztah mezi odraženým a lomeným množstvím světla v závislosti na úhlu dopadu paprsku. Foton se ukládá pouze v případě difuzního odrazu, protože při reflexním odrazu je pravděpodobnost, že se nějaké světlo odrazí do oka pozorovatele téměř nulová.

2.2.1.3 Praktická reprezentace fotonu

Struktura fotonu obsahuje souřadnice jeho pozice, jeho energii a také směr pod jakým foton na povrch dopadl. Směr dopadu fotonu je důležitý pro určení příslušnosti daného fotonu k danému povrchu, protože tato informace není explicitně uložena a stanovuje se pouze přibližně. Struktura fotonu by měla mít co nejmenší paměťové nároky, protože při velkém počtu fotonů můžou být tyto nároky neúnosné. Proto se používá různá komprese energie fotonu a směrového vektoru. Směrový vektor může být komprimován v podobě sférických souřadnic a uložen jako dva úhly, ale je zde nutný přepočet do sférických souřadnic a nazpět. Dnes v době 64 bitových systémů se možnosti podstatně změnily než jaké byly v době navržení algoritmu a taková komprese nemusí být nezbytně nutná.

2.2.1.4 Efektivní simulace kaustik

Při lomu nebo odrazu světla mohou paprsky světla směřovat k sobě nebo se naopak rozbíhat a tím můžou na okolních objektech vytvářet místa se zvýšenou nebo sníženou světelnou intenzitou. Tyto jevy se nazývají kaustiky. Příkladem může být třeba sklenice vody na přímém slunci, lupa nebo odraz způsobený zrcadlem vystaveným přímému slunečnímu svitu. Tyto efekty se dají efektivně simulovat právě pomocí algoritmu photon mapping. Protože reflexní povrchy způsobují vysoké koncentrace fotonů na malé ploše, je nutné pro kvalitní výstup použít oddělenou mapu fotonů pro difuzně odražené fotony a zvlášť ukládat fotony odražené reflexně. Kaustické efekty bývají tvořeny jemnými artefakty a je důležité použít detailnější mapu fotonů, aby tyto jemné artefakty nezanikly.

2.2.1.5 Rozptyl Světla pod povrchem

Některé částečně průhledné materiály mají značně nepravidelnou vnitřní strukturu danou množstvím mikroskopických částic. Tento jev je do jisté míry patrný na všech nekovových materiálech, které potom dostávají mléčný nádech. Světlo prochází objektem, kde se nekontrolovaně odráží a opouští objekt v těžko předvídatelné pozici a směru. Jensen v roce 1998 původně navrhl výpočet pomocí photon mappingu, který sice produkoval realistické výsledky, ale výpočet byl příliš náročný. V roce 2000 Jensen zkoumal alternativy k Monte Carlo metodám a později představil model BSSRDF [16], který výpočet rozptylu světla značně urychluje.

2.2.2 Photon rendering

Ve druhé fázi algoritmu se provede zobrazení scény a využije se k tomu informací z mapy fotonů. Základní verze vykreslení scény funguje podobně jako samotný algoritmus sledování paprsku. Paprsky z oka pozorovatele se rekurzivně sledují přes scénu a dělí se na odraženou, lomenou a difuzní složku osvětlení. Nejrychlejší verze algoritmu tuto difuzní složku odhadne přímo z fotonové

mapy. Výsledek je však často nevyhovující kvůli nepřesnému odhadu. Lepšího výsledku je dosaženo když se difuzní složka rozloží na přímé a nepřímé osvětlení. Přímé osvětlení se spočítá analyticky a nepřímé se odhadne z mapy fotonů. Pro nejvyšší kvalitu výstupu ani tento postup není dostatečný.

Kvalitního výstupu se dosáhne při vyžití Monte Carlo vzorkování polokoule k odhadu nepřímého osvětlení. Postup vzorkování polokoule je popsán v kapitole Irradiance caching. Při vzorkování polokoule potřebujeme znát příspěvek radiance v daném směru. Radiance se odhadne z mapy fotonů. Tato technika se nazývá final gathering. Aby se odhad irradinace v místech průsečíků paprsků polokoule nemusel několikrát opakovat, ukládá se tato hodnota pro pozdější použití. Pro urychlení výpočtu se final gathering kombinuje s některou technikou interpolace osvětlení jako například v algoritmu irradiance caching.

2.2.2.1 Odhad irradiance z mapy fotonů.

Odhad se provádí následovně. Najde se několik nejbližších fotonů od místa vzorkovaného bodu. Počet nejbližších fotonů a poloměr vyhledávání ovlivňuje výslednou kvalitu výstupu. Energie fotonů se integruje na ploše povrchu.

$$E = \frac{\sum_{i=1}^{n} {}_i\Phi_P}{A} = \frac{\sum_{i=1}^{n} {}_i\Phi_P}{\pi r^2}$$

Protože fotony hledáme uvnitř koule o poloměru r a předpokládáme, že se nacházíme na jedné rovině, je plocha povrchu na kterém sčítáme energii fotonů rovna právě obsahu kruhu. Do nějaké míry lze i zakřivený povrch považovat za rovinu, ale pokud je zakřivení příliš velké nebo se nacházíme blízko okraje, stává se odhad radiance značně nepřesný. Některé případy lze částečně eliminovat integrací energie fotonů na ploše mnohoúhelníku nebo filtrací. Tyto postupy se však hodí pouze pro přesnější odhad kaustik, které se musí odhadovat z mapy fotonů přímo. U ostatního nepřímého osvětlení počítaného vzorkováním polokoule se chyba přímého odhadu eliminuje integrací mnoha vzorků.

Detailní odvození odhadu osvětlení z mapy fotonů je popsáno například v knize *Realistic image synthesis using photon mapping* [1]. Důkaz, že je tento odhad správný by mohl vypadat následovně. Energie z bodového zdroje světla bude mít ve vzdálenosti r intenzitu:

$$E = \frac{\Phi_S}{4\pi r^2}$$

Energie jednoho fotonu bude:

$$\Phi_P = \frac{\Phi_S}{n}$$

Kde n je počet vyslaných fotonů. Tyto fotony vyslané ze světla narazí na zcela difuzní kouli o poloměru r a po odrazu zaniknou. Na kulové ploše bude uloženo tedy n fotonů. Povrch koule je známý a podle vzorce pro odhad irradiance z mapy fotonů dostaneme:

$$E = \frac{\sum_{i=1}^{n} {}_i\Phi_P}{A} = \frac{\sum_{i=1}^{n} {}_i\Phi_P}{4\pi r^2} = \frac{\Phi_S}{4\pi r^2}$$

Odhad z mapy fotonů odpovídá analytickému řešení a to jsme chtěli dokázat.

2.2.3 Datová struktura pro uložení fotonů

Obecně je pro uložení fotonů vhodná jakákoli struktura pracující s alespoň třemi rozměry a schopná rychlého vyhledávání nejbližších sousedů. Typicky jsou to různé modifikace binárního stromu jako

například oktalový strom, BSP-tree nebo kd-strom. Jensen doporučuje a používá pro uložení fotonů kd-strom. Kompletní implementace kd-stromu pro uložení mapy fotonů je popsána v knize [1].

kd-strom je binární strom schopný ukládat data jakékoli dimenze, ale pro vyšší dimenze existují vhodnější struktury. K vytvoření kd-stromu vede několik možných způsobů. Většinou se používá postup popsaný níže pouze s drobnými úpravami.
Prostorová data se seřadí podle aktuální osy a najde se medián. Ten se zvolí jako kořen stromu. Body nalevo se stávají levým podstromem a body napravo pravým podstromem. Dělící osa se cyklicky mění jak se postupuje níže. V každém uzlu bude uložená dělící osa a poloha dělící roviny. Kromě samotné konstrukce kd-stromu je důležitá pouze jedna operace nalezení nejbližších sousedů. Ostatní operace zde nejsou potřeba, protože struktura s fotony se vytvoří jednou poté co jsou známy všechny polohy fotonů a dále se nemění. Parametry funkce nalezení nejbližších sousedů jsou poloha vzorkovaného bodu, maximální poloměr vyhledávání a maximální počet hledaných fotonů. Pro udržování seřazeného seznamu nejbližších fotonů Jensen používá strukturu max heap.

2.3 Další algoritmy a techniky

V této kapitole jsou popsány algoritmy související z efektivním výpočtem osvětlení. Výpočet rychlého průsečíku paprsku se scénou nám dovolí buď vyšší kvalitu výstupu nebo rychlejší výpočet ve stávající kvalitě. V kapitole Modelování světel je popsáno jejich vzorkování a generování měkkých stínů. Kapitola irradiance caching se věnuje jak interpolaci osvětlení, tak i samotnému efektivnímu vzorkování osvětlení.

2.3.1 Světelné mapy

Informace o osvětlení scény je uložena v rastrových obrázcích. Světelné mapy jsou namapovány pomocí texturovacích souřadnic na objekty scény. Výsledná barva se získá vynásobením barvy textury s intenzitou světelné mapy v daném bodě. Detailnost osvětlení ovlivňuje rozlišení světelné mapy.

Světelné mapy se používají hlavně při zobrazení v reálném čase. Vizualizují se třeba architektonické objekty, kdy se jedná o kompletně statickou scénu nebo se využívají v počítačových hrách. První hra, která používala světelnou mapu byla hra Quake, vydaná v roce 1996. Světelná mapa byla pouze monochromatická a jednalo se jen o přímé osvětlení. Stejné efekty dnes není problém počítat v reálném čase na jakémkoli soudobém grafickém akcelerátoru. Výpočet přímého osvětlení v reálném čase byl poprvé využit u hry Doom 3 vydané roku 2004. Dynamické scény sice vypadají realističtěji, ale celková kvalita osvětlení je poměrně nízká.

Předpočítané osvětlení se může přiblížit fotorealistické kvalitě, ale jeho použití je omezeno výhradně na statické scény difuzní složku materiálu, která je pohledově nezávislá. Při změně geometrie je nutné veškeré osvětlení přepočítat. Mapa osvětlení se vypočítá pomocí nějakého komplexního algoritmu globálního osvětlení. Těchto algoritmů existuje celá řada. V dnešní době už většinou není třeba tímto způsobem počítat pouhé přímé osvětlení. Výsledek se potom uloží do souboru a při zobrazení se vlastně jen mapují textury. Je tedy možné zobrazit komplexní scény v reálném čase a ještě s fotorealistickým osvětlením.

Mezi nevýhody tohoto způsobu zobrazení patří zase větší paměťová náročnost, protože je nutné veškeré osvětlení uložit do textury. Na rozdíl od samotných textur, které se ve scéně často opakují, je osvětlení pro každý bod povrchu unikátní. Další nevýhodou světelných map je nutnost

generovat pro scénu texturovací souřadnice tak, aby se geometrie v texturovacím prostoru nepřekrývala, ale zato byly výstupní soubory kompaktní.

2.3.2 Nalezení nejbližšího průsečíku

Paprsek je zadán počátečním bodem $\vec{P}$ a směrovým vektorem $\vec{d}$. Hledáme takový kladný parametr t přímky $p=\vec{P}+t*\vec{d}$, který odpovídá průsečíku s nejbližším objektem scény. Směrový vektor je vhodné normalizovat, aby parametr t udával zároveň vzdálenost od počátku. Jak bylo řečeno naivní algoritmus spočítá průsečíky se všemi objekty ve scéně a z nich vyber ten, který je nejblíže počátku paprsku. Takové schéma odpovídá lineární časové složitosti. Z toho plynou neúnosné časové nároky výpočtu pro složité scény.

Pro urychlení výpočtu nejbližšího průsečíku bylo navrženo několik algoritmů. Optimální algoritmus není možné prakticky naimplementovat. Složitost nejlepších algoritmů se blíží k logaritmické složitosti. Výkonnost jednotlivých algoritmů často velkou mírou závisí na zvolené scéně, rozmístění objektů ve scéně nebo náročnost výpočtu samotného průsečíku s objektem scény. Obecně se algoritmy skládají ze dvou fází. V první fázi se sestrojí pomocné struktury pro efektivní průchod paprsku a nalezení průsečíku ve fázi druhé. Sestrojení pomocných struktur by mělo odpovídat použití. Pro dynamické scény budeme požadovat rychlé sestrojení těchto struktur. Zaplatíme za to však o něco pomalejším výpočtem druhé fáze. Podobně zase pro statické scény s velkým počtem objektů, kde je nutné vysílat velký počet paprsků, je přijatelná delší konstrukce pomocných struktur. Pomocné struktury se vytvoří pouze jednou na začátku výpočtu a jejich kvalitní návrh může značně urychlit výpočet průsečík.

Nejpoužívanější algoritmy lze rozdělit do tří základních kategorií: obalová tělesa, neadaptivní dělení prostoru, kam patří mřížky a algoritmy adaptivně dělící prostor.

2.3.2.1 Obalová tělesa (Bounding volumes)

Složité objekty scény můžeme zabalit do obalových těles. Průsečík paprsku s obalovým tělesem by měl být značně jednodušší než s vlastním objektem. Nejprve se určí se kterým obalovým tělesem nastane průsečík a potom se teprve zjišťuje průsečík s vnitřními objekty. Obalové těleso musí objekt zcela obklopovat a mělo by být co nejtěsnější. Prostory obalových těles se můžou vzájemně překrývat. Mezi běžně používané obalová tělesa patří například koule nebo obalový kvádr ať už osově zarovnaný nebo obecný rovnoběžnostěn.

