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

ZOBRAZENÍ SCÉNY POMOCÍ HLUBOKÝCH STÍNOVÝCH MAP

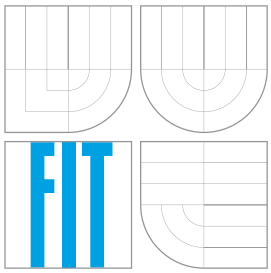
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ REJENT

BRNO 2014

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

ZOBRAZENÍ SCÉNY POMOCÍ HLUBOKÝCH STÍNOVÝCH MAP

RENDERING USING DEEP SHADOWMAPS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ REJENT

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LUKÁŠ POLOK

BRNO 2014

Abstrakt

Vykreslování stínů průhledných objektů je problematické, obzvláště pokud je použito v interaktivních aplikacích. Omezený výpočetní výkon limituje použitelné metody. Tato práce popisuje metodu Depth peeling a její variantu Dual depth peeling pro vykreslování průhledných objektů bez nutnosti jejich seřazení. Následuje popis metody Deep shadow maps pro tvorbu stínů u průhledných objektů. Tyto metody byly použity pro vytvoření demonstrační aplikace, která umožňuje zobrazení stínů průhledných objektů, včetně barevných stínů. Aplikace je vytvořena pomocí OpenGL a Qt frameworku. Součástí práce je také vyhodnocení vlivu různých parametrů na rychlost vykreslování.

Abstract

Rendering shadows of transparent objects in real-time applications is difficult. The number of usable methods is limited by the available computing power. Depth Peeling and Dual Depth Peeling methods are described in this document. These allow rendering of transparent objects without the need of sorting them. Deep Shadow Maps are described as a method for rendering shadows of transparent objects. These methods were used to create an demonstration application. This application provides rendering of transparent objects and their shadows, including colored ones. The Application is build upon OpenGL and Qt framework. Evaluation of rendering speed according to various parameters is also part of this work.

Klíčová slova

stíny průhledných těles, potlačení aliasingu, hloubkové stínové mapy, depth peeling, vykreslování v reálném čase, OpenGL

Keywords

shadowing of transparent objects, aliasing suppression, deep shadow maps, depth peeling, real-time rendering, OpenGL

Citace

Tomáš Rejent: Zobrazení scény pomocí hlubokých
stínových map, diplomová práce, Brno, FIT VUT v Brně, 2014

Zobrazení scény pomocí hlubokých stínových map

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Lukáše Poloka.

.....
Tomáš Rejent
28. května 2014

Obsah

Kapitola 1

Úvod

Zobrazení 3D scény je možné provést několika metodami. První metodou zobrazení je raytracing [1]. Tato metoda se inspiruje aproximací šíření světla pomocí paprsků. Simulace šíření paprsků od zdroje světla k pozorovateli je však značně výpočetně náročná, protože předem není známo, které paprsky k pozorovateli dorazí. Využívá se proto vrhání paprsků směrem od pozorovatele. Scéna je vykreslována po jednotlivých pixelech obrazu (image order). Skrze každý pixel je do scény promítnut paprsek. Pokud je nalezen průsečík P s nějakým objektem, jsou z tohoto průsečíku vyslány stínové paprsky ke každému zdroji světla ve scéně. Pokud stínový paprsek prochází nějakým objektem, bod P se nachází ve stínu vůči tomuto zdroji světla, v opačném případě je tímto zdrojem světla osvětlen. Výsledná barva bodu P je určena na základě příspěvků nezastíněných zdrojů světla. Tento postup je možné opakovat rekurzivně pomocí vyslání sekundárního paprsku z bodu P podle zákonu odrazu. Z principu metody je zřejmé, že umožňuje bez dalších úprav zobrazení ostrých stínů a odrazů světla od objektů.

Nevýhodou raytracingu je výpočetní náročnost. Pokusy o přímou hardwarovou podporu této metody [2] se nikdy nedočkaly širšího rozšíření. Proto se raytracing využívá především pro offline vykreslování. Mezi výhody patří kromě vyšší kvality zobrazení také možnost zobrazení objemových a CSG dat, které se při vykreslování rasterizací nejdříve musí převést na povrchovou reprezentaci.

Další metodou je rasterizace, která zobrazuje scénu průchodem přes objekty scény (object order). Objekty jsou promítnuty do roviny obrazovky a rasterizovány. Pokud je více objektů rasterizováno do stejných pixelů, je nutné určit, který objekt je nejblíže k pozorovateli. Tento problém se řeší pomocí Z-bufferu, který uchovává hloubku pro každý pixel. Rasterizace využívá sousednost pixelů vykreslovaných objektů. Narozdíl od interpolace souřadnic pro každý pixel při zobrazení pomocí raytracingu stačí v rasterizaci zleva doprava přičítat konstantu dx. Současný grafický hardware je navržený pro zobrazování touto metodu, proto se jedná o nejčastěji používanou metodu pro interaktivní zobrazení.

Při vykreslování scény metodou rasterizace nenastává mezi objekty scény žádná interakce, ve výsledném obrazu proto chybí například stíny a odrazy světla od objektů. Základními metodami pro vytvoření stínů ve scéně zobrazené pomocí rasterizace jsou Stínová tělesa (Shadow Volumes) a Stínové mapy (Shadow Maps). Obě metody vyžadují více než jeden vykreslovací průchod. Metoda stínových těles pro svoji funkci využívá stencil buffer, ve kterém označí zastíněné části scény pomocí vykreslení vytvořeného stínového tělesa. Metoda produkuje stíny na pixel přesně a funguje pro směrové i bodové zdroje světla, výsledné stíny jsou však nepřirozeně ostré. Existují různá rozšíření této metody [3] pro plošné zdroje světla a měkké stíny.

Metoda stínových map využívá texturu pro uložení hloubky scény z pohledu světla. Jedná se o jednoduchou a rychlou metodu, na rozdíl od stínových těles se nevytváří žádná nová geometrie. Nevýhodou je konečné rozlišení použité textury. Při přiblížení scény vznikají z důvodu vzorkování zoubkované okraje stínu. Tento problém se snaží řešit různé modifikace, například PSM (Perspective Shadow Maps) [4] a PCF (Percentage Closer Filtering) [5]. Další nevýhodou plynoucí z použití textury je rovnoměrné rozložení vzorků. Pro stíny blíže k pozorovateli je však žádoucí vyšší rozlišení, než pro stíny vzdálené. Modifikace, jako CSM (Cascaded Shadow Maps) [6] nebo LogPSM (Logarithmic Perspective Shadow Maps) [7], představují možná řešení tohoto problému. Stínové mapy fungují pro směrové zdroje světla. V případě bodového zdroje světla je nutná konstrukce šesti textur pro pokrytí všech směrů, případně parabolická projekce [8] pro konstrukci dvou textur, která ovšem přináší další problémy. Stínové mapy pracují s průhlednými objekty stejně jako s neprůhlednými a výsledné stíny jsou tak pro oba typy stejné, což neodpovídá realitě. Cílem této práce je úprava metody stínových map pro vrhání polostínů průhlednými objekty.

Následující kapitola shrnuje teoretické podklady relevantní pro tuto práci. Podkapitola 2.1 přibližuje dvě základní nejčastěji používané metody pro tvorbu ostrých stínů, stínová tělesa a stínové mapy. Další podkapitola 2.2 se zabývá osvětlovacími modely s důrazem na globální osvětlovací modely využitelné v interaktivních aplikacích. Podkapitola 2.3 vysvětluje princip metody Depth Peeling používané pro zobrazování průhledných těles. Kapitola 2.4 popisuje Stochastickou průhlednost, alternativní metodu k Depth Peeling metodě. Poslední podkapitola 2.5 popisuje Hluboké stínové mapy, klíčovou metodu pro vytváření polostínů u průhledných těles. Kapitola 3 stručně popisuje návrh aplikace a propojení metod popsaných v teoretické části. Kapitola 4 popisuje strukturu programu, implementační detaily použitých metod a základní ovládání aplikace. Kapitola 5 popisuje rychlost jednotlivých metod v závislosti na různých parametrech, dále popisuje prostředky použité pro měření rychlosti a zhodnocení výstupů aplikace. Poslední kapitola 6 celkově shrnuje výsledky práce a nastiňuje možnosti jejího pokračování.

Kapitola 2

Teoretický základ

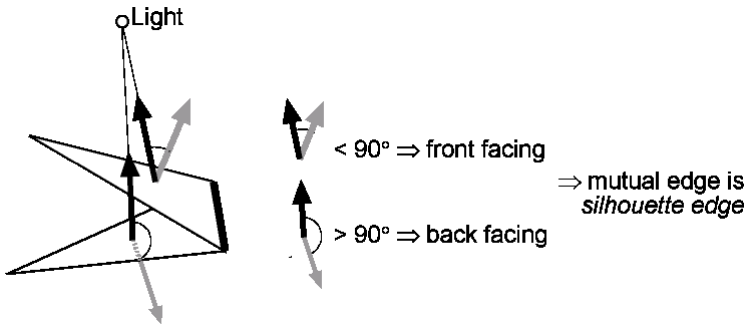
2.1 Metody stínování

Následující dvě podkapitoly přibližují dvě nejčastěji používané metody pro tvorbu stínů v interaktivních aplikacích, Stínová tělesa a Stínové mapy.

2.1.1 Shadow volumes

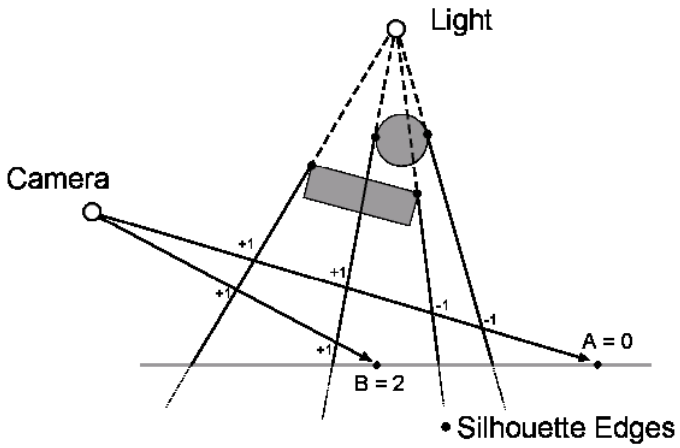
Vytváření stínů metodou stínových těles [9] objevil v roce 1977 Franklin C. Crow. Princip metody spočívá ve vytvoření tělesa, jehož objem pokrývá zastíněný prostor a určení, zda daný pixel leží uvnitř nebo vně tohoto tělesa. Stínové těleso je vytvořeno pro každý objekt scény vrhající stín. Sestává z trojúhelníků přivrácených ke světlu a stěn vytvořených na základě siluety z pohledu světla. Z každé hrany siluety je vytvořena stěna stínového tělesa, která je definována vertexy hrany a vertexy promítnutými do nekonečna ve směru určeném pozicí světla a pozicí vertexů hrany siluety. Při vytváření stěn je nutné dodržet správné pořadí vertexů, aby byly stěny správně přivrácené resp. odvrácené vzhledem ke kameře.

Hranu siluety je možné identifikovat podle vlastností přilehlých trojúhelníků. Je-li jeden trojúhelník přivrácený ke světlu a druhý odvrácený (obrázek 2.1), je jejich společná hrana součástí siluety. Přivrácenost resp. odvrácenost trojúhelníku je možné zjistit pomocí skalárního součinu normály trojúhelníku a vektoru udávajícího směr od povrchu trojúhelníku ke světlu. Tato metoda předpokládá uzavřené trojúhelníkové povrchy těles vrhajících stín. Konstrukci stínových těles, která se dříve prováděla na CPU, je v současnosti možné provést na grafické kartě. Provedení celého algoritmu na GPU zrychluje jeho výpočet a umožňuje jednodušší podporu stínů pro deformovanou a teselovanou geometrii [10].



Obrázek 2.1: Určení hran siluety. Převzato z [10].

Zastínění pixelu se určí podle jeho náležitosti do stínových těles. To lze realizovat inkrementací při průchodu paprsku přivrácenou stěnou a dekrementací pro odvrácenou stěnu. Na obrázku 2.2 můžeme vidět, že zastíněnému pixelu bude přiřazena kladná nenulová hodnota a osvětlenému pixelu nulová hodnota. V roce 1991 Tim Heidmann publikoval algoritmus využívající stencil buffer [11], který představoval alternativu k určování hodnoty pixelu vrháním paprsku a s rozšiřující se dostupností stencil bufferu ho nahradil.



Obrázek 2.2: Test zastínění. Převzato z [10].

Prvním krokem stencil algoritmu je vyčištění color, depth a stencil bufferu a vykreslení scény pouze s ambientním a emisivním osvětlením. Depth buffer nyní obsahuje hloubky viditelných fragmentů. Následující postup je opakován pro všechna světla. Provede se vykreslení stínového tělesa, které zapisuje pouze do stencil bufferu. Dnešní hardware umožňuje vykreslit stínové těleso v jednom průchodu díky možnosti nastavit parametry stencil bufferu zvlášť pro přivrácenou a odvrácenou geometrii. Přivrácená geometrie při průchodu z-testem inkrementuje hodnotu stencil bufferu, odvrácená geometrie hodnotu dekrementuje. Poté je nastaven stencil test tak, aby byly vykresleny pouze pixely s hodnotou stencil bufferu rovnou nule (osvětlené pixely) a provede se vykreslení osvětlené scény (aktuálním světlem), jehož výsledek je aditivně přidán k hodnotám color bufferu.

Výše zmíněný postup však nebere v potaz určitá úskalí. Pozice kamery uvnitř stínového tělesa způsobí invertování zastíněných a osvětlených ploch. Přední a zadní ořezová rovina pohledového objemu může způsobovat artefakty oříznutím stínového tělesa. Tyto problémy řeší Cass Everitt a Mark J. Kilgard v metodě robustních stínových těles [12] prezentované v roce 2002. Metoda využívá obrácený přístup pro získání hodnot stencil bufferu, namísto počítání průsečíků na paprsku od kamery k prvnímu viditelnému fragmentu počítá průsečíky na poprsku od nekonečna k prvnímu viditelnému fragmentu. Tento přístup, označovaný *zfail*, upravuje hodnoty stencil bufferu při selhání depth testu. U přivrácených stěn hodnotu dekrementuje, u odvrácených inkrementuje. Stínové těleso musí být uzavřené, proto je provedena projekce odvrácených trojúhelníků od světla do nekonečna, kde tvoří spodní stěnu stínového tělesa. Dále je nutné posunout zadní ořezovou rovinu do nekonečna, aby nedocházelo k oříznutí stínového tělesa. Projekce odvrácených trojúhelníků i posun zadní ořezové roviny do nekonečna je realizována pomocí homogenních souřadnic.