Hierarchicky uspořádané struktury obalových těles se nazývají bounding volume hierarchies (BVH). Jejich automatická konstrukce byla popsána v dokumentu Automatic Creation of Object Hierarchies for Ray Tracing [6]. Často se však také používají při konstrukci grafu scény. Obalová tělesa jsou uložena ve stromové struktuře. První se testují na průsečík obalová tělesa na nejvyšší úrovni a postupně se dostáváme k samotným objektům.

Nevýhodou obalových těles je jejich chaotické rozmístění ve scéně. U paprsku protínajícího více obalových těles by bylo vhodné nejprve testovat objekty bližší počátku paprsku. Tato informace se musí explicitně určit a někdy se bude muset vyšetřit více uzlů, protože nebude možné určit, který z uzlů je blíže počátku paprsku.

2.3.2.2 Neadaptivní dělení prostoru - mřížky (Grids)

Jako alternativa k obalovým tělesům se nabízí algoritmy dělení prostoru do pravidelné mřížky. Buňky prostoru se nazývají voxely. Konstrukce pravidelné mřížky se zdá být poměrně jednoduchá a průchod

touto pravidelnou mřížkou je velice efektivní. Používá se k tomu algoritmus podobný Bresenhamovu algoritmu rasterizace úsečky. Dělení prostoru do pravidelné mřížky má však celou řadu nevýhod.

Krok mřížky se zvolí na začátku výpočtu podle hustoty objektů v dané ose. Odtud plyne hlavní problém, že algoritmus nerespektuje rozmístění objektů ve scéně a hodí se pouze pro scény s pravidelnou distribucí objektů ve scéně. Další nevýhodou jsou vysoké paměťové nároky, protože je vytvořeno velké množství prázdných voxelů. Procházení řady prázdných voxelů zbytečně zpomaluje výpočet, proto byly navrženy techniky pro přeskočení prázdných voxelů. Každý voxel má uloženou vzdálenost k nejbližšímu neprázdnému voxelu. Používá se i varianta, kdy se uloží počet neprázdných voxelů v každém s šesti směrů. Tím se však paměťové nároky podstatně zvýší.

Problém s nepravidelným rozmístěním objektů ve scéně je možné řešit pomocí nepravidelné mřížky. Hustota dělení prostoru bude závislá na histogramu objektů podél vybrané osy. Nevýhodou je nemožnost použití efektivního algoritmu procházení navrženého pro pravidelné mřížky. Rekurzivní mřížky zase rozdělí scénu podle počátečního kroku a adaptivně se dělí buňky s vysokou hustotou objektů. Tento přístup se liší od oktalových stromů tím, že každý voxel může obsahovat více než 8 subvoxelů. Průchod touto strukturou je však značně složitější než průchod pravidelnou mřížkou. Adaptivní mřížky se vytváří zcela jiným způsobem. Objekty jsou podle nějakého kritéria přiřazeny do shluků. Tyto shluky se sice rozdělí podle pravidelného kroku, ale nad obalovými tělesy shluků se vytvoří BVH.

2.3.2.3 Adaptivní dělení prostoru

Do této kategorie se řadí algoritmy rekurzivně dělící prostor scény na základě rozmístění objektů. Scéna se rekurzivně dělí tak dlouho, dokud není splněno ukončovací kritérium. Často to bývá maximální hloubka zanoření nebo počet objektů v listovém uzlu. Jmenovitě sem patří oktalové stromy, kd-stromy,BSP stromy (binary space partitioning trees), BIH (bounding interval hierarchy) nebo by sem mohli patřit také rekurzivní mřížky.

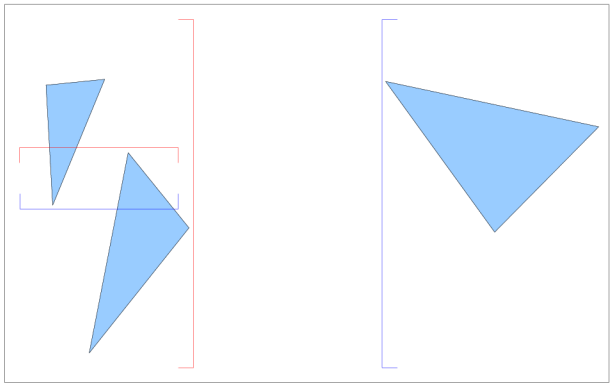
Oktalové stromy dělí obalový kvádr scény rekurzivně na 8 stejných buněk. Každá buňka může být potom dále rekurzivně dělena dokud není splněno ukončovací kritérium. Nevýhodou oktalového stromu oproti binárnímu je složitější algoritmus průchodu stromem. Každý nelistový uzel má pod sebou 8 uzlů a musí buď určit pořadí jejich průchodu. Binární stromy musí testovat pouze 2 synovské uzly. Výhodou oktalového stromu je jeho rychlá a jednoduchá konstrukce. Vhodný je také pro dynamické scény, protože ho lze snadno aktualizovat aniž by se musel budovat celý znovu.

Jak z názvu vyplývá BSP stromy jsou binární stromy. Původně byly navrženy pro výpočet viditelnosti a tím urychlení zobrazení scény v reálném čase. Prostor scény se vhodně dělí podle zvolené obecné roviny na dva podprostory. V jedné variantě algoritmu se jako dělící rovina vybere jeden z polygonů scény. Tato varianta není nejvhodnější pro hledání nejbližšího průsečíku, protože se objekty scény musí omezit pouze na polygony a také průchod touto strukturou je pomalejší, protože se musí hledat průsečík s obecnou rovinou v prostoru.

Další variantou BSP stromů jsou kd-stromy. Některé studie z poslední doby [7] ukazují, že právě kd-stromy podávají pro většinu testovaných scén nejlepší výsledky. Hlavním znakem kd-stromu je osově zarovnaná dělící rovina.

Konkurencí kd-stromům v některých scénách by mohl být algoritmus BIH [8]. Konstrukce stromu tímto algoritmem je velice rychlá a podobně jak některé inteligentní implementace kd-stromu zvýhodňují dělení prostoru s odseknutím prázdného prostoru, je tato vlastnost přirozenou součástí algoritmu BIH. Prostor se dělí podle dané osy vždy na polovinu. Neukládá se však dělící rovina, ale maximum objektů zleva a minimum objektů zprava.

Algoritmus procházení stromem je podobný procházení kd-stromu. Navíc obsahuje možnost, že paprsek neprotne ani levý ani pravý uzel. Podle autorů uvedeného článku podává algoritmus v mnohých scénách lepší výsledky než kd-stromy a je paměťově úspornější. Problémy má však ve scénách obsahující hrubé a jemné objekty zároveň.



Ilustrace 1: Dělení prostoru algoritmem BIH, červená závorka označuje maximum objektů zleva a modrá závorka označuje minimum objektů zprava.

2.3.3 Použití kd-stromu pro nalezení nejbližšího průsečíku

Asi nejrozšířenější strukturou pro urychlení výpočtu nalezení nejbližšího průsečíku paprsku se scénou jsou kd-stromy. Existuje mnoho algoritmů k vytvoření a průchodu stromem, jejichž výkonnost se může podstatně lišit. V následující kapitole je popsána varianta implementovaná v rámci této práce.

2.3.3.1 Konstrukce kd-stromu

Volba dělící roviny je zásadní pro efektivní průchod stromem. Kromě dělící roviny je nutné vhodně zvolit také osu podle které se bude prostor dělit. Osa podle které se bude prostor dělit se může jednoduše cyklicky střídat. To může vést u některých zploštělých scén k velice neefektivnímu rozdělení prostoru. Dělící rovina se může volit jako prostorový medián (dělení obalového kvádru na poloviny) nebo objektový medián (objekty se seřadí v dané ose a prostor se dělí mezi prostředními objekty). Oba tyto přístupy však nejsou pro hledání průsečíku příliš vhodné. Ideální rozdělení by bylo na základě cenové funkce tak, aby pro libovolný vržený paprsek byla námaha vynaložená k nalezení nejbližšího průsečíku konstantní.

Tomuto ideálu se snaží přiblížit heuristika SAH (surface area heuristic). Vychází z předpokladu, že pravděpodobnost protnutí voxelu paprskem je úměrná jeho povrchu. Voxel V se dělí na dva subvoxely V_L s N_L objekty a subvoxel V_R s N_R objekty. Každému voxelu náleží objekty, které ho protínají nebo leží v některé z jeho stěn. Objekty protínající více voxelů budou přiřazeny do všech těchto voxelů. Zvláštní případ tvoří polygony ležící v dělící rovině, které se na základě cenové funkce všechny přiřadí buď levému nebo pravému subvoxelu. Nejlepší možná dělící rovina p bude odpovídat minimu cenové funkce:

$$C(p) = K_T + K_T\left(\frac{SA(V_L)}{SA(V)}N_L + \frac{SA(V_R)}{SA(V)}N_R\right)$$

Kde K_T je cena (čas) jednoho kroku průchod paprsku, K_I je cena (čas) průsečíku paprsku s objektem a $SA(V)$ je povrch voxelu V. Konstanty K_T a K_I se musí změřit nebo experimentálně zvolit podle použitých algoritmů. Cenová funkce by měla také zvýhodňovat roviny odsekávající kus prázdného prostoru. Cenová funkce je lineární na libovolném intervalu, kde se nemění počet objektů.

Tímto se počet možných kandidátů omezí na dolní a horní hranice všech objektů v dané ose. Nejlepší dělící rovina se hledá přes všechny možné kandidáty a přes všechny tři osy. Nalezenou rovinou se aktuální voxel rozdělí na dva subvoxely. Každému se přiřadí přiřadí příslušné objekty. Voxely se takto rekurzivně se dělí dokud není splněno ukončovací kritérium $(C(p) > N * K_I)$.

Rozdělení objektů mezi synovské uzly je náročná operace, proto v dokumentu [9] používají seřazené seznamy událostí nikoliv přímo seznamy objektů. Událostmi jsou myšleny horní a dolní hranice objektů v dané ose a v daném voxelu. Polygony ležící v dělící rovině jsou reprezentovány pouze jednou planární událostí. Levému uzlu se přiřadí seznam všech horních události menších než dělící rovina a k nim příslušné dolní události. Pravému uzlu se přiřadí seznam všech dolních událostí větších něž dělící rovina a k nim příslušné horní události. Funkce hledající nejlepší dělící rovinu zároveň vrací informaci o tom na kterou stranu se mají přiřadit planární události (polygony). Tyto dva seznamy jsou už seřazené a není nutná další úprava. Zbylé události musí nutně patřit polygonům zasahujícím do obou uzlů. Obalový kvádr polygonu se ořeže levým voxelem a nové události se přiřadí do nového neseřazeného seznamu patřícího levému uzlu. Stejná operace se provede pro pravý uzel. Tyto seznamy se seřadí a spojí se s už vytvořenými seznamy událostí patřících výlučně jedné straně. Výsledné seznamy se předají rekurzivní funkci, která buduje strom. Seznamy událostí slouží zároveň pro hledání nejlepší dělící roviny a lze z nich snadno odvodit seznam objektů, kterým náleží. Pro každou osu existuje vlastní seznam událostí.

```
Node BuildTree( E , V )
{
        p = FindSplitPlane( E , V )
        if(Cost( p ) > K_I * N )
                return new Leaf Node( E )
        ( V_L , V_R ) = Split( V , p )
        ( E_L , E_R ) = Split( E , p )
        return new Node( p , BuildTree( E_L , V_L ), BuildTree( E_R , V_R ))
}
```

Algoritmus 2: Rekurzivní vytvoření stromu, každý uzel bude obsahovat dělící rovinu.

2.3.3.2 Procházení kd-stromem

Procházení stromem může být realizováno rekurzivně nebo s využitím zásobníku. Průchod začíná v kořenovém uzlu, který obsahuje obalový kvádr celé scény. Zjistí se se kterým subvoxelem nastane průsečík dříve. Tento subvoxel se prohledá. Pokud je nalezen průsečík, algoritmus končí. Jinak se pokračuje druhým subvoxelem pouze za předpokladu, že jej paprsek protíná, jinak algoritmus končí.

```
Intersection Traverse (node, ray)
{
	while(node is not null)
	{
		if(node is a leaf AND node is not empty)
		{
			intersection = GetNearestIntersection(node, ray)
			if(intersection is not null)
				return intersection
			else if(stack is not empty)
				node = Pop(stack)
			else
				return null
		}
		else
		{
			if(ray intersects both children)
			{
				node = closer child
				Push(stack, further child)
			}
			else if(ray intersects left child)
				node = left child
			else if(ray intersects right child)
				node = right child
			else if(stack is not empty)
				node = Pop(stack)
			else
				return null
		}
	}
}
```

Algoritmus 3: Nerekurzivní varianta procházení stromem s využitím zásobníku.

V efektivním algoritmu se snažíme minimalizovat počet výpočtů průsečíků s rovinami. Za předpokladu, že známe počátek paprsku uvnitř daného voxelu, musíme vypočítat průsečík pouze s dělící rovinou. Počátkem paprsku je myšlen skutečný počátek nebo bod na povrchu voxelu, ve kterém do něj paprsek vstupuje. Pokud existuje takový průsečík, stává se počátkem paprsku v druhém subvoxelu, který se zároveň vloží na zásobník a výpočet pokračuje prvním subvoxelem, který převezme počátek paprsku od otcovského uzlu. Pokud průsečík s dělící rovinou nenastane, pokračuje výpočet subvoxelem obsahujícím počátek paprsku.

2.3.4 Modelování světel

Samotný algoritmus sledování paprsku odhaduje difuzní složku pouze na základě přímého osvětlení. Pokročilejší algoritmy jako například photon mapping zohledňují i příspěvek nepřímého difuzního osvětlení. Zde se budeme zabývat pouze výpočtem přímého osvětlení. Světla ve scéně můžeme

modelovat dvěma základními modely: bodové světlo a difuzní světlo. Složitější modely světel budou definovány na základě omezení těchto základních modelů nebo jejich skládáním.

2.3.4.1 Bodové světlo

Nejjednodušší model světla je právě světlo bodové. Světlo má nulové rozměry. Definováno je polohou a energií. Energie se šíří z počátku všemi směry rovnoměrně.

```
do
{
        x=-1+2*ξ1
        y=-1+2*ξ2
        z=-1+2*ξ3
}while (x²+y²+z²>1.0)
```

Algoritmus 4: Generování náhodného vektoru: ξ_1, ξ_2, ξ_3 *jsou náhodná čísla v intervalu* $\langle 0,1 \rangle$.

Bodové světlo se vzorkuje směrem ze světla do scény generováním náhodných paprsků. Energie každého paprsku bude:

$$\Phi_P = \frac{\Phi_S}{n}$$

Kde Φ_S je energie světla a n je počet paprsků vyslaných ze světla. V případě obráceného vzorkování se energie světla rozloží na plochu A, v tomto případě na povrch koule:

$$E = \frac{\Phi_S}{A} = \frac{\Phi_S}{4\pi d^2}$$

Kde d je vzdálenost mezi vyšetřovaným bodem a pozicí světla.

2.3.4.2 Difuzní světlo

Podobně jako bodové světlo vyzařuje difuzní světlo energii z jednoho bodu, ale tento bod leží na povrchu nějakého objektu. Na rozdíl od bodového světla definovaného polohou a energií je difuzní světlo definováno navíc ještě normálou k povrchu v tomto bodě. Toto světlo odpovídá Lambertovu modelu osvětlení. Ploška je ze všech směrů viditelná pod stejnou intenzitou. Proto musí paprsky generované z tohoto světla směřovat do poloprostoru určeného normálou a musí být váženy kosinem úhlu, který svírají s normálou. Pravděpodobnost, že se vygeneruje paprsek svírající s normálou nějaký úhel je úměrná právě kosinu tohoto úhlu.

$$\theta = \arccos(\sqrt{\xi_1})$$
$$\phi = 2\pi\xi_2$$

Algoritmus 5: Generování kosinem váženého vektoru: ξ_1, ξ_2 *jsou náhodná čísla z intervalu* $\langle 0,1 \rangle$, θ *je úhel mezi vektorem a normálou,* ϕ *je úhel okolo normály.*

Tento algoritmus generování kosinem váženého vektoru je však vcelku neefektivní, protože používá goniometrické funkce, které jsou náročné na výpočet. Úhly je také nutné převést na vektor, což představuje další režii. Uvedený postup však vychází z algoritmu generování náhodného bodu uvnitř kružnice:

$$r = \sqrt{\xi_1}$$
$$\phi = 2\pi\xi_2$$

Algoritmus 6: Generování náhodného bodu uvnitř kružnice.

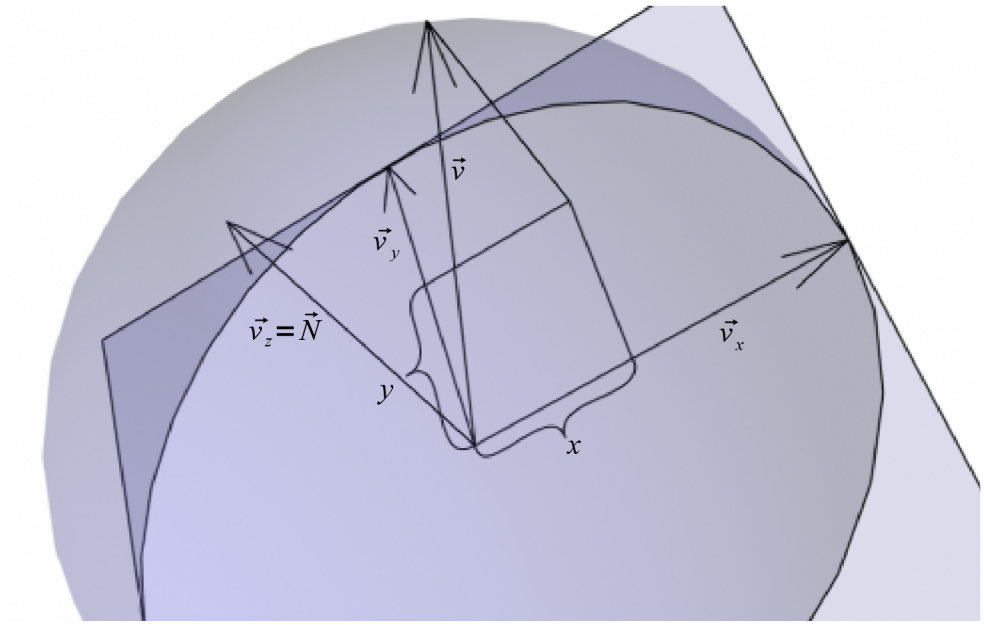
Z těchto dvou postupů vyplývá, že generování kosinem váženého vektoru není nic jiného než generování náhodného bodu uvnitř jednotkové kružnice a promítnutí tohoto bodu na jednotkovou polokouli, protože $r=\cos(\theta)=\sqrt{\xi_1}$.

do
{
$x=-1+2*\xi_1$
$y=-1+2*\xi_2$
}while $(x^2+y^2>1.0)$
$z=\sqrt{1.0-(x^2+y^2)}$

Algoritmus 7: Efektivní generování kosinem váženého vektoru.

Vygenerovaný vektor je v reálné scéně nutné otočit s ohledem na polohu světla. Prakticky stačí složky tohoto vektoru vynásobit s jednotkovými vektory v odpovídajících směrech a násobky sečíst.

$$\vec{v}=x*\vec{v}_x+y*\vec{v}_y+z*\vec{N}$$



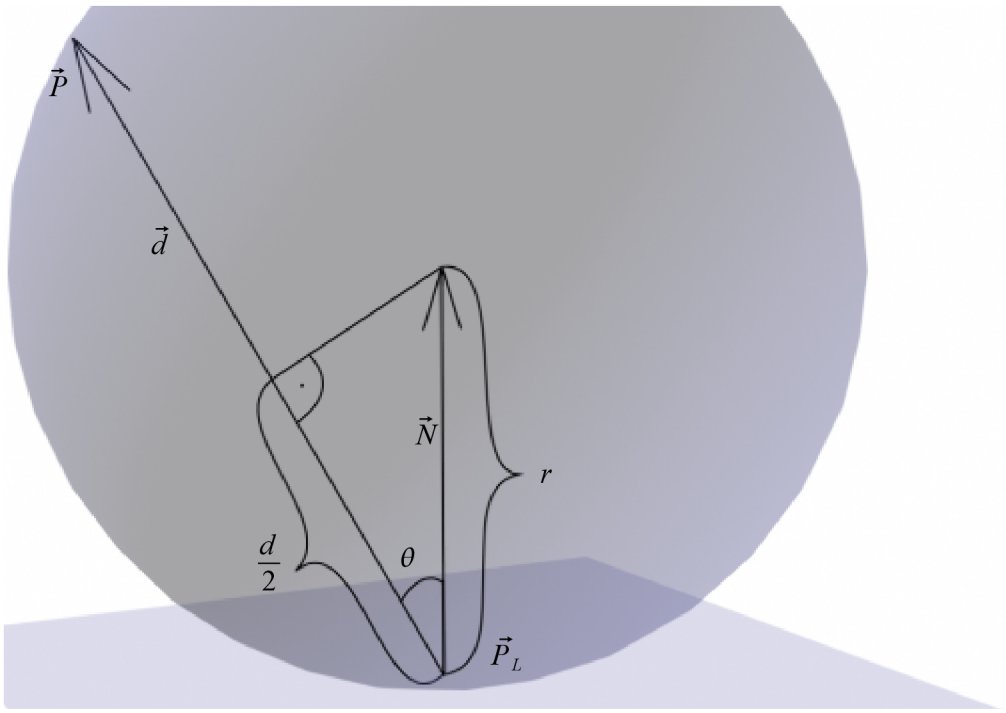
Ilustrace 2: Generování kosinem váženého vektoru a následné otočení do cílové polohy pomocí jednotkových vektorů.

Vzorkování světla v daném bodě scény není o mnoho složitější než vzorkování bodového světla. Vyzářená energie je úměrná kosinu úhlu daného směru s normálou, proto bude mít záření stejnou intenzitu právě na povrchu koule, která se dotýká bodu na povrchu objektu. Pro libovolný bod scény $\vec{P}$ stačí určit poloměr koule definované tímto bodem, pozicí světla $\vec{P}_L$ a středem koule ležícím na přímce určené normálou $\vec{N}$ v bodě $\vec{P}_L$. Z pravoúhlého trojúhelníku dostaneme poloměr této koule:

$$r=\frac{|\vec{d}|}{2\cos(\theta)}$$

Kde $|\vec{d}|$ je vzdálenost od světla a θ je úhel mezi normálou a vektorem $\vec{d}$. Intenzita osvětlení v bodě $\vec{P}$ tedy bude:

$$E=\frac{\Phi_S}{4\pi r^2}=\frac{\Phi_S\cos^2(\theta)}{\pi|\vec{d}|^2}$$



Ilustrace 3: Vzorkování difuzního světla.

2.3.4.3 Kuželové světlo

Kuželové světlo je definováno pozicí počátku, směrem a prostorovým úhlem. Směry všech možných paprsků vyplní v prostoru kužel. Paprsky v tomto kuželu mohou být distribuovány libovolně. Nejjednodušší model kuželového světla vychází ze světla bodového. Paprsky jsou distribuovány rovnoměrně do všech směrů, ale omezeny kuželem. Při obráceném vzorkování bodového světla se energie rozloží na povrch koule. U kuželového světla se jeho energie rozloží na plochu kulového vrchlíku.

$$S = 2\pi r v = 2\pi r^2(1-\cos(\theta))$$

Kde r je poloměr koule (vzdálenost mezi vzorkovaným bodem a pozicí světla), θ je úhel mezi směrovým vektorem a přímkou pláště kuželu. Výsledná intenzita světla v bodě bude:

$$E = \frac{\Phi_S}{2\pi r^2(1-\cos(\theta))}$$

2.3.4.4 Plošné světlo

Plošné světlo je difuzní světlo, které má definovaný povrch a energii. Energie se rovnoměrně rozloží po povrchu objektu, proto potřebujeme znát jeho plochu. Jako plošné světlo bývá často označováno světlo jedné plochy obdélníkového tvaru. Zde však bude myšleno světlo obecného tvaru v prostoru. Plošným světlem je v podstatě jakýkoli objekt scény vyzařující energii. Jakýkoli difuzní objekt scény může vyzařovat energii vlastní sečtenou se zlomkem energie na objekt dopadající (irradiance).

Paprsek ze světla do scény se generuje z náhodného bodu na povrchu objektu. Tento paprsek se generuje stejně jako pro difuzní světlo. Jeho energie bude dána podílem celkové energie objektu a počtu vyslaných paprsků.

Světla mající nenulovou plochu generují měkké stíny. Jejich vzorkování je podstatně složitější než u bodových světel, kde stačilo vyslat jediný stínový paprsek pro určení viditelnosti. Nabízí se dvě základní možnosti jak plošné světlo vzorkovat. První možností je vzorkovat ho stejně jako všechny

ostatní difuzní plochy pomocí Monte Carlo vzorkování polokoule. Bod na povrchu tohoto světla s plochou A bude kromě odražené energie vyzařovat vlastní energii:

$$\Phi_E = \frac{\Phi_s}{A}$$

Toto řešení může být pro vzdálený objekt s výrazně větší energií značně neefektivní. Druhou možností je místo náhodného střílení paprsků do prostoru s mizivou šancí, že paprsek protne dané světlo, generovat rovnoměrně body na povrchu objektu a každý považovat za samostatné světlo. Na povrchu objektu světla se vygeneruje n bodů. Každý tento bod se stává difuzním světlem s energií:

$$\Phi_P = \frac{\Phi_S}{n}$$

```
do
{
        u = ξ1
        v = ξ2
}while (u + v > 1.0)
P = u * V1 + v * V2 + (1 - (u + v)) * V3
```

Algoritmus 8: Generování náhodného bodu $\vec{P}$ uvnitř trojúhelníku definovaného třemi body $\vec{V}_1$, $\vec{V}_2$, $\vec{V}_3$.

```
# i ∈ 1,2,...,n
S0 = 0
Si = Si-1 + SA(Ti)
r = ξ0
if (r ≥ Si-1 ∧ r < Si)
{
        return RandomPoint( Ti )
}
```

Algoritmus 9: Generování náhodného bodu na povrchu objektu definovaného seznamem trojúhelníků T o velikosti n. Pole S je suma prefixů ploch všech trojúhelníků. Funkce $SA(T)$ vrací plochu trojúhelníku a funkce $RandomPoint(T)$ generuje náhodný bod uvnitř trojúhelníku T.