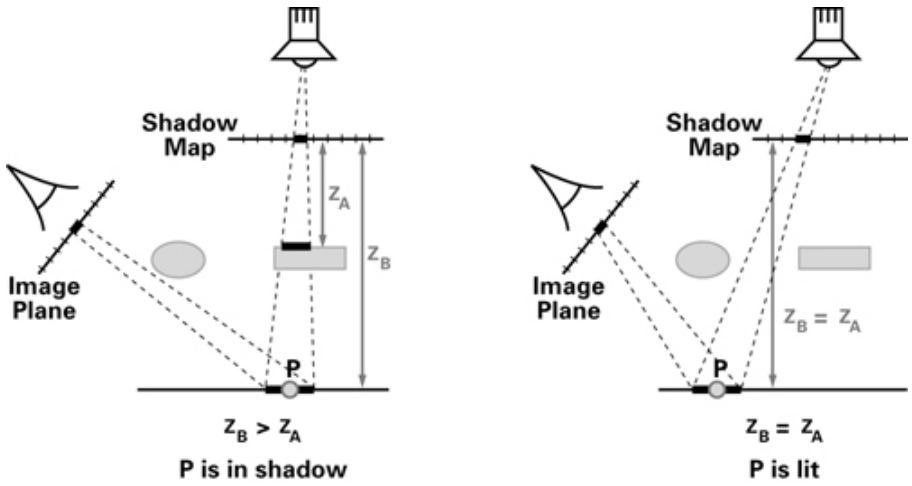
Projekční matice se zadní ořezovou rovinou v nekonečnu má následující podobu:

$$\lim_{Far \to \infty} \mathbf{P} = \mathbf{P_{inf}} = \begin{bmatrix} \frac{2 \times Near}{Right - Left} & 0 & \frac{Right + Left}{Right - Left} & 0 \\ 0 & \frac{2 \times Near}{Top - Bottom} & \frac{Top + Bottom}{Top - Bottom} & 0 \\ 0 & 0 & -1 & -2 \times Near \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{2.1}$$

Metoda robustních stínových těles však vyžaduje splnění určitých podmínek. Zastiňující objekty musí být uzavřené a složené z trojúhelníků (tento požadavek je možné obejít pomocí ošetření speciálních případů [13]). Lze použít pouze perspektivní projekci. Zdroje světla musí být ideální bodové (s využitím homogenních souřadnic umožňuje reprezentovat směrové zdroje světla). K dispozici musí být informace o spojení trojúhelníků pro vytvoření siluety. Počet bitů stencil bufferu musí být dostatečný pro maximální možný počet průchodů stínovými tělesy (8 bitů je pro běžné scény dostačující).

2.1.2 Shadow maps

Lance Williams v roce 1978 představil tvorbu stínů pomocí stínových map [14]. Jedná se o víceprůchodovou metodu využívající depth buffer. Scéna je vykreslena z pohledu světla pomocí transformační matice L_{mvp}, zaznamenává se však pouze hloubka do 2D textury (stínové mapy). Tvorba stínové mapy se opakuje pro každé světlo. Následuje vykreslení scény z pohledu kamery. Hloubka pixelu je porovnána s hloubkou ve stínové mapě každého světla (obrázek 2.3), je-li hloubka pixelu menší nebo rovna, je osvětlen daným světlem, v opačném případě leží ve stínu. Souřadnice ve stínové mapě jsou určeny pomocí matice L_{mvp}, musí být však upravena, aby výsledné souřadnice odpovídaly rozsahu souřadnic textury.



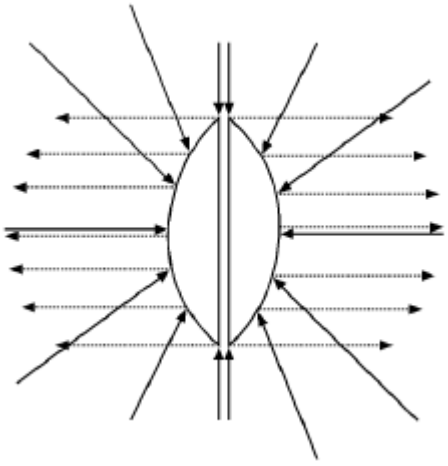
Obrázek 2.3: Určení zastínění podle hloubky. Převzato z [15].

Vzhledem ke kvantizaci hodnot hloubky a omezené přesnosti při výpočtu transformace se nemusí bod z osvětleného povrchu vyskytnout přesně na tomto povrchu, ale pod povrchem nebo nad ním. Při výskytu pod povrchem nástává zastínění na osvětleném povrchu. Tento problém se řeší odečtením malé hodnoty (bias) od hloubky bodu. Tato korekce může způsobit malé posunutí stínů, ovlivnění výsledné kvality je však mnohem menší, než při výskytu artefaktů na osvětlených plochách. Jiným řešením je vytváření stínové mapy pouze

z odvrácených trojúhelníků. Tento přístup neposouvá stíny, ale vyžaduje uzavřené modely se správnou orientací trojúhelníků.

Dalším problémem je omezené rozlišení stínové mapy, které způsobuje zoubkované okraje stínů, které jsou obzvláště patrné v blízkosti kamery. Metoda PCF [5] tento problém řeší úpravou vzorkování ze stínové mapy. Filtrace používaná u běžných textur nedává pro stínové mapy smysl, jelikož filtrované hodnoty jsou použity pro binární rozhodování (stíny zůstanou ostré) a filtrace nemá žádný vztah k geometrii scény. Metoda PCF provádí opačný postup, hodnoty jsou v určitém okolí nejprve porovnány a následně je provedena filtrace binárních hodnot, která určí míru zastínění daného pixelu. Omezení počtu přístupů do stínové mapy je možné pomocí náhodného vzorkování v ohraničující oblasti. Používá se uniformní nebo gaussovo rozložení. Druhým způsobem je rovnoměrné rozdělení ohraničující oblasti na podoblasti, kde se v každé podoblasti provede náhodný výběr.

Metoda umožňuje vytvářet stíny směrových světel. Pro podporu bodových zdrojů světla je potřeba určit hloubku scény ve všech směrech od světla (v nejhorším případě), k čemuž jedna stínová mapa nestačí. První možností je vytvoření šesti stínových map tvořících krychli. Pět vykreslovacích průchodů navíc však znatelně zpomalí výpočet a samozřejmě zvýší i spotřebu paměti. Stínové mapy je možné uložit do specializované textury (cube map), do které stačí pouze jeden přístup při porovnávání hloubky. Druhým přístupem je využití parabolické projekce. Dva paraboloidy stačí pro pokrytí všech směrů (obrázek 2.4), počet vytvářených stínových map se tak sníží z šesti na dvě. Hustota vzorkování není konstantní, liší se však maximálně 4-krát. Parabolická projekce však selhává v případě velkých polygonů. Rasterizace polygonů provádí interpolaci korektně vzhledem k perspektivní projekci, výsledné pozice proto nejsou shodné s parabolickou projekcí. Pro malé polygony je odchylka přijatelná. Velké polygony (např. stěny místnosti) musí být teselovány, nebo vynechány, pokud je předem známo, že nemohou vrhat stíny [8].



Obrázek 2.4: Pokrytí prostoru pomocí dvou paraboloidů. Převzato z [8].

2.2 Osvětlovací modely

Osvětlovací modely můžeme rozdělit do dvou hlavních kategorií. Lokální osvětlovací modely jsou jednoduché z hlediska implementace i potřebného výpočetního výkonu. Určují světlo odražené od objektu směrem k pozorovateli na základě přímého osvětlení ze světelných

zdrojů a vlastností materiálu objektu. Nejčastěji je používán Phongův osvětlovací model [16], který pracuje s difusní a odleskovou složkou světla. Jelikož jsou lokální osvětlovací modely dobře známé, zaměříme se v následujícím textu na globální osvětlovací modely.

Globální osvětlovací modely přidávají do scény kromě přímého osvětlení ze zdrojů světla také osvětlení nepřímé, odražené od objektů. Jelikož se tato práce zaměřuje na interaktivní aplikace, jsou v následujících podkapitolách popsány modely, u kterých je rychlost výpočtu dostatečná pro použití v těchto aplikacích. Dosažení potřebné rychlosti je možné díky poznatku, že ve většině případů nemusí být nepřímé osvětlení naprosto přesné a i přesto poskytuje přijatelnou vizuální kvalitu.

2.2.1 Reflective shadow maps

Tato metoda je vylepšením stínových map umožňující přibližný výpočet nepřímého osvětlení pro dynamické scény. Výpočet je prováděn pouze pro jeden odraz světla od objektů. Princip metody spočívá ve využití stínové mapy k určení míst odrazu světla. Pro jeden zdroj světla jsou všechna místa odrazu zachycená ve stínové mapě vytvořené z pohledu tohoto zdroje, každý pixel stínové mapy tak reprezentuje sekundární zdroj světla. Metoda využívá odloženého stínování (deferred shading) a pracuje převážně v prostoru obrazu (screen-space), proto je její rychlost z větší části nezávislá na komplexnosti scény.

Stínová mapa obsahuje v každém pixelu p kromě hloubky d_p také pozici ve světě x_p, normálu n_p a odrážený zářivý tok Φ_p viditelného bodu povrchu objektu. Pozici x_p je možné spočítat ze souřadnic pixelu p a hloubky d_p, přímý přístup k této hodnotě ve stínové mapě však sníží náročnost výpočtu. Zářivý tok Φ_p určuje jas, normála n_p popisuje prostorovou vyzařovací charakteristiku. Za předpokladu, že sekundární světelný zdroj je nekonečně malý, lze vyjádřit intenzitu zářivosti ve směru ω vztahem $I_p(\omega) = \Phi_p \langle n_p | \omega \rangle_+$, kde $\langle | \rangle_+$ představuje skalární součin, ve kterém jsou záporné výsledky nahrazeny nulou. Ozáření bodu x s normálou n sekundárním zdrojem světla x_p je možné vypočítat pomocí následující rovnice:

$$E_p(x, n) = \Phi_p \frac{\langle n_p | x - x_p \rangle_+ \langle n | x_p - x \rangle_+}{||x - x_p||^4} \tag{2.2}$$

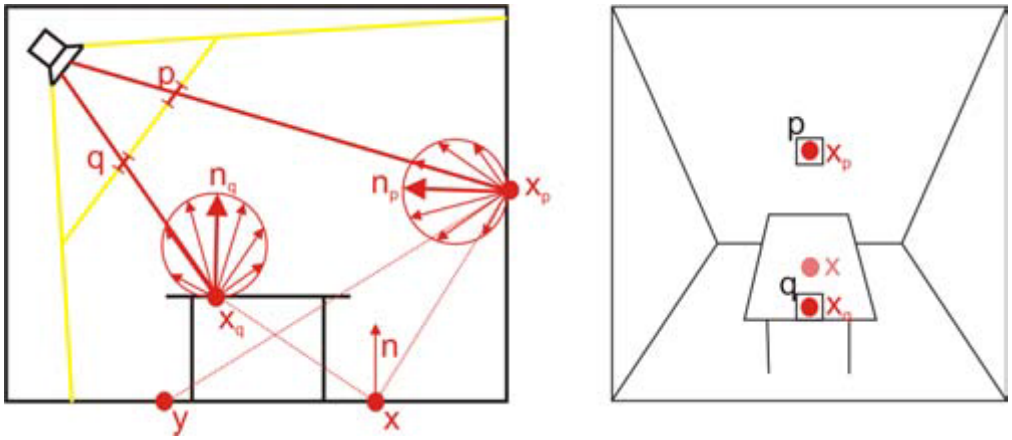
Výsledné nepřímé osvětlení bodu x s normálou n je pak suma dílčích ozáření z jednotlivých sekundárních zdrojů:

$$E(x, n) = \sum_{\text{pixels}p} E_p(x, n) \tag{2.3}$$

Metoda nebere v úvahu viditelnost z pohledu sekundárních zdrojů světla. Bod y na obrázku 2.5 je osvětlen ze sekundárního zdroje světla x_p, i když je bod y zastíněn stolem. Toto zjednodušení může vést ke špatným výsledkům.

Běžná velikost stínové mapy (např. 512x512) by produkovala příliš velké množství sekundárních zdrojů světla, proto je jejich počet omezen na přijatelnou hodnotu (např. 400). Výběr vzorků je prováděn podle důležitosti, která je vyjádřena jejich vzdáleností od osvětlovaného bodu x v prostoru světa (world-space). Pokud jsou body v prostoru světa blízko sebe, musí být blízko sebe také v souřadnicích stínové mapy. Toto neplatí naopak, protože blízké pixely ve stínové mapě se mohou značně lišit hloubkou a tím pádem i vzdáleností v prostoru světa. Blízké body však nepřispívají k osvětlení bodu x, pokud leží ve stejné rovině nebo pokud jejich normála míří směrem od bodu x. Přesto je výběr na základě vzdálenosti nejlepším řešením a pozice vzorků je určena následovně:

$$(s + r_{\max}\xi_1 \sin(2\pi\xi_2), t + r_{\max}\xi_1 \cos(2\pi\xi_2)) \tag{2.4}$$



Obrázek 2.5: Znázornění nepřímých zdrojů osvětlení x_p a x_q odpovídajících pixelům stínové mapy p a q. Převzato z [17].