2.3.4.5 Měkké stíny

Podobně jako plošná světla je možné generovat efektivním způsobem měkké stíny i u světel bodových pouhým roztřesením jejich polohy v prostoru. Bodové světlo s měkkými stíny by se mohlo vzorkovat jako kulové difuzní světlo. Pokud je však světlo dostatečně vzdálené od vzorkované polohy můžeme použít několik zjednodušení. Pro toto vzdálené světlo potřebujeme znát poměr mezi zastíněnou a viditelnou částí. Analytické řešení je příliš složité, ale jako první se nabízí řešení pomocí Monte Carlo integrace. Místo generování bodů na povrchu koule budeme generovat n bodů uvnitř této koule. n bodů odpovídá n vzorkům. Každý vygenerovaný bod představuje nové bodové světlo se zlomkem energie světla původního. Energie světla se rozloží mezi tyto nová světla podle následující rovnice:

$$\Phi_P = \frac{\Phi_S}{n}$$

Tyto nová světla bychom mohli považovat za plnohodnotná světla a vzorkovat je jako světla bodová. Je však možno použít další zjednodušení. Budeme uvažovat vzdálenost těchto nových světel od vzorkovaného bodu pro všechna světla stejnou. Jako pozice vzorkovaného světla se vezme pozice původního bodového světla. Nyní se určí počet zastíněných n_S a viditelných n_V nových světel a odvodí se poměr $ratio = n_V / n_S$ Tím se značně zjednoduší výpočet výsledné intenzity osvětlení v daném bodě na:

$$E = \frac{ratio * \Phi_S}{4\pi d^2}$$

Pro měkké stíny kuželového světla se vztahy odvodí obdobně.

2.3.5 Irradiance caching

Při zobrazení scény pomocí metody sledování paprsku se v každém průsečíku paprsku s objektem vyhodnocují tyto základní složky: příspěvek odraženého a lomeného paprsku, přímé osvětlení a nepřímé difuzní osvětlení. Nepřímé difuzní osvětlení bývá někdy nahrazeno konstantní hodnotou, výsledek však není příliš realistický. Realističtějšího výsledku bývá dosaženo s použitím radiosity. Tento přístup je však nevhodný pro scény se složitou geometrií. Jako vhodnější metoda se nabízí aproximace difuzního osvětlení metodou Monte Carlo integrace.

Výpočet nepřímého difuzního osvětlení je přesto velice časově náročná operace. Na druhou stranu se ale nepřímé osvětlení na povrchu objektu mění jen pozvolna. Proto je možné pro urychlení výpočtu využít interpolaci/extrapolaci dříve spočítaných hodnot. Kromě volby efektivní metody interpolace je třeba rozhodnout ve kterém bodě bude nutné vypočítat hodnotu osvětlení novou a ve kterém je možné využít okolních hodnot pro odhad osvětlení.

Tato metoda byla představena v dokumentu *A ray tracing solution for diffuse interreflection* z roku 1988 [10]. Později se pro ni ustálil název irradiance caching. Původní algoritmus byl v roce 1992 ve článku *Irradiance gradients* [11] doplněn o použití gradientů ke zlepšení odhadu irradiance. Koncept efektivního ukládání a interpolace byl rozšířen na matně lesklé povrchy, kde se však ukládá radiance [12]. Tyto algoritmy jsou velice důležité pro efektivní výpočet globálního osvětlení. Jejich dalšímu vylepšení se věnují například studie [13][14].

2.3.5.1 Výpočet vzorků nepřímého osvětlení

Výpočet nepřímého osvětlení vychází z rovnice radiance:

$$L_r(\theta_r, \phi_r) = \int_0^{2\pi} \int_0^{\pi/2} L_i(\theta_i, \phi_i) f(\theta_i, \phi_i, \theta_r, \phi_r) \cos(\theta_i) \sin(\theta_i) \; d\theta_i \, d\phi_i$$

Kde θ je úhel mezi daným směrem a normálou, ϕ je úhel představující směr okolo normály, $L_i(\theta_i, \phi_i)$ je dopadající radiance a $f(\theta_i, \phi_i, \theta_r, \phi_r)$ je funkce BRDF.

Radiance je množství energie procházející danou plochou v daném směru za jednotku času. Jednotkou je $W\,sr^{-1}\,m^{-2}$. Integrál představuje součet radiancí násobených funkcí BRDF ze všech možných směrů polokoule. BRDF popisuje jaká část energie přicházející z libovolného směru se odrazí v daném výstupním směru. Veličina radiance popisuje vlastně výslednou barva pixelu obrázku. Dále budeme uvažovat pouze difuzní materiály pro které je tato hodnota nezávislá na směru.

Podle Lambertova modelu je radiance ve všech směrech konstantní. Proto musí být konstantní i funkce BRDF. Radiance potom bude záviset pouze na množství energie dopadajícího do daného bodu. Rovnice radiance se zjednoduší na:

$$L_r(\theta_r,\phi_r)=f_d\int_0^{2\pi}\int_0^{\pi/2} L_i(\theta_i,\phi_i)\cos(\theta_i)\sin(\theta_i)\ \ d\theta_i\ d\phi_i=f_d E_i$$

Kde f_d konstantní funkce BRDF, E_i je irradiance – množství energie dopadající do daného bodu. Jednotkou irradiance je Watt. Pro difuzní povrch bude množství odražené energie vzhledem k energii přijaté (diffuse reflectance) dáno vztahem:

$$\rho_d=\frac{\Phi_r}{\Phi_i}=\frac{L_r\int_0^{2\pi}\int_0^{\pi/2}\cos(\theta)\sin(\theta)\ \ d\theta\ d\phi}{E_i}=\pi f_d$$

Kde Φ_r je zářivý tok odražený a Φ_i je zářivý tok dopadající. Na tuto konstantu bude dále odkazováno jako na difuzní faktor. Aproximace irradiance pomocí Monte Carlo integrace se provádí vzorkováním příchozí radiance v náhodných směrech:

$$E_i=\frac{\pi}{n}\sum_{i=1}^{n} L_i(\vec{\omega}_i)$$

Směr $\vec{\omega}_i$ omega musí být kosinem vážený vektor. Jinou méně efektivní možností je použití rovnoměrně distribuovaných vektorů:

$$E_i=\frac{\pi}{\sum_{i=1}^{n}\cos(\theta_i)}\sum_{i=1}^{n}\cos(\theta_i)L_i(\vec{\omega}_i)$$

Ukázalo se, že mnohem efektivnější přístup představuje použití tzv. rozvrstveného vzorkování (stratified Monte Carlo sampling):

$$E_i=\frac{\pi}{M\,N}\sum_{j=1}^{M}\sum_{k=1}^{N} L_i(\theta_j,\phi_k)$$

Kde je směr $(\theta_j,\phi_k)=\left(\arccos\left(\sqrt{\frac{j+\xi_j}{M}}\right),2\pi\frac{k+\xi_k}{N}\right)$. Rozvrstvené vzorkování rozdělí vzorkovaný prostor na pravidelné oblasti. V každé oblasti se generuje náhodný vzorek. Výsledek bude tedy konvergovat ke stejnému řešení jako při zcela náhodném vzorkování, ale rozložení vzorků bude pravidelnější. Metoda Monte Carlo není obecně vhodná pro integraci funkcí s výraznými extrémy, protože se tyto extrémy nemusí do výsledku vůbec započítat. Příkladem může být objekt s vysokou energií avšak malých rozměrů a hodně vzdálený od vzorkovaného bodu. Pravděpodobnost vygenerování paprsku, který protne tento objekt je malá, ale pokud ho paprsek protne, změní se výrazně výsledek integrace.

2.3.5.2 Interpolace/extrapolace vzorků

Podobně jako u algoritmu photon mapping je i v algoritmu irradiance caching informace o osvětlení zcela oddělená od geometrie scény. Vhodní kandidáti podle kterých by se mohlo odhadnout osvětlení se vyhledávají v nejbližším okolí vzorkovaného bodu. Maximální poloměr prohledávání by měla být uživatelsky zadaná konstanta. Pokud se v okolí vzorkovaného bodu nenachází žádný jiný vzorek, musí se v tomto bodu vypočítat nová hodnota osvětlení. Pokud jsou však nalezeny nějaké vzorky, musí se ještě rozhodnout, jestli jsou vhodné pro interpolaci.

Vyloučení nevhodných vzorků se provádí na základě odhadu maximální chyby mezi daným bodem a normálou a testovaným vzorkem. Vzorek irradiance tedy kromě samotné hodnoty obsahuje ještě svoji pozici a normálu. Maximální chyba je odvozena z modelu koule jejíž jedna polovina je bílá a druhá černá. Uprostřed se nachází ploška směřující na rozhraní polokoulí. Ploška se může pohybovat nebo rotovat. Změna poměru černé a bílé barvy promítané na tuto plošku určuje

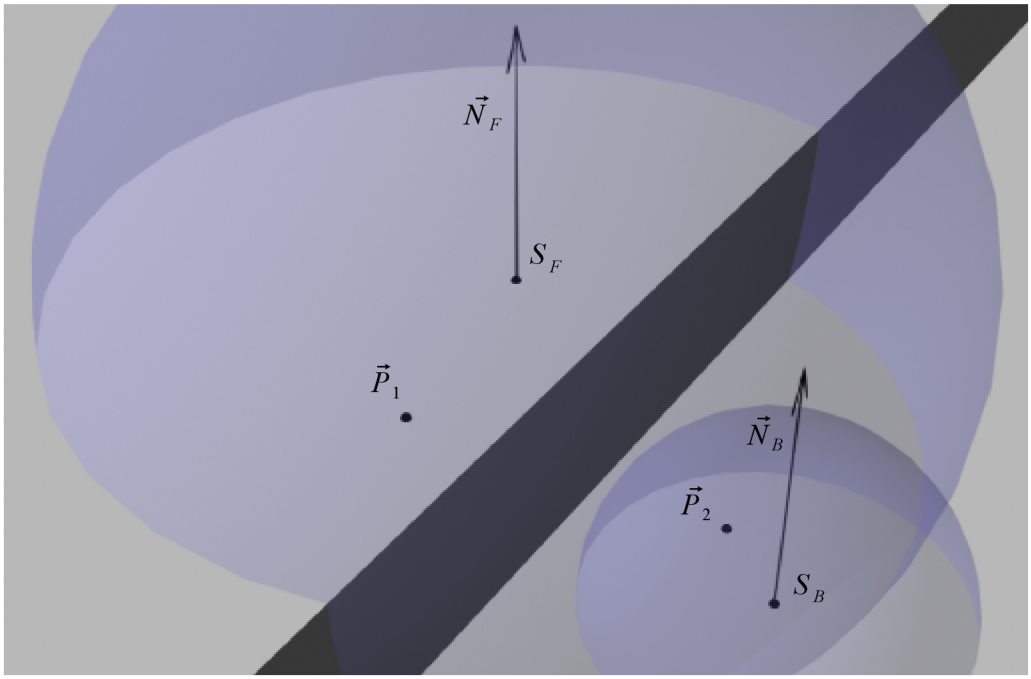
maximální možnou chybu mezi dvěma různě orientovanými ploškami v prostoru. Vzorky jsou váženy převrácenou hodnotou odhadu maximální chyby:

$$w_i(\vec{P}) = \frac{1}{\frac{|\vec{P} - \vec{P}_i|}{R_i} + \sqrt{1 - \vec{N} \cdot \vec{N}_i}}$$

Kde $\vec{P}$ je poloha testovaného bodu, $\vec{P}_i$ je poloha testovaného vzorku, $\vec{N}$ je normála v testovaném bodě, $\vec{N}_i$ je normála vzorku a R_i je harmonický průměr vzdáleností objektů viditelných ze vzorku i. Harmonický průměr R_i se spočítá jako:

$$R_i = \frac{n}{\sum_{i=1}^{n} \frac{1}{d_i}}$$

Kde d_i je vzdálenost průsečíku paprsku s objektem. Harmonický průměr také určuje poloměr platnosti daného vzorku. Vzorek bude mít na otevřeném prostranství širší poloměr platnosti než vzorek v rohu scény obklopený stěnami. Kromě vážení vzorků uvedeným vzorcem by se měli z výpočtu vyloučit vzorky ležící před testovaným bodem.



Ilustrace 4: Vzorek S_F by měl být ze zpracování vyloučen vzhledem k testovanému bodu $\vec{P}_2$. Vzorek S_B bude vzhledem k bodu $\vec{P}_1$ vyloučen automaticky, protože má jako zastíněný bod menší poloměr platnosti.

Provede se jednoduchý test:

$$s_i(\vec{P}) = (\vec{P} - \vec{P}_i) \cdot \frac{\vec{N} + \vec{N}_i}{2}$$

Hustota vzorků odpovídající kvalitě výsledného obrazu se určí uživatelsky zadanou konstantou a. Pro validní vzorek by tedy mělo platit:

$$(w_i(\vec{P}) > 1/a) \wedge (s_i(\vec{P}) > 0)$$

Vzorek ležící za testovaným bodem by měl být ze zpracování vyloučen automaticky, protože jako zastíněný bod by měl mít zároveň menší poloměr platnosti. Za předpokladu, že jsou z výpočtu vyloučeny všechny nevhodné vzorky se irradiance spočítá jako:

$$E(\vec{P}) = \frac{\sum_{i=1}^{n} w_i(\vec{p}) E_i}{\sum_{i=1}^{n} w_i(\vec{P})}$$

2.3.5.3 Struktura pro uložení vzorků irradiance

Podobně jako v algoritmu photon mapping jsou fotony ukládány do kd-stromu i zde je potřeba vzorky irradiance ukládat do struktury, která umožní rychlé vyhledávání nejbližších sousedů v prostoru. V základním algoritmu photon mapping je generování nových fotonů nezávislé na již uložených fotonech. Zde je však generování nového vzorku irradiance přímo závislé na nejbližších sousedních vzorcích. Tudíž je nutné použití takové struktury, která umožní jak rychlé vyhledání nejbližších sousedů, tak i dynamické ukládání nových vzorků. Budování nové struktury pro každý nový vzorek by bylo značně neefektivní.

Autoři v článku [10] navrhují použití oktalového stromu. Možné by bylo i použití velmi rozšířeného kd-stromu. Výběr dělící by se omezil na prostorový medián. Sofistikovanější postupy výběru dělící roviny by totiž vyžadovaly složité přeskládání stromu při vložení nového prvku. Jinou ne příliš paměťově úspornou možností je ukládat vzorky do obrazového bufferu. Tento přístup má však celou řadu nevýhod. Velká část obrazové matice nebude využita. Díky reflexním povrchům se může stát, že se budou zbytečně znovu počítat vzorky v místě prostoru, kde už jsou vypočítány, ale leží na zcela jiném místě obrazové matice. Vyhledávání nejbližších sousedů ve dvojrozměrné matici bude pravděpodobně pomalejší než vyhledávání v prostorové struktuře.

3 Návrh programu

Cílem této práce je implementace algoritmu photon mapping, zjištění jeho omezení, výhod a nevýhod. Systém byl původně navržen pouze pro výpočet světelných map, jejich uložení do souborů a následné zobrazení scény jednoduchou interaktivní aplikací. Navržena byla jedna aplikace, která provedla výpočet světelné mapy a následně zobrazila scénu s vypočítaným osvětlením. Tento přístup má výhodu v tom, že je možné sdílet geometrická data pro výpočet světelné mapy a pro zobrazení scény. Není tedy nutné geometrii načítat dvakrát ze souboru. Při kompilaci pro 64bitový systém nás omezuje použití 32bitové knihovny pro vizualizaci. Proto byl návrh pozměněn a aplikace rozdělena na program počítající světelnou mapu a program zobrazující výslednou scénu. Samotný renderer je možné zkompilovat pro 64bitový systém a program pro zobrazení ponechat 32bitový. Tímto rozdělením se zároveň zpřehlední kód a zbavíme se závislosti na grafické knihovně, pokud bychom chtěli světelnou mapu použít jiným způsobem.

Tento koncept byl do jisté míry ponechán, ale po rozšíření programu o reflexní povrchy bylo nutné pohledově závislou scénu realisticky vykreslit. Jedno řešení by bylo reflexní povrchy simulovat v grafickém akcelerátoru za pomocí shaderů nebo se omezit na vykreslení statických snímků. Vybrána byla druhá možnost, protože nabízí věrohodnější zobrazení modelované scény.

3.1 Reprezentace scény

Popis celé scény je uložen v jednom konfiguračním souboru obsahujícím kompletní popis scény včetně nastavení výstupu a výstupní kvality. Soubor by měl být snadno čitelný jak programem, tak i uživatelem abychom se vyhnuli použití dalšího nástroje na jeho tvorbu. Podobně formáty vstupních souborů s texturami a geometrií by měly být jednoduché a hodně rozšířené tak, aby bylo snadné případné jiné formáty zkonvertovat do požadovaného formátu. Zvolené formáty by pokud možno měli být nezávislé na konkrétním programu či platformě.

Konfigurační soubor tedy obsahuje parametry kvality výstupu. Výstupem může být světelná mapa nebo snímek. Nastavení kamery v textovém souboru není sice ideální, ale pro demonstraci výsledků je dostačující. Podobně rozmístění objektů scény je dáno jejich absolutními polohami. Interaktivní editor scény však není předmětem této práce. Po rozšíření programu o další nastavení povrchů bylo nutné kvůli přehlednosti a také úspoře paměti oddělit nastavení materiálů od objektů s geometrií.

4 Implementace

Kód programu je napsán v jazyce C++. Program je navržen objektově. Zdrojové kódy jsou komentovány stylem programu Doxygen, aby bylo možné vygenerovat programovou dokumentaci. Pro běh programů je nutný konfigurační soubor ze kterého se načtou informace o scéně.

Soubory geometrie jsou typu OBJ. Tento formát obsahuje pouze informaci o geometrii objektu a texturovací souřadnice. Může obsahovat ještě další informace, které jsou ignorovány. Program načítá pouze trojúhelníkové polygony. Pokud bude v souboru uložena ještě jiný typ geometrie může dojít k nepředvídatelným výsledkům.

Barevné textury můžou být pouze ve formátu 24bitového BMP obrázku. Program vytvoří pro každý soubor OBJ vlastní světelnou mapu podle texturovacích souřadnic v tomto souboru obsažených. Světelné mapy se uloží opět do souboru BMP.

4.1 Vlastní implementace

Detailní popis kódu programu se nachází v programové dokumentaci. Zde bude uvedeno pouze schéma aplikace a použité algoritmy.

4.1.1 Datové typy

Základním datovým typem pro reprezentaci dvourozměrného nebo třírozměrného vektoru je struktura obsahující příslušný počet datových položek typu double. Tento typ je využíván pro reprezentaci RGB barvy, vektoru v prostoru, bodu v prostoru nebo jen čísel bez udané jednotky. Tento jednotný koncept je použit kvůli častým operacím mezi těmito různými typy. S jednotnou reprezentací vektoru nejsou nutné žádné explicitní přetypování a konverze. K datovým prvkům je možné přistupovat podle významu buď jako xyz nebo rgb nebo pomocí operátoru [].

Základní geometrický objekt je zde trojúhelník. Trojúhelník se kromě metod pro nalezení průsečíku s paprskem, rasterizaci do světelné mapy a dalších metod skládá ze třech vrcholů. Každý vrchol obsahuje normálu, texturovací souřadnice a pozici.

Třídou na nejvyšší úrovni hierarchie je třída zapouzdřující celou scénu. Scéna obsahuje seznam trojúhelníků, světel, materiálů, světelných map, kameru, globální parametry a další pomocné struktury jako například kd-strom pro uložení fotonů a kd-strom pro urychlení výpočtu průsečíku s paprskem. Dále scéna obsahuje metody pro načtení konfiguračního souboru a provedení vlastního výpočtu.

4.1.2 Implementované algoritmy

Pro uložení fotonů je použita modifikovaná verze kd-stromu podle [1]. Průsečík paprsku s nejbližším objektem se počítá s využitím kd-stromu. kd-strom se vytvoří podle algoritmu popsaného v [9]. Dělící rovina se určí podle heuristiky SAH. Průchod stromem je popsán v teoretické části.

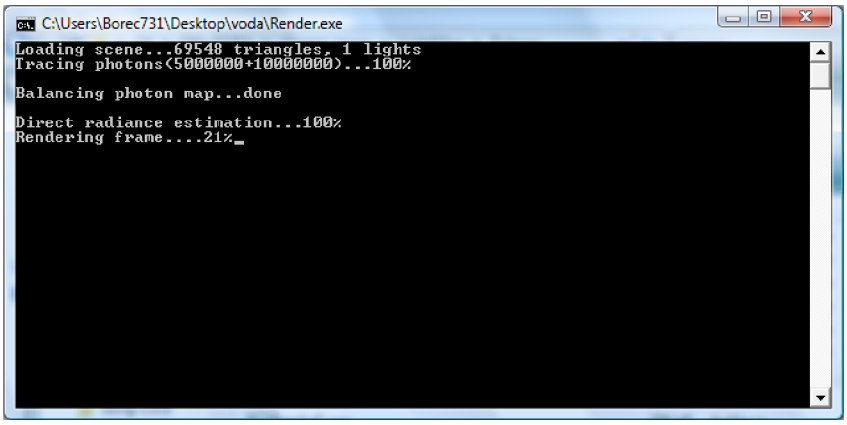
Po načtení scény se vytvoří kd-strom pro efektivní sledování paprsku. Následuje fáze sledování fotonů. Fotony se vysílají tak dlouho, dokud není naplněna fotonová mapa. Každý foton musí nést ještě informaci o tom, kterému světlu patří, aby mohla být upravena jeho energie. Po vytvoření mapy fotonů následuje vykreslení scény.

Pokud je nastaveno použití odhadu radiance přímo z mapy fotonů, vykreslí se scéna sledováním paprsku nebo rasterizací osvětlení do světelné mapy. Pokud je však požadován výstup v lepší kvalitě s využitím sbírání radiance, je osvětlení nejprve uloženo do světelných map. Stačí použít menší rozlišení i počet fotonů. Potom teprve následuje fáze sledování paprsku. Povrchy můžou být zcela reflexní, zcela průhledné nebo difuzní. V této fázi už není potřeba mapa difuzně odražených fotonů. Nepřímé difuzní osvětlení se počítá Monte Carlo vzorkováním. Příspěvky radiance se vezmou z uložené světelné mapy. K nepřímému difuznímu osvětlení se přičte osvětlení přímé a kaustiky. Kaustiky se musí odhadnout přímo z kaustické mapy fotonů.

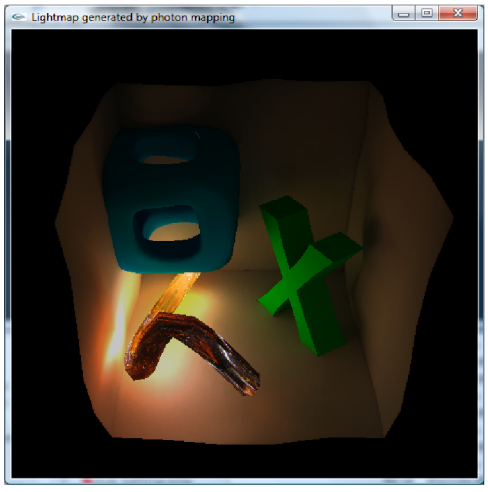
Výpočet výsledného snímku probíhá hierarchicky, aby bylo v algoritmu irradiance caching dosaženo spíše interpolace než extrapolace. Výpočet pixelů obrazové matice probíhá paralelně s vyžitím OpenMP API (Open Multi-Processing).

4.2 Rozhraní aplikací

Programy musí mít k dispozici validní konfigurační soubor a samozřejmě i všechny ostatní soubory s geometrií a texturami. Aplikace pro rendering zobrazuje procentuální stav procentuální stav výpočtu jednotlivých fází. Aplikace pro zobrazení světelných map se ovládá myší. Pravé tlačítko slouží k přibližování a levé k otáčení scény.



Ilustrace 5: Ukázka výstupu aplikace při výpočtu.



Ilustrace 6: Zobrazení světelné mapy v OpenGL aplikaci.

4.3 Konfigurační soubor

Soubor XML popisující celou scénu včetně parametrů photon mappingu a rozlišení světelných map.

Příklad konfiguračního souboru:

```
<scene>
<global_params>
        <gather>1</gather>
        <render_frame_only>1</render_frame_only>
        <max_cache_error>0.01</max_cache_error>
        <gather_eps>0.0005</gather_eps>
        <max_depth>20</max_depth>
        <photonmap_size>5000000</photonmap_size>
        <max_dist>0.5</max_dist>
        <num_nearest>5000</num_nearest>
        <caustic_photonmap_size>0</caustic_photonmap_size>
        <caustic_max_dist>0.08</caustic_max_dist>
        <caustic_num_nearest>5000</caustic_num_nearest>
</global_params>
<camera>
        <eye position="-4.0 0.0 7.0" target="-1.0 -3.0 0.0" sky="0.0 1.0 0.0" fov="90.0" ratio="4/3"/>
        <frame width="800" height="600" samples="16"/>
</camera>
<materials>
        <material name="grey" texture="grey.bmp" diff_fac="0.8" smooth_angle="60.0"/>
        <material name="green" texture="green.bmp" diff_fac="0.8" smooth_angle="60.0"/>
        <material name="sky" texture="sky1.bmp" diff_fac="0.8" smooth_angle="60.0"/>
</materials>
<objects>
        <object file="cross.obj" lightmap_res="256" material="green"/>
        <object file="dragon.obj" lightmap_res="256" material="grey"/>
        <object file="ground.obj" lightmap_res="256" material="grey"/>
</objects>
<lights>
        <complex_light file="sky.obj" lightmap_res="256" power="1500 1500 1500" sample="1"
        material="sky"/>
</lights>
</scene>
```

5 Výsledky

K demonstraci výsledků byla použita uzavřená scéna ve tvaru krychle. Do scény je vložen model draka pocházející ze Stanford University Computer Graphics Laboratory. Velikost původního modelu byla redukována na 108 691 trojúhelníků. Do scény je vložen trojrozměrný model písmene X zelené barvy, aby lépe vynikl efekt mísení barev (color bleeding). Veškeré zde uvedené ukázky výstupu jsou k dispozici na přiloženém CD.

Procesor	AMD Athlon X2 4850e
Velikost operační paměti	4 GB
Operační systém	Windows Vista (64 bitů)

Tabulka 1: Testovací konfigurace.