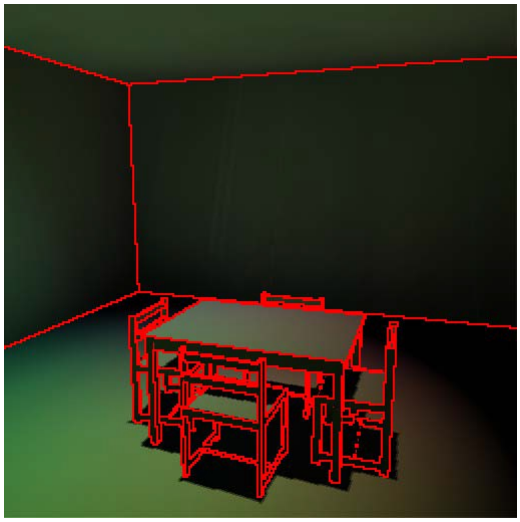
Souřadnice s a t jsou určeny promítnutím bodu x do stínové mapy, ξ_1 a ξ_2 jsou náhodná čísla s rovnoměrným rozložením pravděpodobnosti. Hustota vzorků v okolí (s, t) klesá se čtvercem vzdálenosti, což je dosaženo pomocí výběru vzorků v polárních souřadnicích. Měnící se hustota vzorků je kompenzována váhováním s ξ_1^2. Rozložení vzorků je možné předpočítat a používat pro všechny pixely, což omezí blikání v dynamických scénách a zrychlí výpočet.

Výpočet nepřímého osvětlení pro každý pixel je i přes omezení počtu sekundárních zdrojů světla příliš náročný pro využití v interaktivních aplikacích. Zrychlení se dosahuje pomocí interpolace nepřímého osvětlení. Výpočet je nejprve proveden pro obraz s nízkým rozlišením z pohledu kamery. Poté je vykreslován obraz v plném rozlišení. Interpolace proběhne pokud jsou tři nebo čtyři okolní vzorky z obrazu s nízkým rozlišením vyhovující, v opačném případě je pixel zahozen. Vzorek je považován za vyhovující, pokud je jeho normála a pozice v prostoru světa podobná normále a pozici počítaného pixelu. Zahozené pixely jsou doplněny v posledním vykreslovacím průchodu a hodnota jejich nepřímého osvětlení je vypočítána rovnicí 2.3. Na obrázku 2.6 je možné vidět míru využití interpolace. Interpolace je velmi efektivní pro rovné a nečlenité objekty. Naopak pro komplexní objekty (např. strom) interpolaci není možné použít [17].

2.2.2 Light propagation volumes

Metoda Light propagation volumes (LPV) provádí výpočet nepřímého osvětlení pro jeden i více odrazů v dynamických scénách. Na rozdíl od Reflective shadow maps pracuje s viditelností pro nepřímé osvětlení. Metodu je možné rozšířit tak, aby uvažovala vliv prostředí, ve kterém se světlo šíří. Dalším rozšířením jsou odlesky. Princip LPV je založen na výpočtu šíření světla skrze prostorovou mřížku. Určování viditelnosti je realizováno pomocí další prostorové mřížky uchovávající vzorky scény. Vnořené mřížky umožňují interaktivní zobrazení i pro komplexní scény. Výpočet sestává ze čtyř kroků:

1. Inicializace LPV mřížky z povrchů způsobujících nepřímé osvětlení.
2. Vzorkování povrchů scény pro naplnění mřížky uchovávající prostorovou reprezentaci překážek.
3. Šíření světla v LPV mřížce a akumulace výsledků.

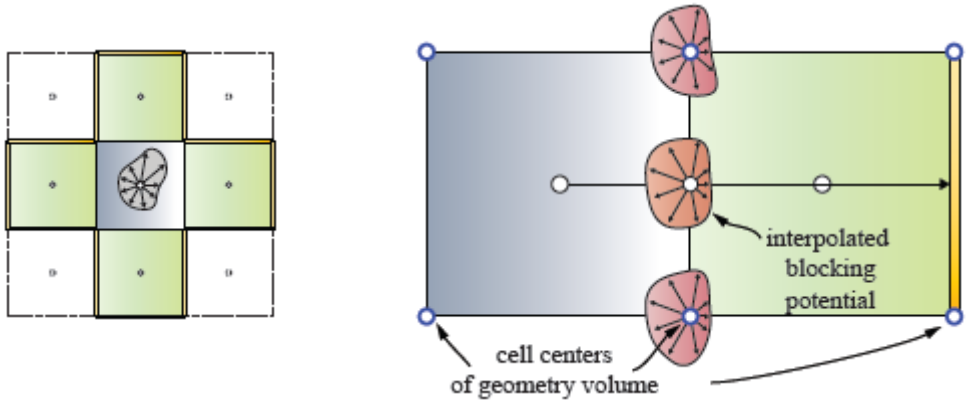


Obrázek 2.6: Interpolace nepřímého osvětlení. Pouze pro červeně zvýrazněné pixely nebylo možné použít interpolaci. Převzato z [17].

4. Aplikování vypočítaného osvětlení do scény.

Fáze inicializace LPV mřížky začíná vykreslením Reflective shadow mapy (RSM) pro každý zdroj světla. Každý pixel p mapy odpovídá sekundárnímu zdroji světla (SZS) popsaného funkcí distribuce intenzity $I_p(\omega) = \Phi_p \langle n_p | \omega \rangle_+$, kde n_p je normála pixelu p, Φ_p udává zářivý tok a ω je požadovaný směr. Pro jednoduchost je vynechána závislost na spektru, následující postup se však provádí pro všechny složky RGB. Velký počet SZS je přijatelný, protože se použijí pouze při inicializaci mřížky a jejich počet tím pádem nezpomaluje výpočet šíření světla. Pozici SZS v mřížce lze snadno určit z mapy. Směřuje-li normála od středu buňky, může nastat osvětlení nebo zastínění buňky sama sebou. Řešením tohoto problému je posunutí SZS o polovinu velikosti buňky ve směru normály před určením pozice v mřížce. Hodnoty směrové intenzity v buňkách mřížky jsou reprezentováný pomocí sférických harmonických (spherical harmonics) [18], vyjádřených koeficienty $c_{l,m}$ a bázovými funkcemi $y_{l,m}(\omega)$. Platí $-l \leq m \leq l$, kde l je index pásma a m je stupeň. Počet koeficientů odpovídá druhé mocnině počtu pásem (používají se 2 až 4 pásma). Po výpočtu koeficientů na základě normály n_p je provedeno váhování hodnotou zářivého toku. Pokud se do jedné buňky přiřadí více SZS, koeficienty se sčítají.

Druhým krokem je inicializace mřížky reprezentující povrchy ve scéně (geometry volume, GV) sloužící k vytváření stínů z nepřímého osvětlení. Tato mřížka má stejné rozlišení jako LPV mřížka, je však posunutá o polovinu buňky, jak je možné vidět na obrázku 2.7b. Účelem posunutí je snažší interpolace při vyhodnocování překážek. K inicializaci se využívají vzorky získané při vytváření RSM a vykreslování pohledu kamery v předchozím kroku. Není-li množství takto dostupných vzorků dostatečné, lze k vykreslování RSM nebo pohledu kamery přidat depth-peeling průchod. Každý vzorek reprezentuje surfel (část povrchu), jehož vlastnosti jsou pozice, orientace určená normálou n_s a velikost A_s. Viditelnost není reprezentována binárně, ale pomocí pravděpodobnosti zastínění světla. Pravděpodobnost, že surfel zablokuje světlo šířící se směrem ω pro mřížku s velikostí buňky s je $B(\omega) = A_s s^{-2} \langle n_s | \omega \rangle_+$. Pravděpodobnost se poté převádí do reprezentace sférických harmonických, která se ukládá do mřížky. Jelikož vzorky pocházejí z různých pohledů, může být jeden surfel zachycen ve více vzorcích. V takovém případě se do mřížky uloží



(a) Šíření světla ve směru os. (b) Určování zastínění.

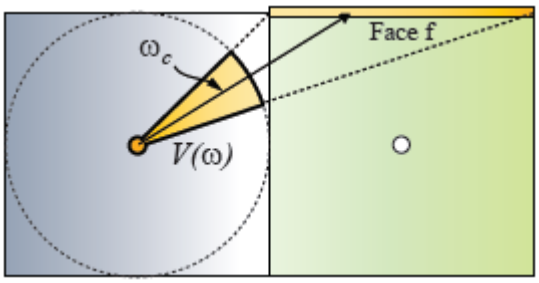
Obrázek 2.7: Šíření světla a určování zastínění. Zdrojové buňky jsou vyznačeny modrým pozadím, cílové zeleným. Převzato z [19].

maximu z vektorů koeficientů sférických harmonických. Takto vytvořená mřížka umožňuje vykreslení měkkých stínů pro nepřímé osvětlení, pokud jsou povrchy objektů větší než velikost buňky. Dále se předpokládá uzavřenost povrchů. Tato metoda neumožňuje vzájemné zastínění povrchů v rámci jedné buňky, což se nahrazuje použitím ambient occlusion.

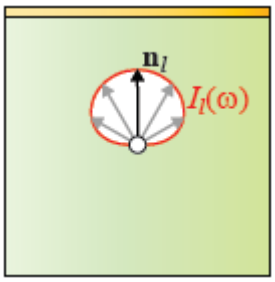
Šíření světla probíhá po krocích. Vstupem prvního kroku je inicializovaná LPV mřížka, v dalších krocích jsou vstupem výsledky předchozích kroků. Obrázek 2.7a ukazuje šíření podél os ve čtyřech směrech, ve 3D mřížce je to směrů šest. Pro žlutě označené stěny cílových buněk se provede výpočet dopadajícího zářivého toku. Šíření světla ze zdrojové buňky do cílové můžeme vidět na obrázku 2.8a. Funkce $V(\omega)$ určuje viditelnost stěny f cílové buňky vzhledem ke středu zdrojové buňky. Pokud paprsek ze středu zdrojové buňky ve směru ω protíná stěnu f, pak $V(\omega) = 1$, v opačném případě $V(\omega) = 0$. Na základě funkce $V(\omega)$ je určen prostorový úhel $\Delta\omega_f = \int_\Omega V(\omega)\mathrm{d}\omega$. Z $\Delta\omega_f$ je určen centrální směr kuželu viditelnosti ω_c. Zářivý tok dopadající na stěnu f je $\Phi_f = \Delta\omega_f/(4\pi) \cdot I(\omega_c)$, tím pádem je intenzita ve směru ω_c považována za průměrnou intenzitu v rámci prostorového úhlu. Intenzitu lze přibližně vyjádřit jako $I(\omega) \approx \sum_{l,m} c_{l,m} y_{l,m}(\omega)$. Intenzita je snížena podle pravděpodobnosti překážky, která je bilineárně interpolována z GV mřížky. Překážky se neberou v úvahu v prvním kroku z důvodu sebe zastínění. Následuje vytvoření zdroje světla ve středu cílové buňky (obrázek 2.8b), které vyšle k dané stěně stejné množství zářivého toku jako bylo vypočítáno šířením ze zdrojové buňky. Vztah mezi tokem Φ_f dopadajícím na stěnu f a celkovým vyzářeným tokem ze zdroje světla Φ_l je $\Phi_f = \int_\Omega \Phi_l \langle n_l|\omega\rangle_+ \mathrm{d}\omega$ a tudíž $\Phi_l = \Phi_f/\pi$. Zdroj světla je poté převeden na reprezentaci pomocí sférických haromonických stejným způsobem jako při inicializaci LPV mřížky.

Posledním krokem je aplikace vypočítaného osvětlení. Nejjednodušším způsobem je trilineárně interpolované vyhledávání koeficientů sférických harmonických. Následuje získání funkce intenzity a její vyhodnocení pro směr opačný k normále osvětlovaného povrchu. Intenzita je převedena na zář, přičemž jako vzdálenost od zdroje k povrchu se uvažuje polovina velikosti buňky mřížky.

Počet kroků při výpočtu šíření ovlivňuje dosah osvětlení a délku výpočtu. Potřebný počet kroků závisí na rozlišení mřížky a měl by dosahovat dvojnásobku nejdelší strany mřížky. Takový počet kroků je však příliš velký pro použití v interaktivních aplikacích, využívají se proto kaskádové mřížky pohybující se s kamerou (obrázek 2.9), u kterých je



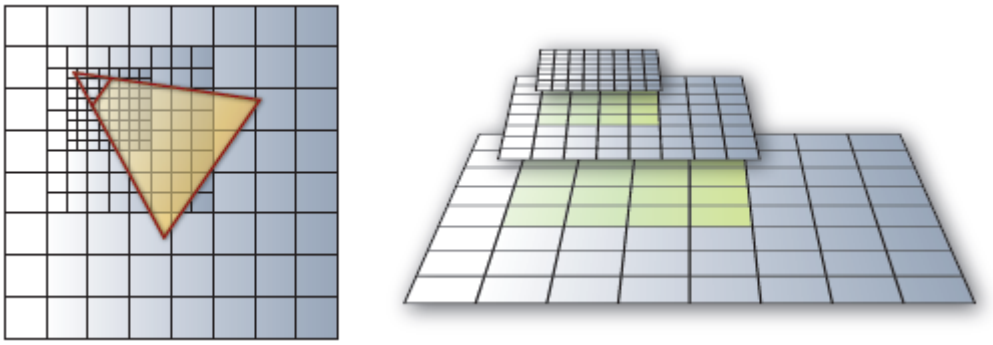
(a) Výpočet zářivého toku dopodajícího na stěnu sousední buňky.



(b) Rekonstrukce zdroje světla.

Obrázek 2.8: Výpočet přeneseného zářivého toku a jeho rekonstrukce do bodového zdroje světla. Zdrojové buňky jsou vyznačeny modrým pozadím, cílové zeleným. Převzato z [19].

dostačující menší počet kroků. Dostačující velikost mřížky je 32^3 a jako vhodný počet kroků šíření je stanoveno číslo osm [19].



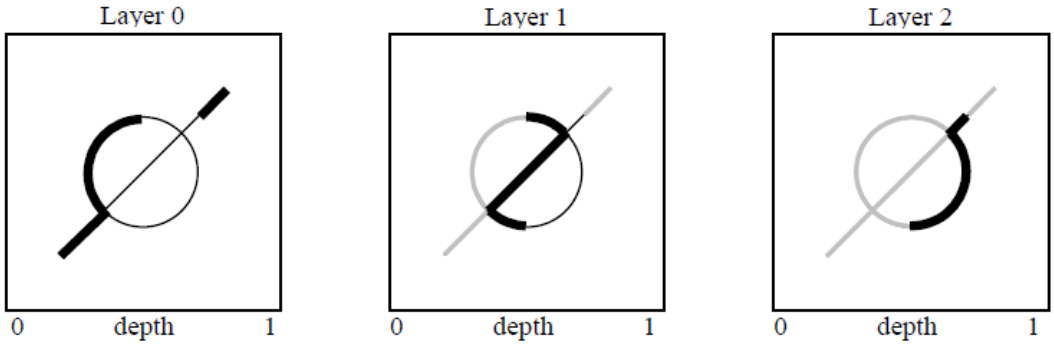
Obrázek 2.9: Umístění mřížky vzhledem ke kameře a vnoření mřížek. Převzato z [19].

2.3 Depth peeling

Částečně průhledné objekty je v OpenGL nutné vykreslovat ve správném pořadí (od nejvzdálenějšího k nejbližšímu), což vyžaduje seřazení. Metoda Depth peeling [20] umožňuje vykreslení průhledných objektů bez nutnosti řazení.

Vykreslování probíhá po vrstvách viditelnosti (obrázek 2.10). Hloubka v rámci vrstvy je proměnlivá a množství vzorků se zmenšuje. Počet nutných vykreslovacích průchodů odpovídá počtu vrstev scény s jedním průchodem navíc pro záverečné sestavení scény. První vrstva je vykreslena běžným způsobem. Při vykreslování následující vrstvy se využijí hloubky získané v předchozím průchodu k odstranění předchozích vrstev. Fragmenty s nižší nebo stejnou hloubkou jsou zahozeny. Obraz každé vrstvy je uložen do textury. V posledním průchodu je v prostoru obrazu vykreslen obdélník, na který jsou přimíchány jednotlivé textury vrstev v pořadí od nejvzdálenější k nejbližší.

Vylepšením této metody je Dual depth peeling [21], který snižuje nutný počet průchodů pro N vrstev na hodnotu $N/2+1$. Vrstvy jsou procházeny zároveň ve směru zepředu dozadu i odzadu dopředu. V jednom průchodu jsou tak zpracovány dvě vrstvy. Prostřední vrstva by však mohla být zpracována z obou směrů, aby se tomu zabránilo, využívá se posuvné okno



Obrázek 2.10: Vykreslování po vrstvách. Převzato z [20]

nad dvěma po sobě jdoucími vrstvami. To vyžaduje jeden inicializační průchod, proto není hodnota nutných průchodů $N/2$. Inicializační průchod nastaví hloubku na minimum resp. maximum pro první resp. poslední vrstvu. Další průchody již zpracují vrstvy podle hloubek z předchozí vrstvy posuvného okna. Prostřední vrstvu je možné zpracovat z libovolně zvoleného směru.

Pro oba směry průchodu je vytvořena textura akumulující barvy vrstev. Po dokončení průchodů jsou tyto dvě textury spojeny do konečného výsledku. Průchod ve směru odzadu dopředu využívá běžné aditivní přimíchávání vyjádřené vztahem:

$$C_{dst} = A_{src}C_{src} + (1 - A_{src})C_{dst} \tag{2.5}$$

Průchod v opačném směru však vyžaduje úpravu tohoto vztahu (under blending). Úprava využívá alfa složku obou míchaných barev, jak je vidět v rovnici 2.6. Hodnota A_{dst} je inicializována na 1 a poté se aktualizuje na základě rovnice 2.7. Posledním krokem je přimíchání barvy pozadí podle rovnice 2.8.

$$C_{dst} = A_{dst}(A_{src}C_{src}) + C_{dst} \tag{2.6}$$
$$A_{dst} = (1 - A_{src})A_{dst} \tag{2.7}$$
$$C_{dst} = A_{dst}C_{bg} + C_{dst} \tag{2.8}$$

2.4 Stochastic Transparency

Metoda Stochastic Transparency [22] je založena na principu Screen-door transparency, kde jsou pixely průhledných povrchů buď zcela průhledné (stav off), nebo neprůhledné (stav on). Poměr on a off pixelů závisí na míře průhlednosti daného tělesa. Výsledný efekt napodobuje průhlednost. Stejně jako v případě Depth Peelingu není třeba řazení průhledných objektů. Vylepšení kvality zobrazení je možné pomocí vhodného rozložení průhledných pixelů. Problém nastává v případě použití shodných vzorů rozložení pixelů ve všech vrstvách průhledných objektů. Vzory se pak vzájemně překrývají, což způsobuje vizuální artefakty namísto efektu průhlednosti.

Stochastická průhlednost používá náhodné rozložení na úrovni sub-pixelů, které způsobuje nezávislost rozložení mezi jednotlivými vrstvami, což snižuje riziko překrývání. Náhodné rozložení však do obrazu vnáší šum, proto se používá řada dalších metod pro potlačení tohoto šumu. Snížení množství šumu je možné provést zvýšením počtu vzorků na pixel. Snížení průměrné chyby na polovinu však vyžaduje čtyřnásobný počet vzorků. Chybu lze snížit použitím rozvrstveného vzorkování (stratified sampling). Určení viditelnosti vzorků

neprobíhá nezávisle pro jednotlivé vzorky, ale pro celou skupinu. Je-li S počet vzorků na pixel a R počet vzorků pokrytých průhledným fragmentem, pak $\alpha = R/S$. Například pro $S = 4$ a $\alpha = 0,45$ může R nabývat hodnot od 0 do 4, průměrně to však bude $\alpha S = 1,8$. Rozvrstvené vzorkování určuje počet pokrytých vzorků vztahem $R_i = \lfloor \alpha_i S + \xi \rfloor$, kde ξ je náhodné číslo s rovnoměrným rozložením z intervalu $\langle 0, 1)$. Pro výše uvedený příklad tak R bude nabývat hodnoty 1 s pravděpodobností 20% a hodnoty 2 s pravděpodobností 80%. Jsou tak vyloučeny hodnoty 0, 3 a 4.

Další možností pro potlačení šumu je alfa korekce. Celkovou hodnotu alfa všech průhledných fragmentů v daném pixelu lze exaktně určit vztahem $\alpha_{total} = 1 - \prod(1 - \alpha_i)$. Jelikož je tento vztah nezávislý na pořadí fragmentů, může být vyhodnocen v jednom průchodu bez nutnosti řazení. Získaná α_{total} se použije ke korekci průměrných (náhodně získaných) barev jejich vynásobením hodnotou $\alpha_{total}/(R/S)$. Pro pixely s jednou průhlednou vrstvou je výsledek přesný. Pro více vrstev se u některých pixelů může chyba zvýšit, empiricky se však celkově sníží. V případě stejné barvy průhledných fragmentů je výsledná chyba nulová, proto tato metoda podává nejlepší výsledky, pokud mají fragmenty podobnou barvu.

Metoda Depth-based Stochastic Transparency využívá aproximaci viditelnostní funkce. Viditelnostní funkce určuje jak velká část světla dosáhne hloubky z pro daný pixel. Hodnota viditelnostní funkce odpovídá vztahu:

$$\mathrm{vis}(z) = \prod_{z_i < z} (1 - \alpha_i) \tag{2.9}$$

Tuto viditelnostní funkci lze aproximovat hrubým, ale kompaktním vztahem, který nevyžaduje řazení hloubek:

$$\mathrm{vis(z)} \approx \mathrm{count}(z \leq z_i)/S \tag{2.10}$$

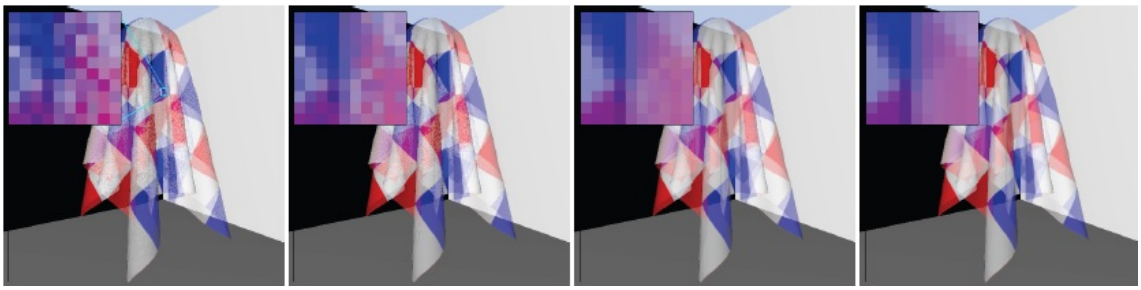
Výsledná barva pixelu odpovídá příspěvkům fragmentů v závislosti na jejich průhlednosti a viditelnosti, jak je znázorněno v následujícím vztahu, kde c_i je barva fragmentu:

$$\text{Final color} = \sum \mathrm{vis}(z_i)\alpha_i c_i \tag{2.11}$$

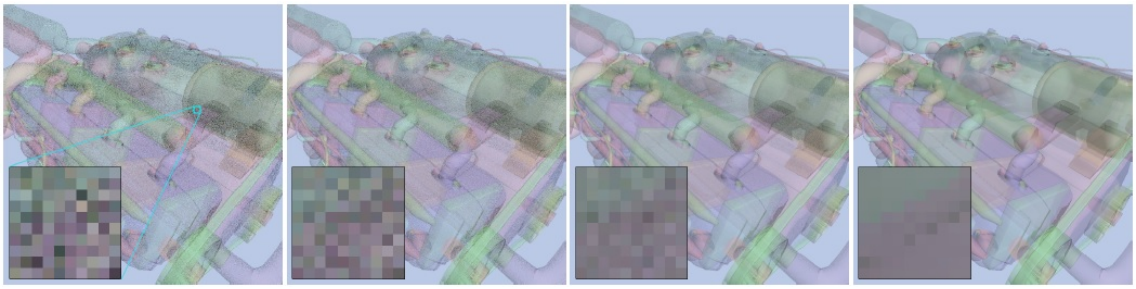
Postup metody pro vzorky ve středech pixelů je následující:

1. Vykreslení všech neprůhledných objektů a pozadí. Následující kroky pracují pouze s průhlednými objekty a zahazují všechny objekty zakryté neprůhlednými objekty.
2. Určení a zápis celkové alfy do odděleného bufferu.
3. Vykreslení stochastické průhlednosti s S vzorky na pixel a uchování pouze hloubek z.
4. Akumulace barev fragmentů podle rovnice 2.11, s viditelností odhadnutou podle rovnice 2.10. Shader porovnává hloubku aktuálního fragmentu s hloubkami z předchozího kroku.
5. Složení barev sum fragmentů násobených alfa korekcí s neprůhlednými objekty scény.

Tato modifikace využívá namísto kvantizace alfy na násobky $1/S$ kvantizaci viditelnostní funkce. K výsledku navíc přispívají všechny fragmenty, ne jen ty vybrané, jako u základní verze Stochastické průhlednosti. Výsledek proto obsahuje mnohem méně šumu a je přesnější. Chyba je natolik malá, že použití alfa korekce ji zvýší, zabraňuje však vzniku artefaktů v některých scénách.



Obrázek 2.11: Porovnání variant Stochastické průhlednosti. Z leva základní varianta, varianta s alfa korekcí, varianta Depth-based a referenční výsledek z Depth Peelingu. Převzato z [22]



Obrázek 2.12: Porovnání Stochastické průhlednosti a Depth Peelingu. Z leva Stochastická průhlednost s 8, 16, 64 vzorky na pixel, Depth Peeling. Převzato z [22]

Na obrázku 2.12 můžeme vidět porovnání jednotlivých variant redukce šumu. Alfa korekce přináší dobré výsledky v oblastech s podobnou barvou, Depth-based varianta funguje lépe v komplexních oblastech. Obrázek 2.12 znázorňuje závislost kvality na počtu vzorků a porovnání s výsledky metody Depth Peeling. Metoda Depth Peeling je schopná poskytnout kvalitnější výsledek, ale je pomalejší a počet nutných vykreslovacích průchodů stoupá s komplexností scény.

Metodu Stochastic transparency je možné použít nejen pro zobrazování průhledných těles, ale i pro konstrukci stínových map. Nejjednodušší možností je použití jednoho vzorku na pixel při vyšším rozlišení stínové mapy. Míra zastínění se pak aproximuje pomocí metody Percentage Closer Filtering. Druhou možností je uchování více hloubek na pixel, které mohou aproximovat viditelnostní funkci podle rovnice 2.10.

2.5 Hluboké stínové mapy

Hluboké stínové mapy [23] vychází z tradičních stínových map. Jejich účelem je efektivní tvorba stínů pro částečně průhledné objekty a pro objekty malých rozměrů, jako například vlasy nebo srst. Tradiční stínové mapy nepodporují průhlednost objektů a pro vytvoření stínů malých objektů musí mít velmi vysoké rozlišení. Hluboké stínové mapy umožňují zobrazení stejně kvalitních stínů při nižším rozlišení, navíc umožňují zpracování objemových jevů, jako jsou mlha a kouř. Nezanedbatelnou výhodou je podpora mip-mappingu a možnost rozmazání stínů pohybem (motion blur).

Pixely hluboké stínové mapy uchovávají namísto hloubky funkci viditelnosti $V_{i,j}(z)$, která určí míru světla pronikajícího do zadané hloubky podél paprsku od zdroje světla skrz

pixel se souřadnicemi (i, j). Funkce viditelnosti dokáže podobně jako alfa kanál u obrázků určit míru viditelnosti na základě průhlednosti objektu, případně částečného obsazení pixelu objektem (pokud objekt nezakrývá celou plochu pixelu, jsou v tomto pixelu částečně viditelné objekty ve vyšší hloubce). Hodnota alfa kanálu však reprezentuje zastínění namísto viditelnosti a to pro hloubku odpovídající nekonečnu. Vztah mezi alfa kanálem a funkcí viditelnosti tudíž můžeme vyjádřit jako $\alpha_{i,j} = 1 - V_{i,j}(\infty)$. Hloubková stínová mapa odpovídá výpočtu hodnot $1 - \alpha$ pro všechny hloubky a jejich uložení ve formě funkce s parametrem hloubky.