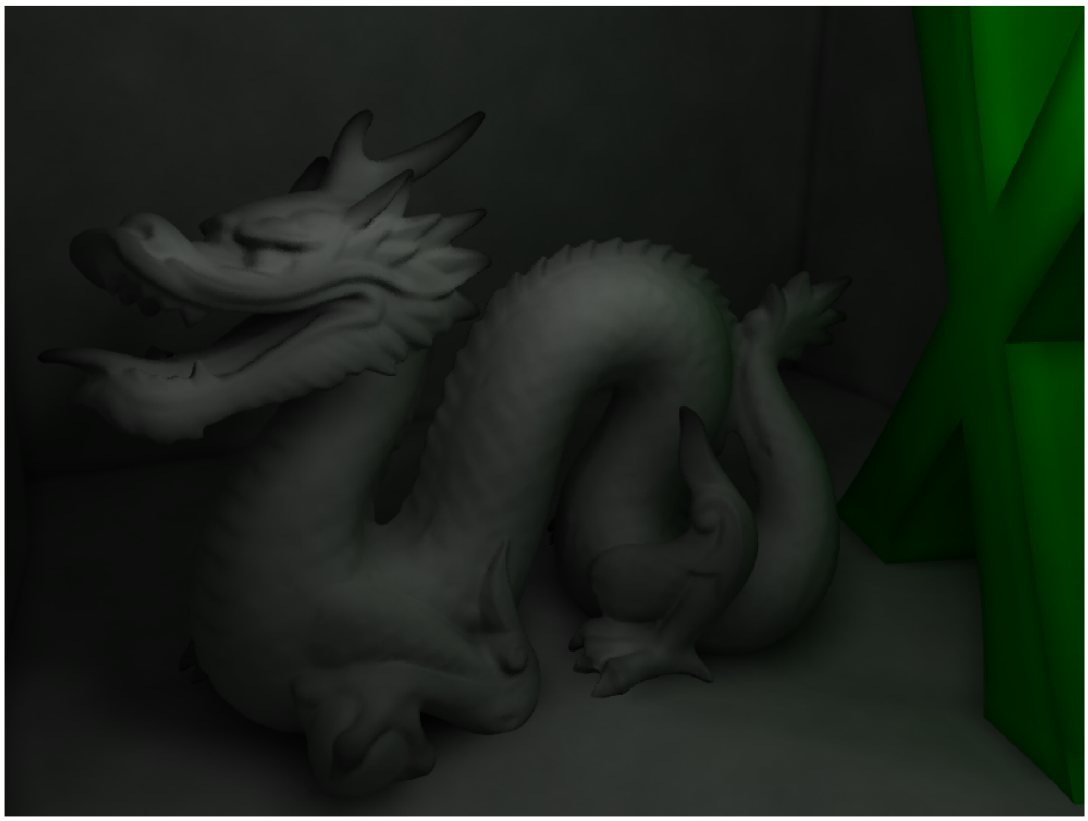
Počet difuzně odražených fotonů	5 000 000
Maximální poloměr vyhledávání	0,3
Počet nejbližších fotonů	5 000
Počet kaustických fotonů	10 000 000
Maximální poloměr vyhledávání	0,08
Počet nejbližších fotonů	5 000
Maximální chyba vzorků irradiance (interiér)	0,1
Maximální chyba vzorků irradiance (exteriér)	0,01
Maximální chyba vzorkování polokoule	0,0005
Maximální hloubka zanoření paprsku	20

Tabulka 2: Základní parametry testované scény.

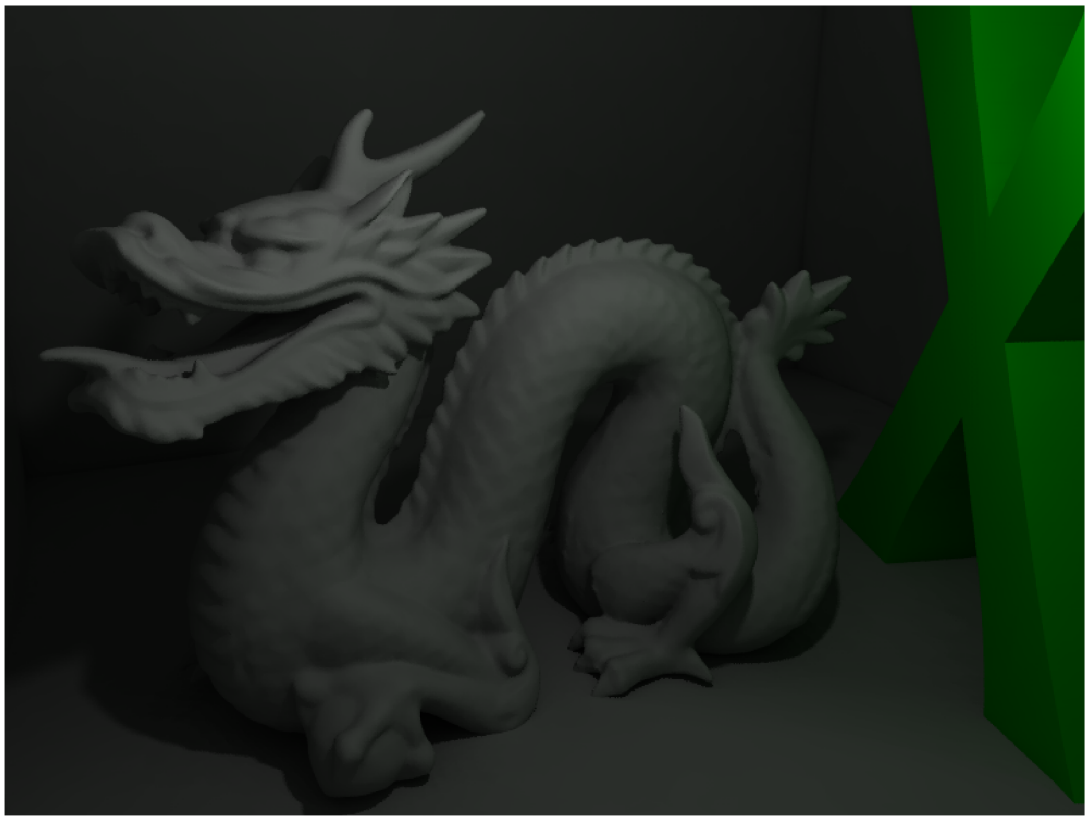
Scéna č.	1	2	3	4	5	6	7	8	9	10
Počet trojúhelníků	109 245	109 245	109 245	110 037	109 245	217 934	109 155	109 155	109 155	4 908
Čas výpočtu	00:02:27	00:33:42	00:22:02	01:43:20	01:17:34	02:05:12	04:37:57	04:59:35	05:19:48	00:27:27

Tabulka 3: Časy výpočtu scén zobrazených níže.

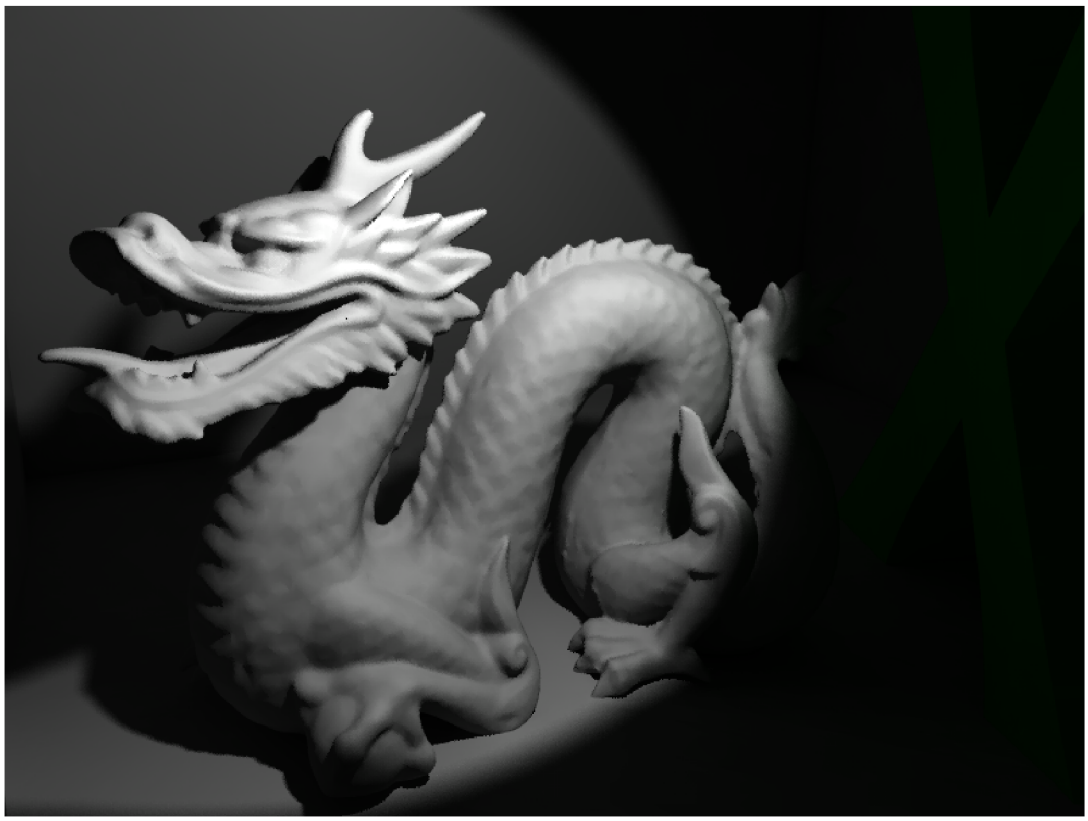
Ukázky scén byly vyrenderovány v rozlišení 800*600 pixelů. Bylo použito 16 náhodných vzorků na jeden pixel. Pro srovnání parametrů v tabulkách byl při výpočtu antialiasing vypnutý.



Scéna 1: Přímý odhad osvětlení ze světelné mapy. Scéna je osvětlena jedním bodovým světlem.

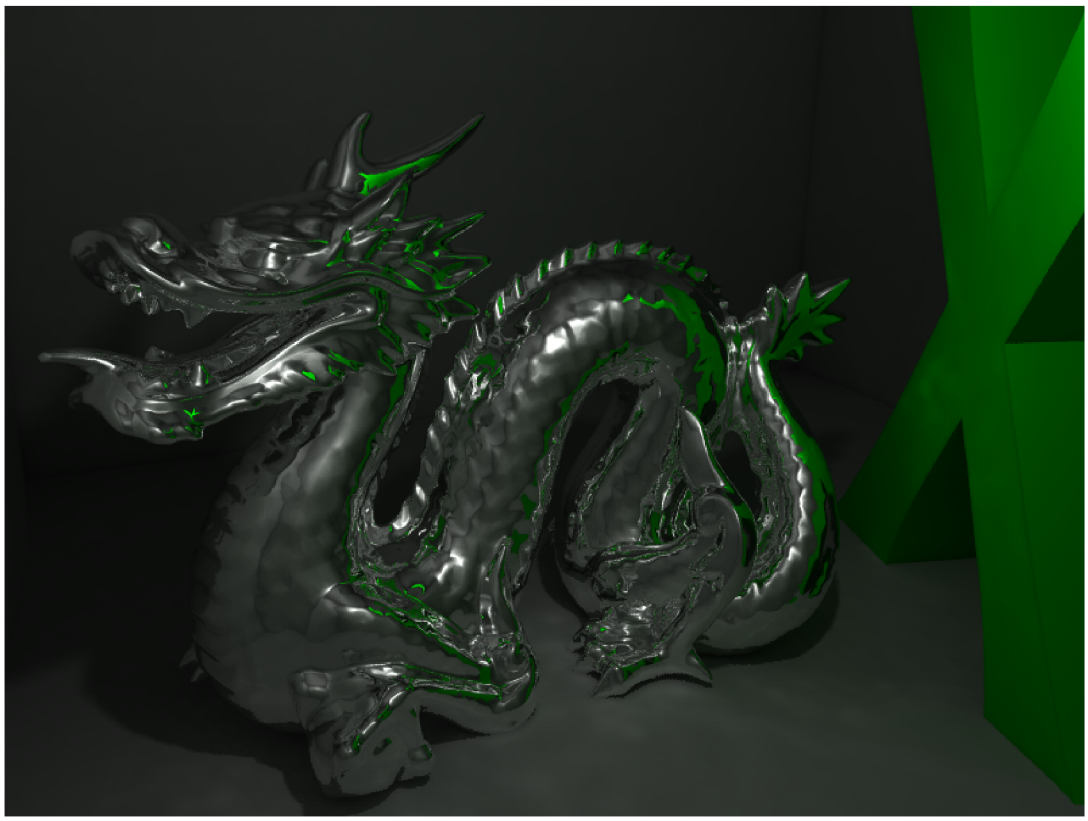


Scéna 2: Stejná scéna jako výše, ale s využitím algoritmu irradiance caching.

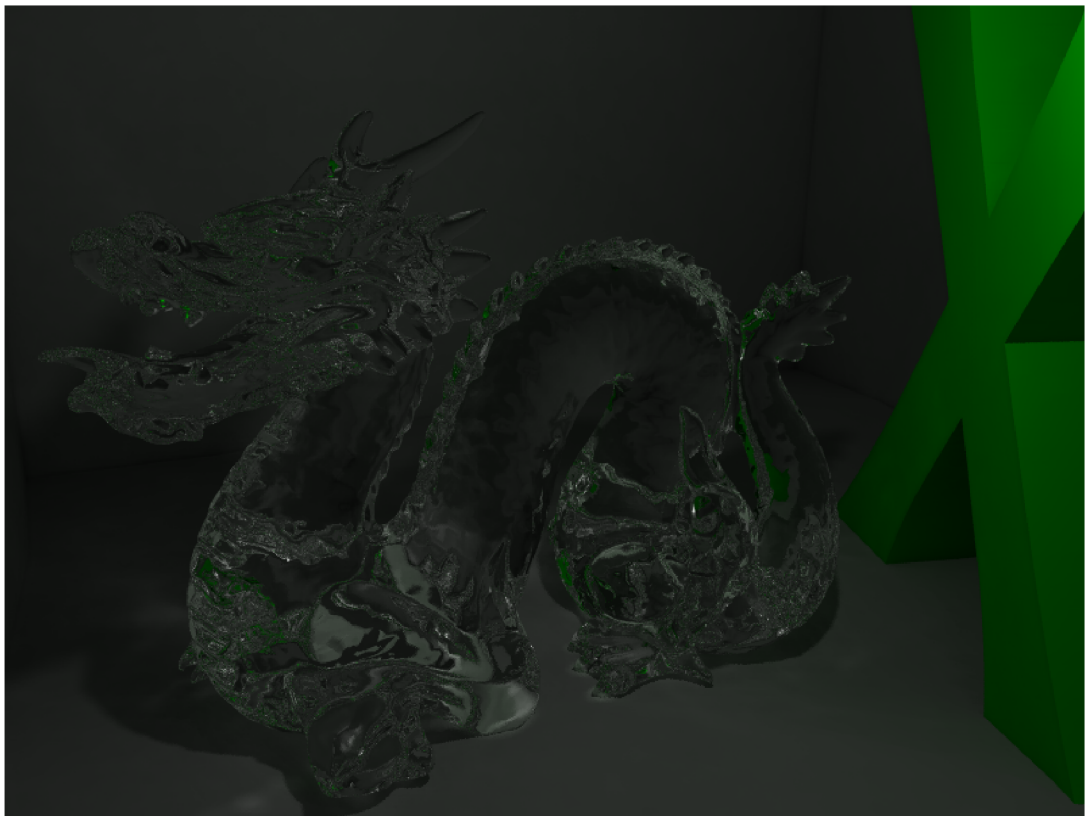


Scéna 3: Kuželové světlo.