Obor hodnot funkce viditelnosti leží v intervalu od nuly do jedné a lze ji definovat následovně:

$$V_{i,j}(z) = \int_{-r}^{r}\int_{-r}^{r} f(s,t)\tau(i+\frac{1}{2}-s, j+\frac{1}{2}-t, z)\mathrm{d}s\,\mathrm{d}t \tag{2.12}$$

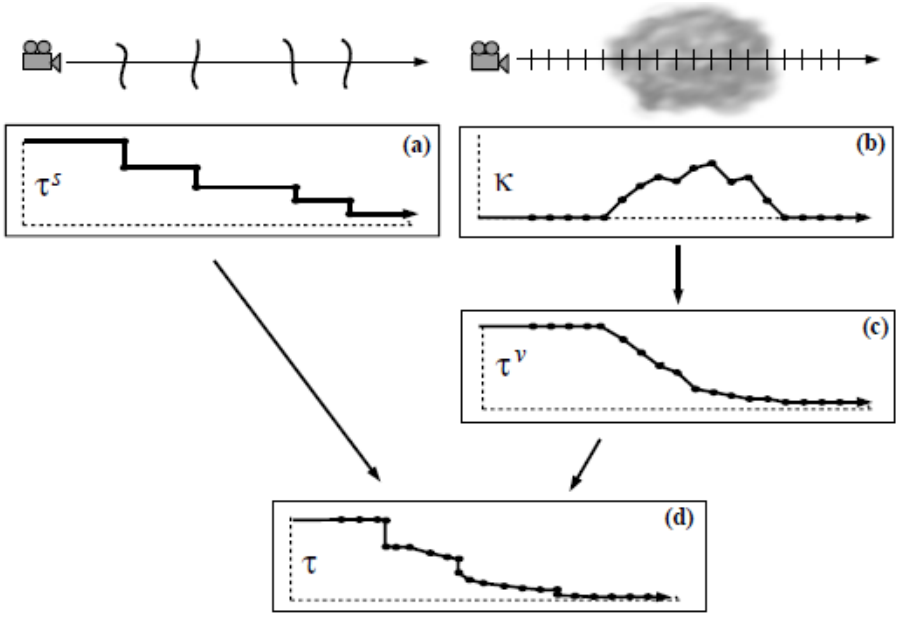
Transmitance $\tau(x, y, z)$ podobně jako funkce viditelnosti určuje míru pronikajícího světla na daných souřadnicích obrazu do hloubky z. Rozdílem je, že (x, y) nejsou souřadnice pixelu, ale souřadnice bodu v obrazu (v rámci jednoho pixelu může být definováno více transmitancí). Bod $(i+\frac{1}{2}, j+\frac{1}{2})$ představuje střed pixelu. Viditelnostní funkce pro hloubku z vznikne filtrováním hodnot transmitancí v této hloubce. Hodnota r udává poloměr filtru f.

Transmitance v bodě (x, y) je složena ze dvou složek, povrchové τ^s a objemové τ^v. Povrchová složka je určena z povrchů protnutých paprskem procházejícím bodem (x, y). Zjišťování protnutých povrchů je možné realizovat pomocí vrhání paprsků nebo rasterizérem s vypnutým depth testem. Každý průsečík i obsahuje údaj o hloubce z_i^s a neprůhlednosti O_i. Okrajové body $z = 0$ s hodnotou 1 a $z = \infty$ s hodnotou 0 jsou uvažovány implicitně, nejsou tudíž uloženy v τ^s. Každý průsečík přidá do τ^s dva body v hloubce z_i^s, první s hodnotou předchozího bodu, druhý s hodnotou předchozího bodu násobenou $1 - O_i$. Vytvořená funkce τ^s je schodovitého tvaru (obrázek 2.13a), nespojitosti jsou odstraněny v průběhu komprimace popsané dále v textu. Objemová složka τ^v je sestavena vzorkováním hustoty podél paprsku v pravidelných rozestupech. Každý vzorek je reprezentován hloubkou z_i^v a koeficientem zániku κ_i. Lineární interpolací mezi vzorky je vytvořena zániková funkce κ (obrázek 2.13b). Transmitance se určí ze zánikové funkce rovnicí 2.13, výsledná funkce však není po částech lineární. Funkce je proto aproximována, pomocí rovnice 2.15 je vypočítána transmitance pro jednotlivé body, ze kterých je sestavena funkce τ^v (obrázek 2.13c) podobným postupem jako u povrchové složky. Rozdílem je použití interpolace namísto diskrétních přechodů, nevznikají tak nespojitosti. Výsledná transmitance je výsledkem součinu povrchové a objemové složky, jelikož také není po částech lineární, je provedena interpolace mezi body zkombinovanými z τ^s a τ^v.

$$\tau^v(z) = \exp(-\int_0^z \kappa(z')\mathrm{d}z') \tag{2.13}$$

$$T_i = \exp(-(z_{i+1}^v - z_i^v)(\kappa_{i+1} + \kappa_i)/2) \tag{2.14}$$

Hloubková stínová mapa vzniká pořízením určitého počtu vzorků (například 16) pro každý pixel. Vzorky jsou rozmístěny v jednotlivých buňkách mřížky a náhodně umístěny v rámci buňky. Pro každý vzorek je určena funkce transmitance. Výpočet funkce viditelnosti pro pixel (i, j) v hloubce z je realizován filtrováním transmitancí pomocí rovnice 2.15, kde n je počet transmitancí v rámci poloměru filtru se středem v $(i+\frac{1}{2}, j+\frac{1}{2})$ a w_k je normalizovaná



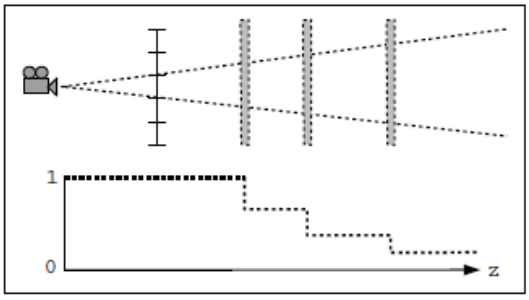
Obrázek 2.13: Sestavení funkce transmitance. Převzato z [23].

váha filtru odpovídající bodu (x_k, y_k). Výsledkem je po částech lineární funkce s počtem bodů přibližně n-krát větším než je počet bodů u funkcí transmitance. Na obrázku 2.14 jsou ukázky funkce viditelnosti pro různé typy objektů.

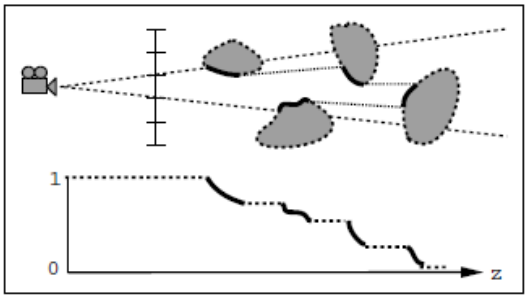
$$V_{i,j}(z) = \sum_{k=1}^{n} w_k \tau_k(z) \tag{2.15}$$

Počet bodů viditelnostní funkce může být velký, závisí na poloměru filtru a počtu vzorků na pixel. Průběh funkce je však většinou hladký, tím pádem je možné provést efektivní kompresi. Komprimovaná funkce je uchována jako pole dvojic desetinných čísel uchovávajících hloubku z a míru viditelnosti V. Komprese musí zachovat přesnost hloubky v celém intervalu $[0, \infty)$, jinak hrozí vznik artefaktů sebe zastínění. Komprese vytváří novou funkci V', pro kterou platí $|V'(z) - V(z)| \leq \epsilon$, kde ϵ je zvolená maximální chyba pro hodnotu V. Příklad vstupní funkce V a její povolené chyby je znázorněn na obrázku 2.15a. Kompresní algoritmus postupuje po jednotlivých bodech funkce ve směru rostoucí hloubky. V každém kroku se algoritmus pokouší vytvořit co nejdelší segment z počátečního bodu takovým způsobem, aby nebyla překročena maximální povolená chyba. Pro jednodušší implementaci jsou omezeny hloubky výstupních bodů na podmnožinu hloubek vstupních bodů. Uvažujme počátek segmentu (z'_i, V'_i), v každém kroku je určen rozsah možného sklonu nového segmentu $[m_{lo}, m_{hi}]$ (na počátku je hodnota $[-\infty, \infty]$). Každý další bod (z_j, V_j) vstupní funkce V omezuje rozsah sklonu oknem v hloubce z_j s krajními body $V_j \pm \epsilon$ (obrázek 2.15b). Rozsah sklonu je postupně omezován dalšími body dokud není interval prázdný (obrázek 2.15c), poté je vytvořen bod funkce V' s hodnotou $(m_{lo} + m_{hi})$ z posledního neprázdného rozsahu sklonu s hloubkou z posledního navštíveného bodu. Vytvořený bod se stává počátkem dalšího segmentu a proces se opakuje. Výsledná funkce V' má obvykle mnohem méně bodů než vstupní funkce, porovnání je možné vidět na obrázku 2.15d.

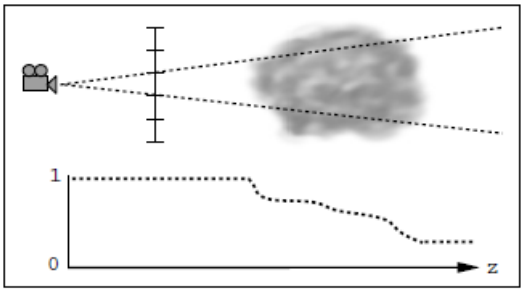
Dotazování do hluboké stínové mapy probíhá podobně jako u běžných textur pomocí převzorkování a rekonstrukčního filtru pracujícím nad čtvercovým polem pixelů. Dotaz na



(a) Průhledné objekty.

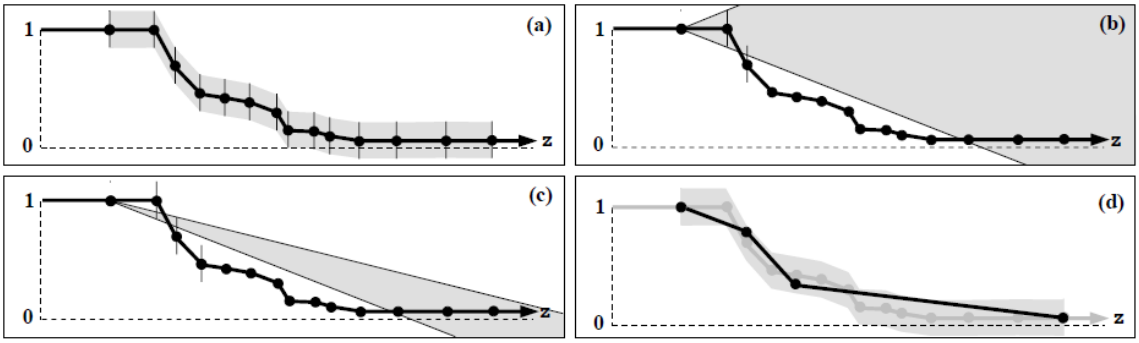


(b) Částečné pokrytí neprůhlednými objekty.



(c) Útlum v objemu.

Obrázek 2.14: Ukázka průběhů funkce viditelnosti pro různé typy objektů. Převzato z [23].



Obrázek 2.15: Proces komprimace funkce viditelnosti. Převzato z [23].

bod (x, y, z) je pak vyhodnocen vztahem:

$$V(x, y, z) = \frac{\sum_{i,j} w_{i,j} V_{i,j}(z)}{\sum_{i,j} w_{i,j}} \tag{2.16}$$

Suma je vytvářena přes všechny pixely (i, j) v rámci dosahu filtru, $w_{i,j} = f(i+\frac{1}{2}-x, j+\frac{1}{2}-y)$ je váha filtru pro pixel (i, j). Vyhodnocení funkcí viditelnosti v hloubce z vyžaduje vyhledávání mezi kontrolními body. V závislosti na počtu bodů je možné vyhledávat sekvenčně nebo binárně. Zlepšení průměrné doby přístupu je možné dosáhnout uložením ukazatele na poslední zpřístupněný segment a v dalším dotazu sekvenčně vyhledávat od tohoto bodu směrem vpřed nebo vzad. Urychlení vyplývá z poznatku, že následující dotazy jsou často prováděny pro blízké hodnoty hloubky.

Mezi hlavní výhody hloubkových stínových map oproti běžným stínovým mapám je podpora předfiltrování. Hodnoty získané filtrováním transmitancí v rámci dosahu filtru

pixelu. Výsledná hodnota v dané hloubce se pak podobá hodnotě ze stínové mapy využívající PCF (percentage closer filtering). Velikost rozdílu závisí na velikosti chyby při kompresi. Počet pořízených vzorků ze scény musí být pro objekty jako vlasy a srst stejně velký jako u běžné stínové mapy, díky předfiltrování a kompresi je však výsledná velikost hluboké stínové mapy mnohem menší při zachování stejné kvality. Filtrování umožňuje také použití mip-mappingu. Každá další úroveň je vytvořena podvzorkováním na poloviční rozlišení a průměrováním. S každou úrovní je snižována maximální povolená chyba komprese, aby se zabránilo příliš velké akumulaci chyby.

Kapitola 3

Návrh aplikace

Cílem práce je navrhnutí metody umožňující zobrazení polostínů průhledných objektů a implementace této metody v jednoduché demonstrační aplikaci. Zobrazování neprůhledných objektů je v dnešním hardwaru dobře zvládnutou oblastí, z-buffer umožňuje jejich efektivní zobrazení v libovolném pořadí. Průhledné objekty však stále vyžadují zvláštní zacházení. Metody vykreslení průhledných objektů můžeme rozdělit na závislé a nezávislé na pořadí vykreslování. Metody závislé na pořadí vyžadují před vykreslením seřazení průhledných objektů od nejvzdálenějšího k nejbližšímu. Řazení však může být problémové pokud se hloubky různých objektů překrývají, nebo je využívána teselace. V této práci bude pro zobrazení průhledných objektů využita metoda nezávislá na pořadí vykreslování popsaná v kapitole 2.3, Dept peeling. Alternativní metodou k Depth Peelingu může být Stochastická průhlednost, která je přiblížena v kapitole 2.4. Výhodou oproti Depth Peelingu je pevně daný počet vykreslovacích průchodů a vyšší rychlost, nevýhodou je nižší kvalita.

Stíny průhledných objektů budou realizovány pomocí hluboké stínové mapy. Podpora bodových zdrojů světla bude zajištěna pomocí vytvoření stínové mapy ve všech šesti pohledech ze světelného zdroje pomocí Cube Mapy. Hluboká stínová mapa bude vytvářena pomocí Depth Peelingu, který umožní stanovení hodnot viditelnostní funkce v hloubkách odpovídajících jednotlivým průhledným objektům. Počet vykreslovacích průchodů Depth Peelingu bude omezen pevně stanovenou hodnotou. Omezení urychlí vykreslování a na kvalitu stínů bude mít jen minimální vliv, protože hodnota viditelnostní funkce ve vzdálených vrstvách bude nízká a již se nebude příliš měnit. Hodnoty viditelnostní funkce závisí na předchozích hodnotách, kde počáteční hodnota je rovna jedné. Z tohoto důvodu nebude pro vytváření stínové mapy použita varianta Dual Depth Peeling, ale varianta postupující pouze v jednom směru. Při vytváření stínové mapy se nabízí možnost využití vykreslování do více cílů framebufferu a ve stejném vykreslovacím průchodu tak získat informace o scéně použitelné v dalších metodách. Lze tak například vytvořit Reflective Shadow Mapu, která bude sloužit pro výpočet nepřímého osvětlení nebo jako základ pro metodu Light Propagation Volumes.

Kapitola 4

Implementace

Aplikace je vytvořena v jazyce C++ s využitím frameworku Qt, který poskytuje běžné widgety pro tvorbu uživatelského rozhraní. Uživatelské rozhraní poskytuje uživateli některé údaje o vykreslování a umožňuje zvolení metody vykreslování a nastavení jejích parametrů. Vykreslování scény je realizováno pomocí OpenGL. Qt framework obsahuje komponenty pro propojení s OpenGL. Komponenta `QGLWidget` umožňuje vykreslování OpenGL grafiky pomocí volání běžných OpenGL funkcí a lze ji vložit do uživatelského rozhraní jako běžný widget. Tato komponenta obsahuje tři virtuální funkce určené pro implementování. Funkce `initializeGL` slouží k vytvoření a nastavení potřebných objektů pro vykreslování, je volána pouze jednou před začátkem vykreslování. Druhá funkce je volána při změně rozměrů komponenty (včetně prvního zobrazení), slouží k nastavení viewportu a nazývá se `resizeGL`. Jejím argumentem je nový rozměr komponenty. Třetí funkcí je `paintGL`, volá se pokud je potřebné překreslení komponenty a obsahuje příkazy pro vykreslení scény.

Qt framework poskytuje třídy `QOpenGLContext` a `QOpenGLFunctions`, které dokáží určit vstupní body OpenGL funkcí bez nutnosti použití externích nástrojů jako GLEW nebo GLee.

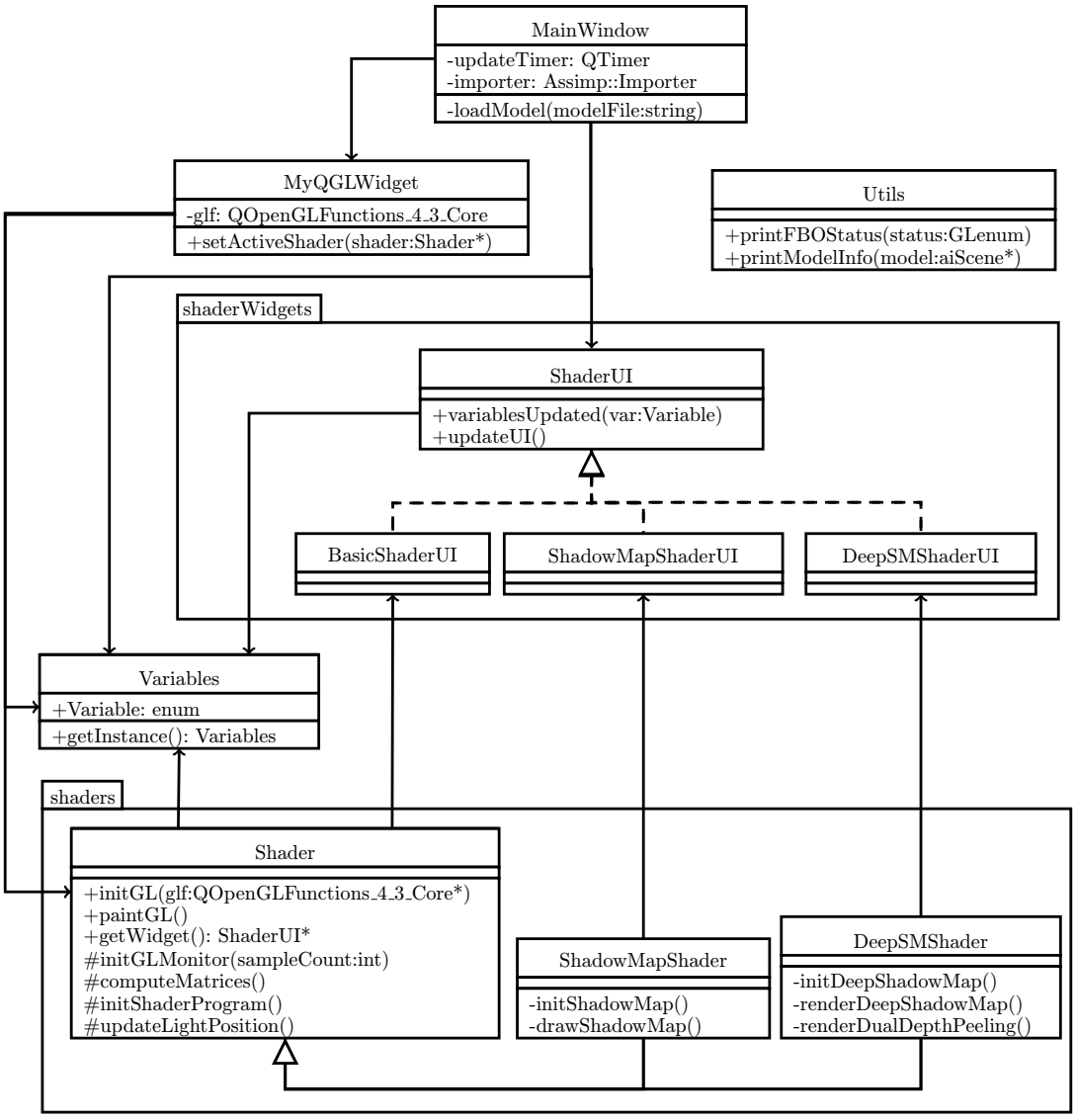
Mezi další podpůrné třídy patří například `QOpenGLShaderProgram`, umožňující snadné načítání a sestavování shader programů. Podporovány jsou i nové typy shaderů jako Compute shader a Tesselation shadery. Třída `QOpenGLTimeMonitor` představuje nadstavbu nad OpenGL dotazy (Query object) pro získávání časových intervalů mezi požadovanými body vykreslování. Lze tak snadno změřit dobu vykonávání různých částí vykreslování, například vytváření stínové mapy. Třída `QOpenGLDebugLogger` umožňuje snadné sledování chyb, varování a informačních hlášení z OpenGL. Není tak nutné provádět kontrolu chyb po každém volání funkce z OpenGL API. Pro funkci této třídy musí být dostupné rozšíření `GL_KHR_debug`. Hlášení mohou být logovány do souboru, nebo zasílány posluchačům.

Součástí aplikace je knihovna ASSIMP (Open Asset Import Library) zajišťující načítání modelů ze souborů různých formátů. Knihovna kromě samotného načítání modelů obsahuje prostředky pro jejich dodatečné zpracování. Při načítání modelu je tak možné například generovat normály, spojovat identické vrcholy, převést model pouze na trojúhelníky, změnit orientaci trojúhelníků nebo detekovat a sloučit duplicitní specifikace materiálů. Knihovna zajišťuje také korektní odstranění modelu z paměti, z tohoto důvodu je načtený model svázán s instancí třídy Importer a tuto instanci je tak nutné udržovat i po načtení modelu. Při ladění aplikace bylo zjištěno, že knihovna není příliš robustní a nezvládne načíst některé modely, i přesto že je jejich formát podporován (.obj). Vzhledem k tomu, že pro demonstraci implementovaných metod postačí načtení jen některých modelů a využití jiné knihovny by zabralo další čas, byla v aplikaci tato knihovna ponechána.

4.1 Struktura aplikace

Struktura aplikace je znázorněna na obrázku 4.1 diagramem tříd. Vstupním bodem aplikace je třída `MainWindow`, která dědí z Qt třídy reprezentující okno aplikace `QMainWindow`. Tato třída zajišťuje inicializaci uživatelského rozhraní. Dále obsahuje časovač, který spouští po 20 milisekundách překreslení scény a po 200 milisekundách aktualizaci hodnot zobrazovaných v uživatelském rozhraní. Součástí třídy je také Importer knihovny ASSIMP a funkce pro načtení modelu ze zadaného souboru. Dále jsou v této třídě zpracovány události kláves.

Třída `MyQGLWidget` rozšiřuje třídu `QGLWidget` a implementuje její metody pro vykreslování scény. Inicializuje OpenGL kontext a předává řízení vykreslování instanci třídy reprezentující konkrétní metodu vykreslování. Dále jsou zpracovány události myši pro navigaci ve scéně.



Obrázek 4.1: Diagram tříd.

Všechny parametry pro vykreslování i informace pro uživatele jsou obsaženy ve třídě `Variables`. Tato třída je vytvořena podle návrhového vzoru jedináček (singleton) a všechny

ostatní třídy k ní tak mají snadný přístup pomocí funkce `getInstance`. K dispozici je signál `variablesUpdated`, který je zaslán při změně některé hodnoty obsažené v této třídě. Hodnotu, která byla změněna, lze rozpoznat podle argumentu, který nabývá některou z hodnot enumerace `Variable`. Veškerá komunikace mezi uživatelským rozhraním a vykreslovacími metodami probíhá skrze tuto třídu.

Všechny konkrétní metody vykreslování jsou obsaženy v balíku `shaders`. Základní třída určující rozhraní pro všechny ostatní metody vykreslování a implementující některé základní funkce je `Shader`. Kromě základních funkcí pro inicializaci a vykreslování deklaruje funkci `getWidget`, která umožňuje získat widget obsahující uživatelské rozhraní specifické pro danou metodu vykreslování. Tato třída obsahuje funkci pro inicializaci `QOpenGLDebugLoggeru` a jeho posluchače, inicializaci `QOpenGLTimeMonitoru`, načtení modelu do GPU, výpočet transformačních matic pro kameru a funkci pro animaci světla. Metoda vykreslování této třídy spočívá v zobrazení načteného modelu a objektu světla s Phongovým lokálním osvětlovacím modelem. Rozšiřující třída `ShadowMapShader` poskytuje vykreslování modelu se stíny vytvořenými pomocí stínové mapy. Další rozšiřující třídou je `DeepSMShader`, poskytuje vykreslování modelu včetně stínů průhledných objektů. Stíny jsou realizovány pomocí hluboké stínové mapy.

Balík `shaderWidgets` obsahuje třídy reprezentující widgety pro jednotlivé typy vykreslovacích metod. Společné rozhraní je určeno třídou `ShaderUI`, jejíž funkce `updateUI` slouží k vyžádání aktualizace údajů zobrazovaných daným widgetem. Funkce `variablesUpdated` pak danému widgetu poskytuje možnost reagovat na změnu vybraných hodnot ve třídě `Variables`. Toto rozhraní umožňuje snadnou výměnu ovládacího panelu aplikace při změně vykreslovací metody. Implementující třídy zajišťují zpracování událostí a nastavení získaných hodnot do třídy `Variables`, odkud si je mohou přebrat vykreslovací metody.

Třída `Utils` sdružuje pomocné funkce, mezi které patří výpis stavu framebufferu a výpis informací o načteném modelu. Dále obsahuje funkce pro přednastavení pohledu a parametrů animace světla pro různé modely.

4.2 Implementace vykreslovacích metod

Všechny implementované metody využívají pro vykreslování Vertex Array Objects (VAO), které jsou součástí specifikace OpenGL od verze 3.0. VAO udržují nastavení atributů vertexů a odkaz na Vertex Buffer Object, atributy vertexů tak stačí nastavit pouze jednou. Přepínání VAO je snadnější a rychlejší než nastavování atributů při každém vykreslovacím průchodu.

Další využívanou funkcionalitou jsou subrutiny v shader programech. Fungují podobně jako ukazatele na funkce v jazyce C. Umožňují tvorbu přehlednějších shaderů, než při využívání větvení pomocí if-else příkazů. Přepnutí subrutiny je také rychlejší než změna aktivního shader programu. Aplikace využívá tři typy subrutin, `computeColor` se stará o získání barvy fragmentu (z textury, barva materiálu, nebo barva zadaná pomocí uniform hodnoty pro vykreslení objektu světla a některých průhledných oken), `computeLight` zajišťuje výpočet osvětlení (ambientní, Phongův model) a `computeShadow` slouží k nastavení výpočtu stínů (bez stínů, monochromatické stíny, barevné stíny). Nastavením subrutin je možné snadno obsáhnout všechny kombinace možností vykreslování. Identifikace subrutin je realizována pomocí explicitního nastavení jejich indexů, ušetří se tak volání funkce pro zjišťování indexu subrutiny podle jména. Tento přístup však vyžaduje minimální verzi GLSL 4.30.

Metoda běžné stínové mapy je realizována pomocí Cube Map textury pro uložení hloubek z pohledu světla. Z důvodu zachování co nejvyššího rozlišení hloubek je nastavena zadní ořezová rovina na co nejnižší hodnotu tak, aby ukázkové modely byly zobrazeny bez oříznutí.

Hodnotu zadní ořezové roviny je možné změnit v souboru `shadowmapshader.cpp` pomocí konstanty `FAR_PLANE_SHADOW`. Vykreslení stínové mapy lze provést postupně po jednotlivých stěnách Cube Mapy. Druhou možností je provedení vykreslení v jediném průchodu pomocí geometry shaderu, kde se určí cílová stěna Cube Mapy pomocí hodnoty `gl_Layer`. Porovnání rychlosti těchto přístupů je popsáno v kapitole **??**. Porovnání hloubky vykreslovaného fragmentu s hloubkou obsaženou ve stínové mapě je provedeno pomocí `samplerCubeShadow` při čtení z textury, které vrací namísto hloubky rovnou výsledek porovnání. Textura obsahující hloubky musí mít nastaven režim porovnávání `GL_TEXTURE_COMPARE_MODE` na hodnotu `GL_COMPARE_REF_TO_TEXTURE`). Funkci provádějící porovnání je možné nastavit parametrem `GL_TEXTURE_COMPARE_FUNC`.