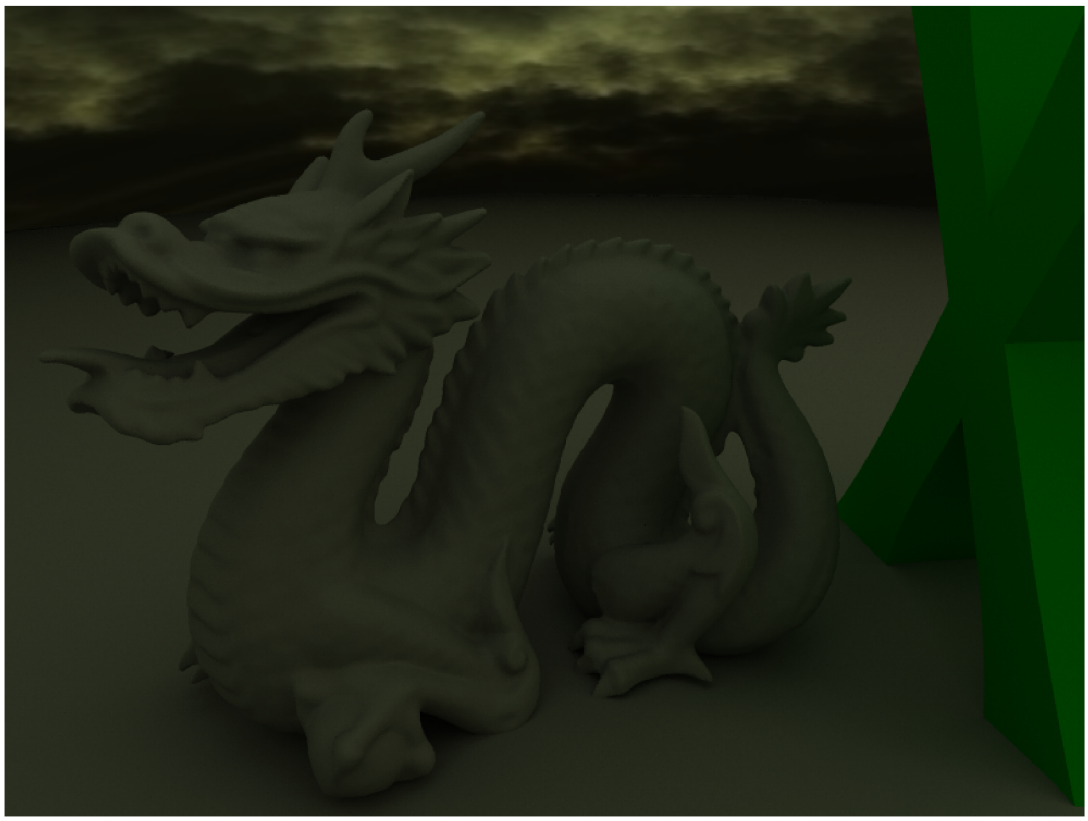
Scéna 4: Difuzní světlo vzorkované samostatně.



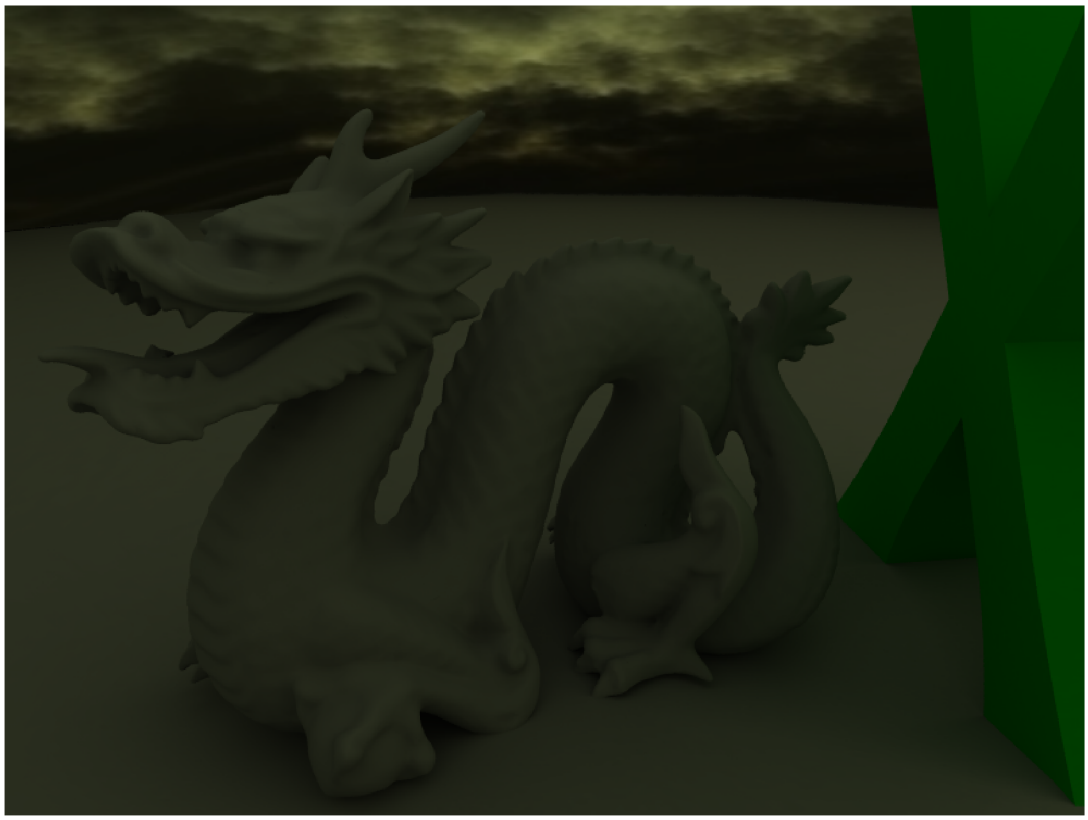
Scéna 5: Materiál draka je zrcadlově odrazivý.



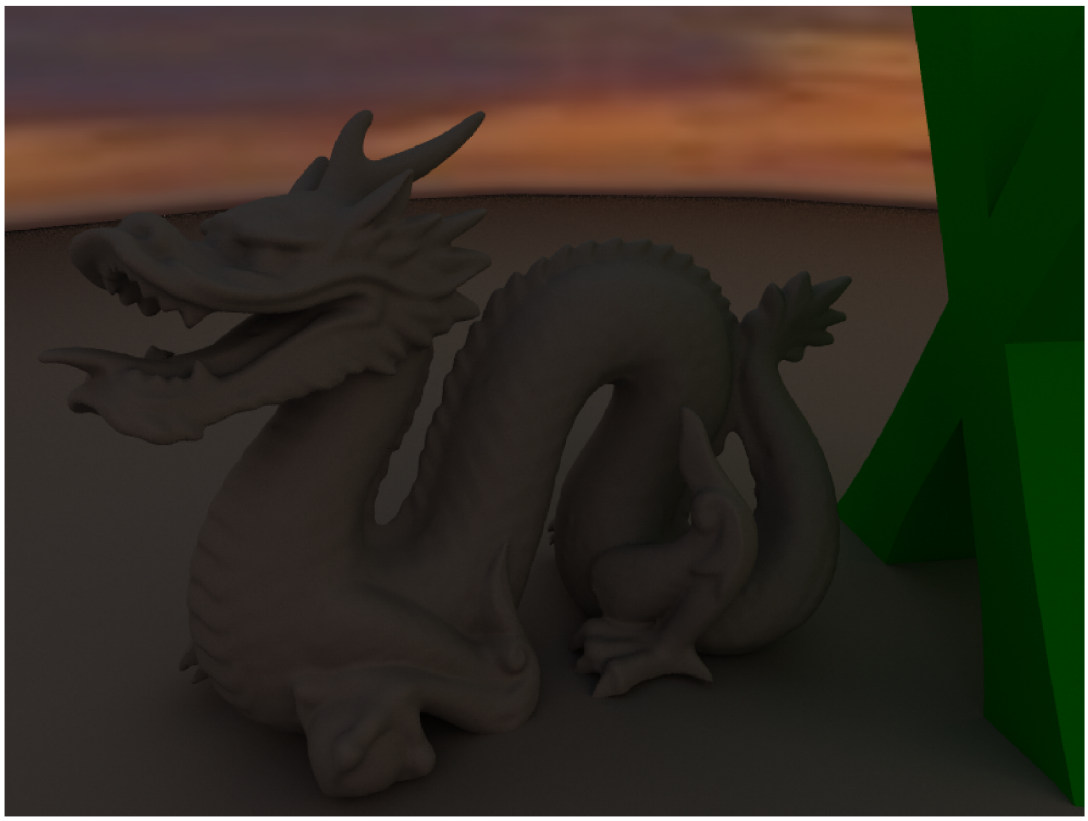
Scéna 6: Materiál draka je zcela průhledný. Index lomu je nastaven na 1,6.



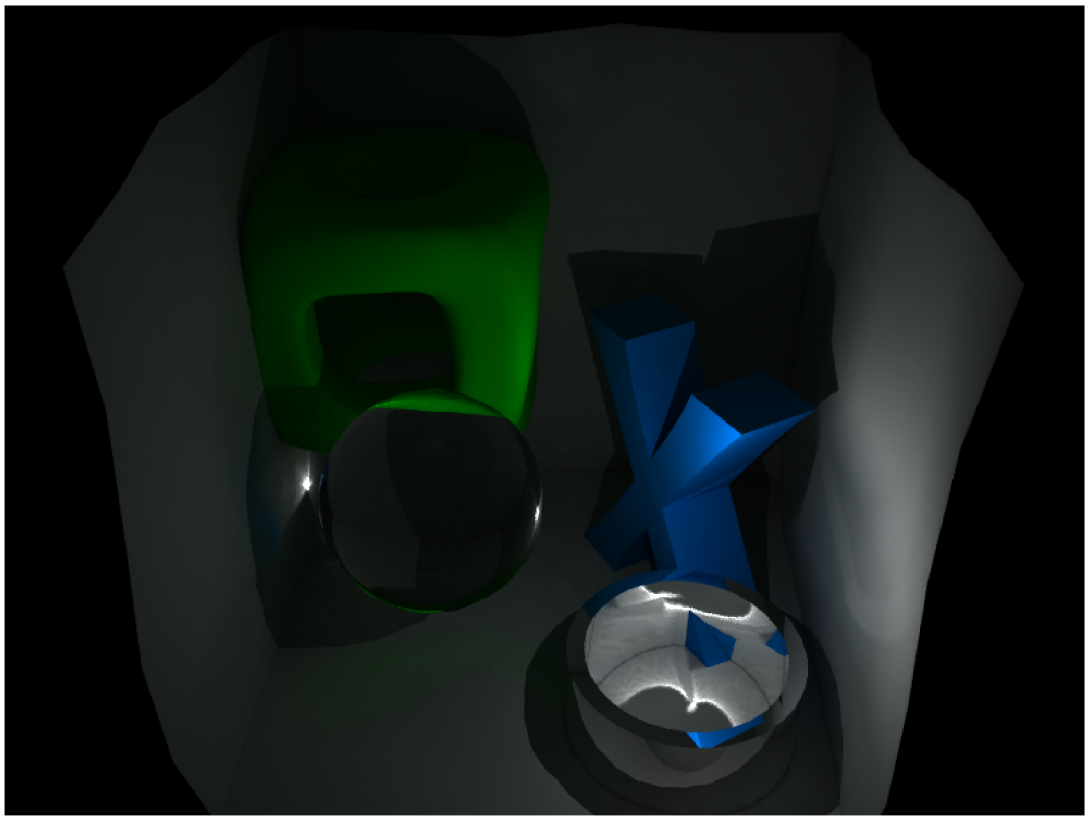
Scéna 7: Venkovní scéna osvětlená difuzní polokoulí s texturou oblohy. Světlo je vzorkováno samostatně.