Metoda hluboké stínové mapy sestává ze dvou hlavních fází. První fází je vytvoření hluboké stínové mapy pomocí Depth Peelingu. Vytváření probíhá pomocí vykreslování do framebufferu z pohledu světla s depth attachmentem a jedním color attachmentem. Depth attachment naplňuje texturu, která je využívána pro odstraňování jednotlivých vrstev na základě hloubky metodou Depth Peelingu. Color attachment naplňuje texturu reprezentující hlubokou stínovou mapu. První složkou této textury je hloubka. V případě monochromatických stínů obsahuje textura v druhé složce hodnotu viditelnostní funkce odpovídající dané hloubce. V případě barevných stínů obsahuje druhá až čtvrtá složka textury hodnoty viditelnostní funkce pro jednotlivé barevné kanály. Hodnoty viditelnostní funkce jsou vypočítávány z předchozích hodnot této funkce (počáteční hodnota se rovná jedné). Z důvodů read/write hazardů je hluboká stínova mapa rozdělena do dvou textur, kde jedna obsahuje sudé vrstvy a druhá liché. Tyto textury si v každém průchodu Depth Peelingu mění role, jedna slouží pro čtení předchozích hodnot viditelnostní funkce, druhá pro zápis viditelnostní funkce pro aktuální vrstvu hloubek. Stejným způsobem se prohazují dvě textury depth attachmentu.

Textury uchovávající hloubku pro Dept Peeling i textury reprezentující hlubokou stínovou mapu jsou Cube Mapy. Jelikož je hluboká stínová mapa vrstvená textura, musí být vrstvená i textura obsahující hloubky, jelikož framebuffer umožňuje použít pouze stejný typ textur (z hlediska vrstvení). Textura hloubky je proto vytvořena jako vrstvená, i když obsahuje pouze jednu vrstvu. Připojování textur k framebufferu je realizováno funkcí `glFramebufferTextureLayer`, tato funkce však pracuje se stěnami Cube Mapy jako s nezávislými vrstvami, proto není možné připojit k jednomu attachmentu framebufferu všech šest stěn dané vrstvy Cube Mapy. Vykreslování tak musí probíhat postupně pro jednotlivé stěny Cube Mapy a nelze tak využít geometry shader pro vykreslení všech šesti stěn v jednom průchodu, jako tomu bylo u metody běžné stínové mapy.

Druhou fází je zobrazení scény z pohledu kamery s využitím Dual Depth Peelingu pro správné zobrazení průhledných objektů a hluboké stínové mapy pro určení zastínění. Stejně jako při vytváření hluboké stínové mapy jsou využívány dvojice textur kvůli read/write hazardům. Počet průchodů Dual Depth Peelingu jednotlivými vrstvami hloubek scény není omezen jako při vytváření hluboké stínové mapy. Metoda využívala OpenGL dotazu (query) na počet vykreslených fragmentů, a průchod vrstvami hloubky končil až ve chvíli, kdy nebylo co vykreslit. Tento přístup byl následně nahrazen použitím atomického čítače, který byl inkrementován v shaderu zajišťujícím Dual Depth Peeling v případě, že byl zpracován průhledný fragment (alfa menší než 1). V případě, že čítač zůstal s nulovou hodnotou, je možné vykreslování ukončit, protože všechny vykreslované fragmenty v aktuální vrstvě byly neprůhledné a fragmenty z následujících vrstev by tak stejně nebyly viditelné. Toto vylepšení pomohlo snížit počet prováděných průchodů Dual Depth Peelingu a zvýšit rychlost vykreslování v případech, kdy scéna obsahuje neprůhledné objekty s vyšším počtem

vrstev hloubky (například model sponzy).

Atomické čítače jsou však specifické pro svůj atomický přístup, inkrementace je tudíž blokující operací, což zpomaluje paralelní zpracování fragmentů v shaderech. Pro ukončení vykreslování je podstatnou informací pouze to, zda byl vykreslen nějaký průhledný fragment. Zápis této informace však nevyžaduje výlučný přístup. Atomický čítač byl proto nahrazen Shader Storage Bufferem, který umožňuje neblokující přístup, což umožňuje rychlejší zpracování v shaderech. Tento přístup byl využit i při vytváření hluboké stínové mapy. Pokud jsou všechny fragmenty dané vrstvy neprůhledné, není třeba pokračovat do hlubších vrstev, protože viditelnostní funkce již má nulovou hodnotu. Tím lze dosáhnout menšího počtu vykreslovacích průchodů a rychlejší tvorby hluboké stínové mapy ve většině scén.

4.3 Ovládání aplikace

Demonstrační aplikace obsahuje v horní části jednoduché menu. Položka *Scene* obsahuje možnosti pro načtení modelu a ukončení aplikace. Položka *View* umožňuje skrýt nebo zobrazit kontrolní panel. Kontrolní panel se nachází v pravé části aplikace, zobrazuje počet snímků, který zvolená metoda dokáže vykreslit za sekundu. Dále umožňuje výběr vykreslovací metody. Zbývající položky kontrolního panelu jsou závislé na zvolené vykreslovací metodě. Navigace ve scéně se provádí pomocí myši v panelu zabrazujícím scénu. Tahem při stisknutém levém tlačítku se provádí otáčení kamery ve směru pohybu myši. Pohyb ve směru pohledu je proveden při stisknutém pravém tlačítku a pohybem myši vpřed. Pohyb vzad provede pohyb v protisměru pohledu. Aplikace obsahuje několik klávesových zkratek. Pro funkci těchto zkratek je nutné mít focus na panelu scény (provede se kliknutím). Klávesa *L* umožňuje skrýt nebo zobrazit objekt reprezentující světlo. Klávesy *Q* a *W* provádí pohyb světla v kladném a záporném směru osy x, *A* a *S* pro osu y, *Y* a *X* pro osu z. Zvětšení popřípadě zmenšení načteného modelu je možné provést klávesami + a -. Pro vykreslovací metodu *DeepShadowMap shader* je možné zobrazit obsah hluboké stínové mapy pomocí klávesy *D*. Stěny se přepínají klávesami *F1* až *F6*, vrstvy klávesami *1* až *9*.

Kapitola 5

Vyhodnocení výsledků

Tato kapitola je rozdělena na tři podčásti. V první části jsou popsány prostředky, které jsou použity v demonstrační aplikaci a stroje, na kterých byla měřena rychlost vykreslování. Druhá část obsahuje ukázku vizuálních výstupů aplikace. V poslední části jsou porovnány rychlosti vykreslování různých variant implementace.

5.1 Použité prostředky

Vyhodnocování výsledků probíhalo na dvou počítačích s následujícími specifikacemi:

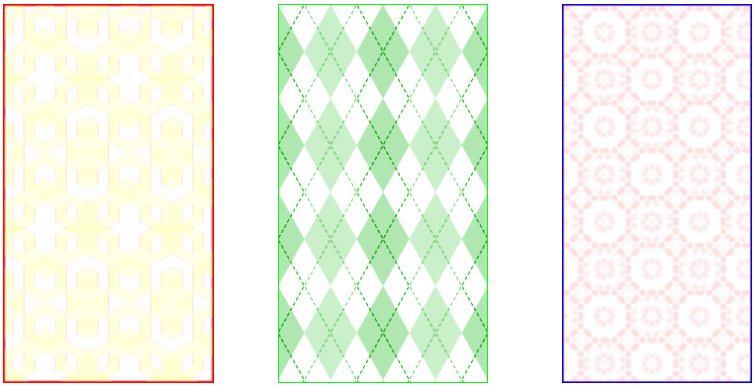
1. GeForce GT 525M, Intel Core i5-2410M 2.3 GHz, 6 GB RAM,
   Windows 7 Professional x64.
2. AMD Radeon HD 8790M, Intel Core i7-4800MQ 2.7 GHz, 8 GB RAM,
   Windows 7 Professional x64.

Použité modely pochází z webových stránek [24]. V následující tabulce je uveden seznam modelů s některými jejich údaji:

Název	Autor	Počet trojúhelníků	Počet vrcholů
Cube	-	12	24
Dabrovic Sponza	Marko Dabrovic	66 450	59 810
Sibenik Cathedral	Marko Dabrovic	75 284	83 490
Conference Room	Anat Grynberg a Greg Ward	331 179	216 862

Tabulka 5.1: Seznam modelů použitých v aplikaci. Modely pochází z [24].

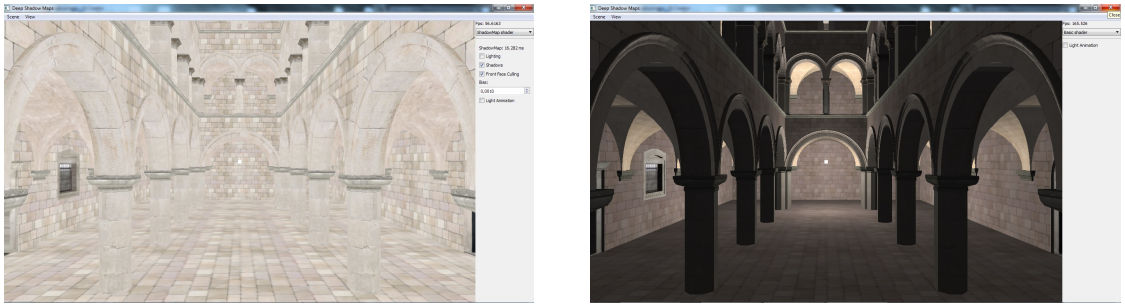
Dále byly použity pro demonstraci hlubokých stínových map textury pro průhledná okna, které je možné vidět na obrázku 5.1. Vzory byly převzaty z webových stránek [25] a upraveny pro potřeby aplikace. Průhledná okna byla špatně viditelná, proto byl k texturám přidán barevný rámeček. Textury byly také obarveny pro demonstraci barevných stínů.



Obrázek 5.1: Vzory použitých textur.

5.2 Vyhodnocení výstupů aplikace

Na obrázku 5.2 můžeme vidět výstupy s modelem Dabrovic Sponza a porovnání ambientního osvětlení s Phongovým osvětlovacím modelem. Knihovna pro načítání modelů umožňuje použít post processing pro vyhlazení normál. Porovnání originálních a vyhlazených normál je možné vidět na obrázku 5.3. Vyhlazené normály vypadají v některých oblastech lépe, což je viditelné například na sloupech. Normály jsou však upraveny i v místech, kde to není vhodné, což způsobuje vznik artefaktů. Takovým artefaktem je například zaoblení začátku klenby na bočních zdech. Z důvodu vzniku těcho artefaktů nebylo vyhlazování normál dále v aplikaci používáno.



(a) Ambientní osvětlení.

(b) Osvětlení Phongovým modelem.

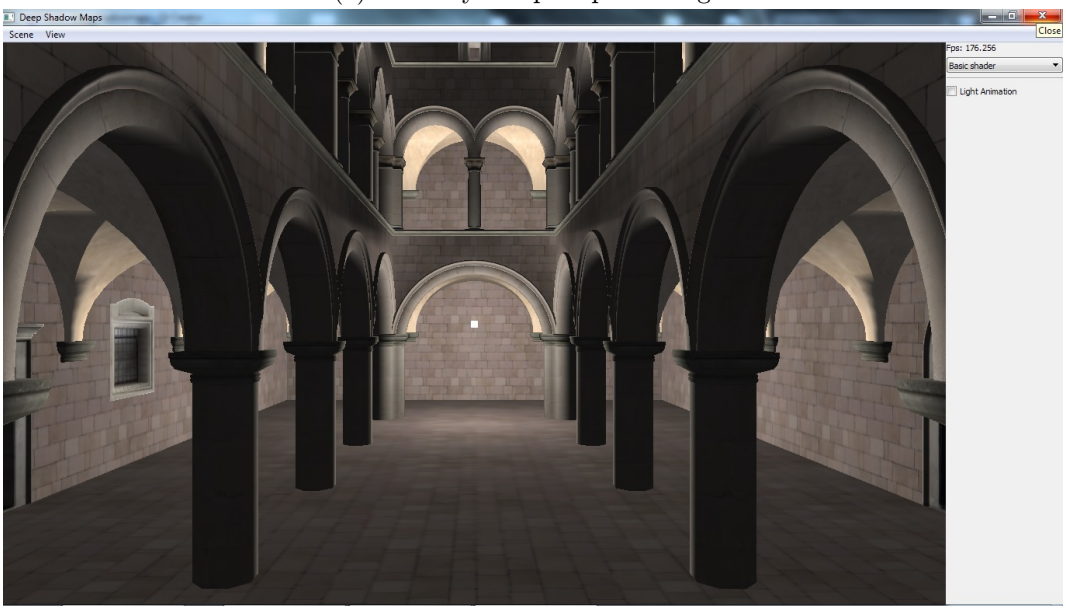
Obrázek 5.2: Porovnání lokálních osvětlovacích modelů.

Na obrázku 5.4 jsou prezentovány výstupy metody běžné stínové mapy. Problémem stínových map jsou nepřesnosti při převodu hloubky fragmentu z prostoru kamery do prostoru světla, což způsobuje sebezastínění (obrázek 5.4a) v místech, kde nepřesnost způsobí změnu vyhodnocení podmínky zastínění. Při porovnávání se proto běžně používá mírný posun hodnot (bias). Posun dokáže zabránit sebezastínění (obrázek 5.4b), ale způsobuje posunutí stínů. Posunutí se zvyšuje s rostoucí vzdáleností od světla a je zvláště patrné u tenkých objektů vrhajících stín. Alternativní možností řešení sebezastínění je využití face cullingu při vytváření stínové mapy. Face culling se nastaví na stěny přivrácené ke světlu a stíny jsou tím pádem vrhány stranami odvrácenými. Rozdíl hloubek je dostatečně velký, aby zabránil sebezastínění ploch přivrácených ke světlu. Sebezastínění však nastává na odvrácených stěnách vrhajících stín (obrázek 5.4c), není však tolik patrné, jako na osvětlených stěnách. Částečným řešením může být záporný posun, který odstraní sebezastínění u odvrá-

cených stěn (obrázek 5.4d). Může však opět nastat problém s tenkými objekty vrhajícími stín.