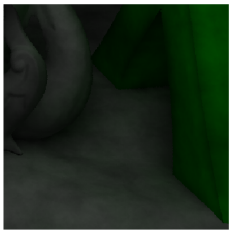
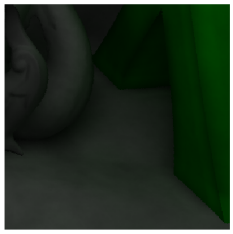
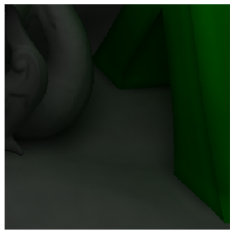
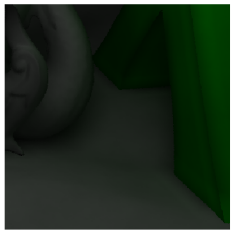
Scéna 8: Stejná scéna jako výše jen světlo je vzorkováno jako difuzní povrch.



Scéna 9: Stejná scéna jako výše pouze s jinou texturou oblohy. Světlo je vzorkováno samostatně.



Scéna 10: Ukázka kaustických efektů, počet nejbližších fotonů byl nastaven na 1 000.

Počet fotonů	500 000	2 000 000	5 000 000	10 000 000
Čas výpočtu	00:00:21	00:00:52	00:02:03	00:05:05
Výstup 2				

Tabulka 4: Vliv počtu fotonů na výstup.

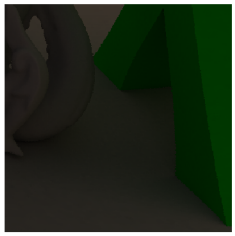
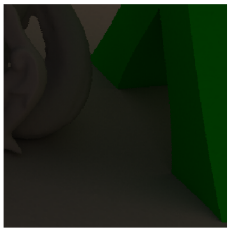
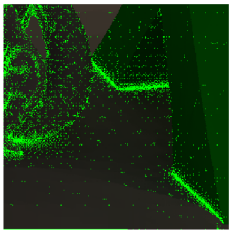
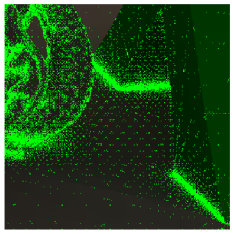
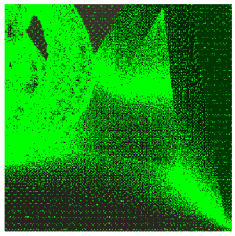
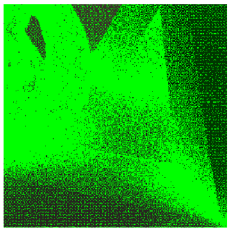
Maximální poloměr hledání	1,0	0,5	0,3	0,1
Čas výpočtu	00:07:32	00:04:52	00:02:05	00:01:24
Výstup				

Tabulka 5: Vliv maximálního prohledávaného poloměru na výstup.

Počet nejbližších fotonů	50	200	1 000	5 000
Čas výpočtu	00:01:21	00:01:27	00:01:53	00:02:04
Výstup				

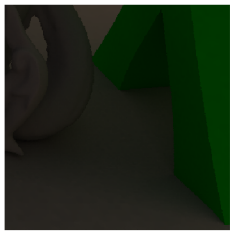
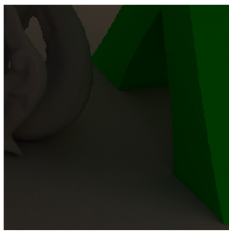
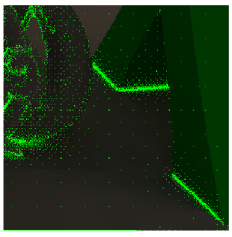
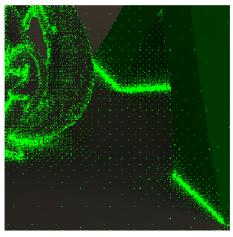
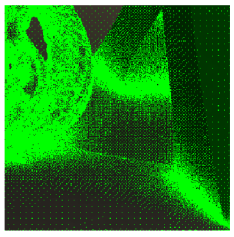
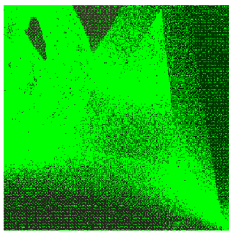
Tabulka 6: Vliv počtu nejbližších fotonů na výstup.

Základní parametry algoritmu photon mapping ovlivňují hlavně míru výskytu šumu ve výsledném obrázku. Maximální poloměr hledání určuje detailnost osvětlení podobně jako počet nejbližších fotonů. Větší poloměr hledání (více nejbližších fotonů) znamená potlačení šumu, ale zhoršení odhadu na hranách, rozích a ostrých výstupcích. Nastavením parametrů se vždy snažíme dosáhnout hladkého osvětlení s maximálními detaily. Toto platí jak pro difuzně odražené fotony tak pro kaustiky. Kaustická mapa fotonů bývá v zásadě detailnější, skládá se z více fotonů a pro vizualizace je třeba zvolit menší poloměr hledání než u globální mapy fotonů. Tyto parametry velkou mírou závisí na vykreslované scéně a je vhodné je upravit experimentálně.

Maximální odchylka vzorků irradiance	0,1	0,05	0,01	0,005
Čas výpočtu	00:03:59	00:06:26	00:14:58	00:18:27
Výstup				
Vizualizace vzorků				

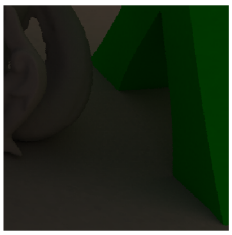
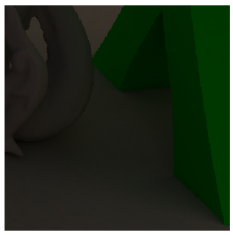
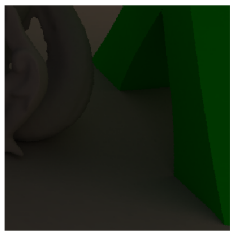
Tabulka 7: Vliv maximální chyby vzorků irradiance na výstup. Z výpočtu jsou vyloučeny vzorky ležící v prostoru před vzorkovaným bodem.

Maximální odchylka vzorků irradiance závisí velkou mírou na intenzitě světla ve scéně. Vzdálený difuzní povrch s velkou energií může být výhodnější vzorkovat jako plošné světlo, protože nebude nutné vysílat extrémní množství paprsků pro vzorkování difuzní polokoule. Běžný počet vyslaných paprsků jsou stovky až tisíce. Vyslání více paprsků se stává neefektivní.

Maximální odchylka vzorků irradiance	0,1	0,05	0,01	0,005
Čas výpočtu	00:03:10	00:05:05	00:13:15	00:16:46
Výstup				
Vizualizace vzorků				

Tabulka 8: Vliv maximální chyby vzorků irradiance na výstup. Do výpočtu jsou zahrnuty i vzorky ležící v prostoru před vzorkovaným bodem.

Zanedbání testu zdali zkoumaný vzorek leží před vzorkovaným bodem způsobuje pravidelnější distribuci vzorků po povrchu. Časy výpočtu jsou o něco menší než při kompletním testu, výsledný obraz je však téměř stejné kvality jako při provedení kompletního testu.

Maximální chyba vzorkování polokoule	0,1	0,005	0,0005	0,0001
Čas výpočtu	00:11:18	00:14:59	00:43:53	02:01:34
Výstup				

Tabulka 9: Vzorkování polokoule různým počtem paprsků.

Chyba vzorkování polokoule je počítána jako

$$E_R = |R_O - R_C| + |G_O - G_C| + |B_O - B_C|$$

Kde RGB hodnoty vzorků irradiance pro jednotlivé barevné kanály a index O značí předchozí hodnotu irradiance a index C značí aktuální hodnotu. Polokoule se vzorkuje 200 paprsky v jedné iteraci tak dlouho, dokud není chyba menší než zadaná mez. Už při základních 400 vzorcích algoritmus produkuje slušné výsledky v poměrně krátkém čase. Při zvýšení meze se výsledek zlepší jen málo, ale čas výpočtu se prodlouží výrazně.

Scéna č.	4 (plošné světlo)	4 (difuzní povrch)	7 (plošné světlo)	7 (difuzní povrch)
Čas výpočtu	01:43:20	00:29:40	04:37:57	04:59:35
Výstup				

Tabulka 10: Vzorkování difuzních světel jako plošná světla nebo difuzní povrchy.

Vzorkování plošného světla ve scéně č.4 jako difuzního povrchu se ukázalo jako nevhodné, protože světlo má příliš vysokou energii a malou plochu. Čas výpočtu je sice krátký, ale výstup není použitelný. Naopak ve scéně č.7 jsou kvalita výstupu i čas výpočtu podobné. Světlo vzorkované těmito různými metodami má trochu odlišný charakter důsledkem rozmístění vzorků. Povrch vzorkovaný jako plošné světlo má lépe vykreslené detaily. Vzorkování plošného světla generováním rovnoměrně distribuovaných bodů po povrchu má zase nevýhodu, když je vzorkovaný bod příliš blízko objektu světla, tak se odhad provede jen z několika málo viditelných bodů a dochází k velkým chybám odhadu.

6 Závěr

Photon mapping je velice univerzální algoritmus. Lze s ním simulovat celou řadu optických efektů. Na rozdíl od algoritmu bidirectional path tracing negeneruje tento algoritmus vysokofrekvenční šum, ale má zase jiné nevýhody. Při integraci energie fotonů na ploše dochází k velkým nepřesnostem, protože povrch samotné plochy je odhadnut pouze přibližně. Přímý odhad radiance z mapy fotonů na difuzním povrchu je vhodný pouze pro rychlý náhled scény s hrubým odhadem osvětlení. Výjimkou je simulace kaustik, kdy je přímý odhad radiance z fotonové mapy nutný. Potom se nabízí pouze zvýšení počtu fotonů a tím zmenšení poloměru vyhledávání nejbližších sousedů nebo dodatečnou filtrací.

Pro výstup ve vysoké kvalitě je nutné algoritmus photon mapping kombinovat s technikou final gathering nebo nějakou její modifikací. Přímé osvětlení je možné počítat přímo, nepřímé je počítáno vzorkováním polokoule mnoha paprsky a tím je výsledná chyba značně redukována. Protože se nepřímé osvětlení mění pouze pozvolna využívá se interpolace řídce rozmístěných vzorků. Pokud je však zdroj světla příliš silný dochází při interpolaci k velkým odchylkám a je nutné buď zvýšit hustotu vzorků nebo využít gradientu osvětlení k lepšímu odhadu. Jinou možností je tento difuzní povrch s vysokou energií vzorkovat jako plošné difuzní světlo. Potom je nutné rozhodnout jestli se bude povrch vzorkovat jako plošné světlo nebo jen další difuzní povrch. Rozhodnutí tohoto problému automaticky nemusí být však jednoduché.

Možností rozšíření stávající práce se nabízí mnoho. Vykreslení probíhá vcelku rychle, avšak paměťovým optimalizacím nebyla věnovány velká pozornost. Větší pozornost by mohla být věnována také problému paralelizace. V současném stavu je paralelní pouze výpočet výsledného snímku a nikoli samotné vysílání fotonů. Odhad kaustik prováděný základním algoritmem by se mohl doplnit o dodatečnou filtraci. Materiály by se mohly rozšířit o další parametry a efekty. Přidána by mohla být simulace rozptylu světla pod povrchem nebo volumetrických materiálů. Také interpolace osvětlení algoritmem irradiance caching by se mohla vylepšit využitím gradientů.

Literatura

[1] Henrik Wann Jensen: *Realistic image synthesis using photon mapping*, A K Peters, 2001

[2] Henrik Wann Jensen: *A practical guide to global illumination using photon mapping*, SIGGRAPH, 2002

[3] Henrik Wann Jensen: *Global illumination using photon maps*, Eurographics Workshop on Rendering, 1996

[4] Arthur Appel: *On calculating the illusion of reality*, IPIF Congress, 1968

[5] Turner Whitted: *An improved illumination model for shaded display*, SIGGRAPH, 1980

[6] Jeffrey Goldsmith, John Salmon: *Automatic creation of object hierarchies for ray tracing*, IEEE Computer Society Press, 1987

[7] Vlastimil Havran: *Heuristic ray shooting algorithms*, diplomová práce, ČVUT Praha, 2001

[8] Carsten Wächter, Alexander Keller: *Instant ray tracing: The bounding interval hierarchy*, Eurographics Symposium on Rendering, 2006

[9] Ingo Wald, Vlastimil Havran: *On building fast kd-trees for ray tracing, and doing that in* $O(N \log N)$, IEEE Symposium on Interactive Ray Tracing, 2006

[10] Gregory J. Ward, Francis M. Rubinstein, Robert D. Clear: *A ray tracing solution for diffuse interreflection*, SIGGRAPH, 1988

[11] Gregory J. Ward, Paul S. Heckbert: *Irradiance gradients*, Eurographics Workshop on Rendering, 1992

[12] Jaroslav Křivánek, Pascal Gautron, Sumanta Pattanaik, Kadi Bouatouch: *Radiance caching for efficient global illumination computation*, IEEE TVCG, 2005

[13] Jaroslav Křivánek, Kadi Bouatouch, Sumanta Pattanaik, Jiří Žára: *Making radiance and irradiance caching practical: Adaptive caching and neighbor clamping*, Eurographics Symposium on Rendering, 2006

[14] Jaroslav Křivánek, Pascal Gautron, Kadi Bouatouch, Sumanta Pattanaik: *Improved radiance gradient computation*, SIGGRAPH, 2005

[15] James T. Kajiya: *The rendering equation*, SIGGRAPH, 1986

[16] Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, Pat Hanrahan: *A practical model for subsurface light transport*, SIGGRAPH, 2001