Aplikace podporuje výpočet viditelnostní funke pro monochromatické stíny (jedna viditelnostní funkce pro všechny barevné kanály) i barevné stíny (tři viditelnostní funkce, pro každý barevný kanál jedna). Výpočet probíhá stejným způsobem pro libovolný zdroj barvy fragmentu. Kromě objektů se zadanou průhlednou barvou lze použít i objekty s průhlednou texturou, stačí změnit volanou subrutinu pro získání barvy fragmentu. Porovnání monochromatických a barevných stínů pro objekty s homogenní barvou a objekty s texturou je možné vidět na obrázku 5.5.

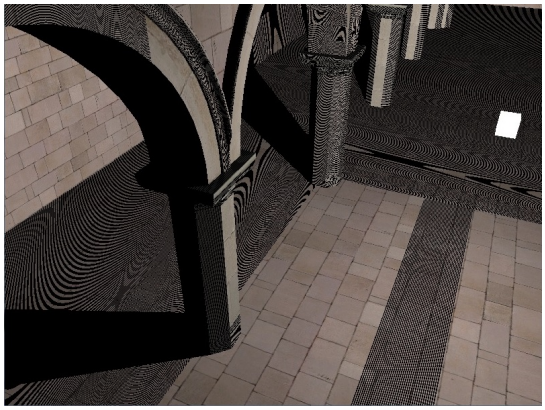
(a) Normály bez post processingu.



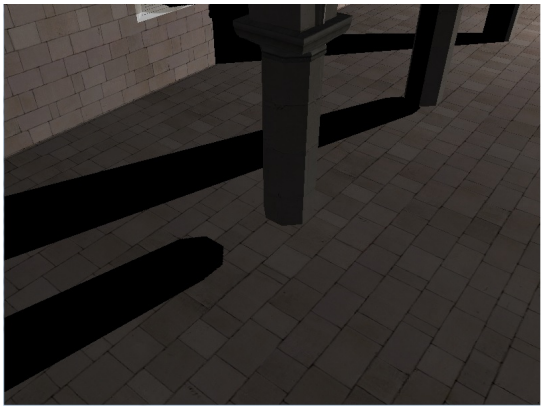
(b) Vyhlazené normály.

Obrázek 5.3: Porovnání neupravených a vyhlazených normál.

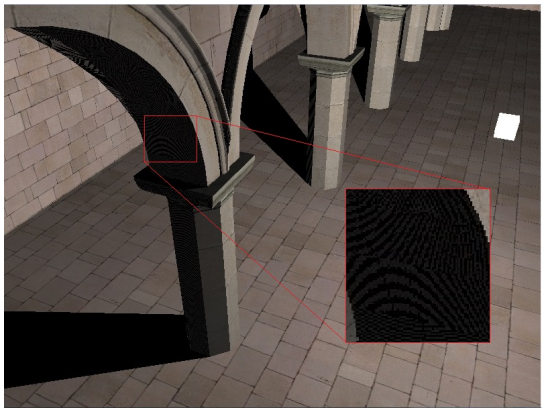
Vztah pro výpočet viditelnostní funkce pro monochromatické a barevné stíny se liší. Výpočet monochormatických stínů nebere v úvahu barvu objektu, ale pouze jeho průhlednost. Výpočet je popsán vztahem 5.1, kde dolní index udává kanál, c reprezentuje barvu fragmentu a V' hodnotu viditelnostní funkce z předchozího kroku (na počátku je 1). Při výpočtu viditelnostní funkce je nutné do vztahu navíc zahrnout barvu průhledného objektu. Na vztah jsou také kladena určitá omezení, pro hodnotu alfa blížící se k jedné se musí barva stínu nezávisle na barvě objektu blížit k černé. Pro hodnotu alfa blížící se k nule musí klesat vliv barvy objektu a pro hodnotu rovnou nule objekt nesmí vrhat stín. Kvůli tomuto omezení není možné použít vztah s pouhým vynásobením monochromatické viditelnostní funkce barvou objektu. Naprosto průhledný (neviditelný) objekt v tomto případě tvoří stín s barvou objektu, což vypadá nerealisticky.



(a) Bez biasu a bez face cullingu.



(b) Bias, bez face cullingu.



(c) Bez biasu, s face cullingem.

(d) Face culling se záporným biasem.

Obrázek 5.4: Porovnání variant běžné stínové mapy.

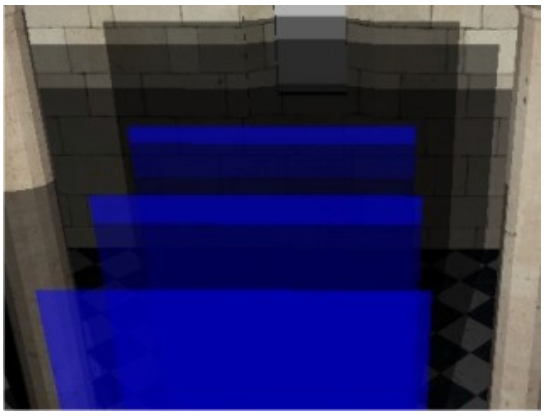
$$V_r = V'_r(1 - c_a) \tag{5.1}$$

$$V_{rgb} = V'_{rgb}(1.0 - c_{aaa})^{(2.0 - c_{rgb})} \tag{5.2}$$

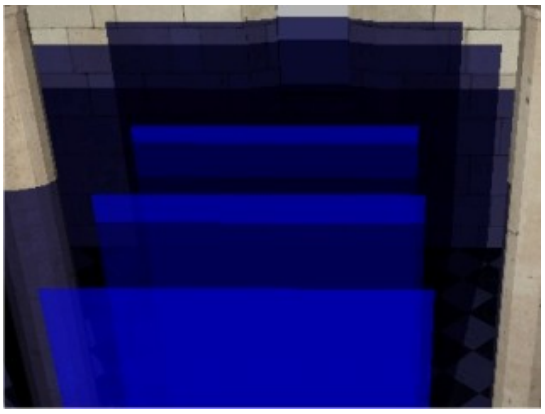
$$V_{rgb} = V'_{rgb}(1.0 - c_{aaa})\text{clamp}_{0-1}(1.0 - c_{aaa} + c_{rgb}) \tag{5.3}$$

Experimentálně byly sestaveny rovnice 5.2 a 5.3, které splňují předem popsaná omezení (obrázek 5.6a a 5.6b). Rovnice byly dále porovnány pomocí výpočtu a porovnání hodnot pro všechny kombinace hodnot alfy a barvy v rozsahu od 0 do 1 s krokem 0,1. Rozdíl výsledků

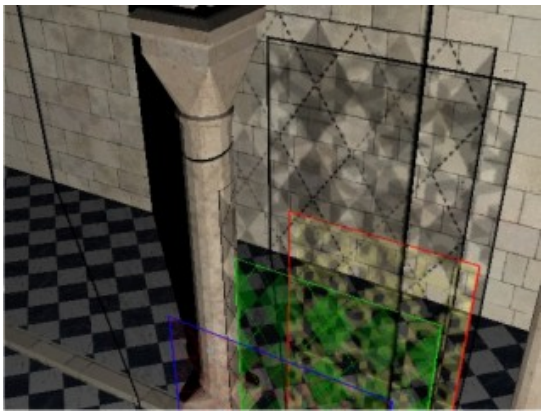
je nulový pro okrajové hodnoty alfy a je malý i pro okrajové hodnoty barev nezávisle na hodnotě alfy. Největší rozdíl nastává v případě hodnoty 0,4 pro alfu i barvu. Rozdíl barev lze porovnat na obrázkách 5.6c (výsledek rovnice 5.2) a 5.6d (výsledek rovnice 5.3). Z vypočítaných hodnot vyplývá, že rovnice 5.3 poskytuje sytější barvy stínů, než rovnice 5.2. Z hlediska rychlosti vykreslování byla volba rovnice zanedbatelná, i když umocňování v rovnici 5.2 je časově náročnější operace než sčítání a násobení.



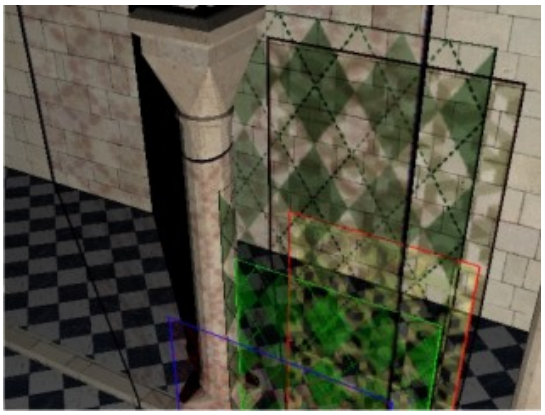
(a) Monochromatický stín oken s homogenní barvou.



(b) Barevný stín oken s homogenní barvou.

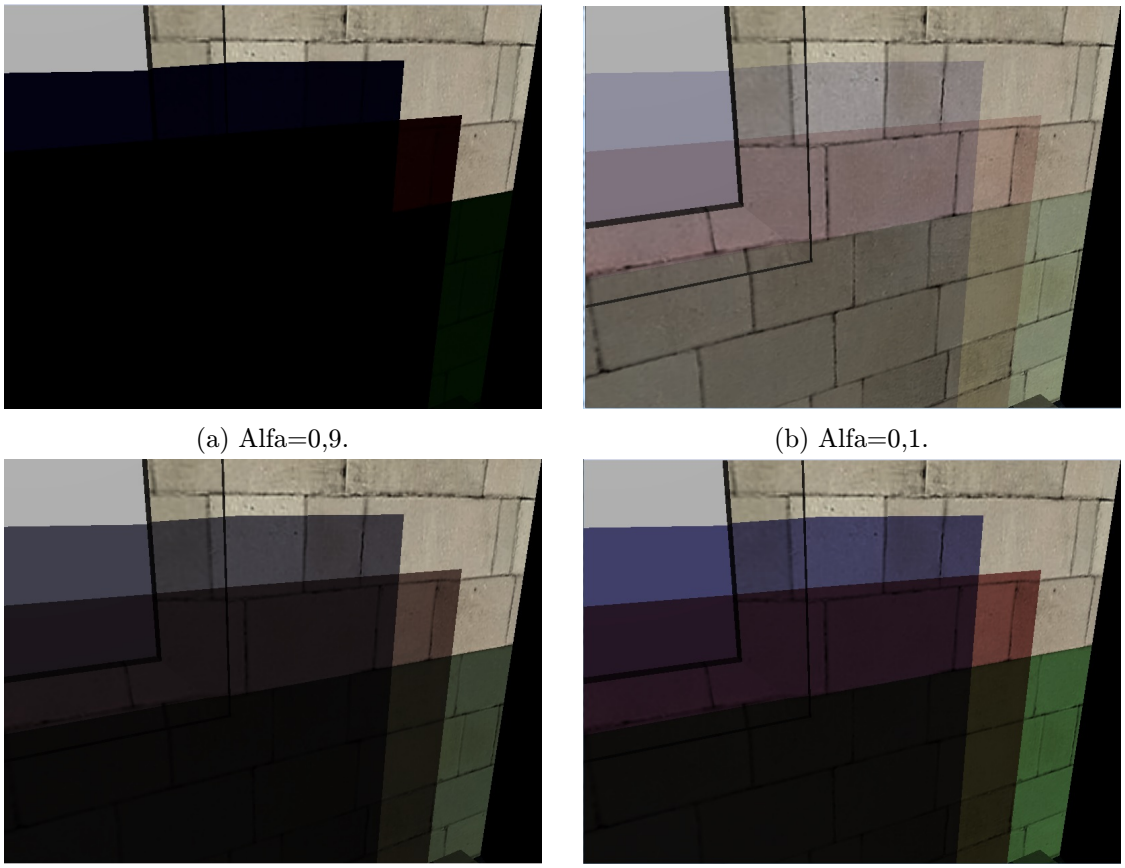


(c) Monochromatický stín oken s texturou.



(d) Barevný stín oken s texturou barvou.

Obrázek 5.5: Porovnání barevných a monochromatických stínů průhledných objektů.



(a) Alfa=0,9.

(b) Alfa=0,1.

(c) Alfa=0,4. Výpočet podle 5.2.

(d) Alfa=0,4. Výpočet podle 5.3.

Obrázek 5.6: Porovnání výpočtu viditelnostní funkce pro barevné stíny.

5.3 Vyhodnocení rychlosti aplikace

Na sestavě s kartou Radeon se z neznámých důvodů nepodařilo načíst model Dabrovic Sponza, proto v grafech měřených na této sestavě chybí tento model. Veškeré měření probíhalo při rozlišení viewportu 800x600. Hodnoty byly měřeny pomocí OpenGL Timer Queries prostřednictvím QOpenGLTimeMonitor objektu.

V kapitole 4.2 byly zmíněny možné implementace pro vytvoření stínové mapy. V následujících grafech jsou porovnány doby vytváření stínové mapy pro jednotlivé varianty implementace. První varianta pracuje bez využití geometry shaderu (v grafech označeno jako Bez GS), stěny cube mapy jsou vykreslovány pomocí cyklu prováděného na CPU. Druhá varianta využívá geometry shader s jednou invokací a cyklem, který postupně přepíná stěny cube mapy (algoritmus 5.1). Třetí varianta využívá geometry shader bez cyklu s šesti invokacemi (algoritmus 5.2).

Z grafů 5.1 a 5.2 je zřejmé, že doba vytváření stínové mapy závisí jen minimálně na jejím rozlišení pro všechny tři typy implementace. Doba vykreslování je více ovlivněna počtem trojúhelníků ve scéně, jak je možné vidět v grafech 5.3 a 5.4. Ve všech případech byl geometry shader využívající cyklus rychlejší než varianta využívající invokace. Sestava GeForce vykazovala jako ve většině případů nejrychlejší implementaci bez využití geometry shaderu. Pouze v případě modelu Dabrovic Sponzy byla implementace geometry shaderu s cyklem rychlejší, vzhledem k nízkému rozdílu hodnot se však může jednat o chybu v měření. Na se-

stavě Radeon byla situace opačná, implementace využívající geometry shader s cyklem byla ve většině případů rychlejší, než implementace bez geometry shaderu. To byl očekávaný výsledek, protože s využitím geometry shaderu sníží množství komunikace mezi CPU a GPU.

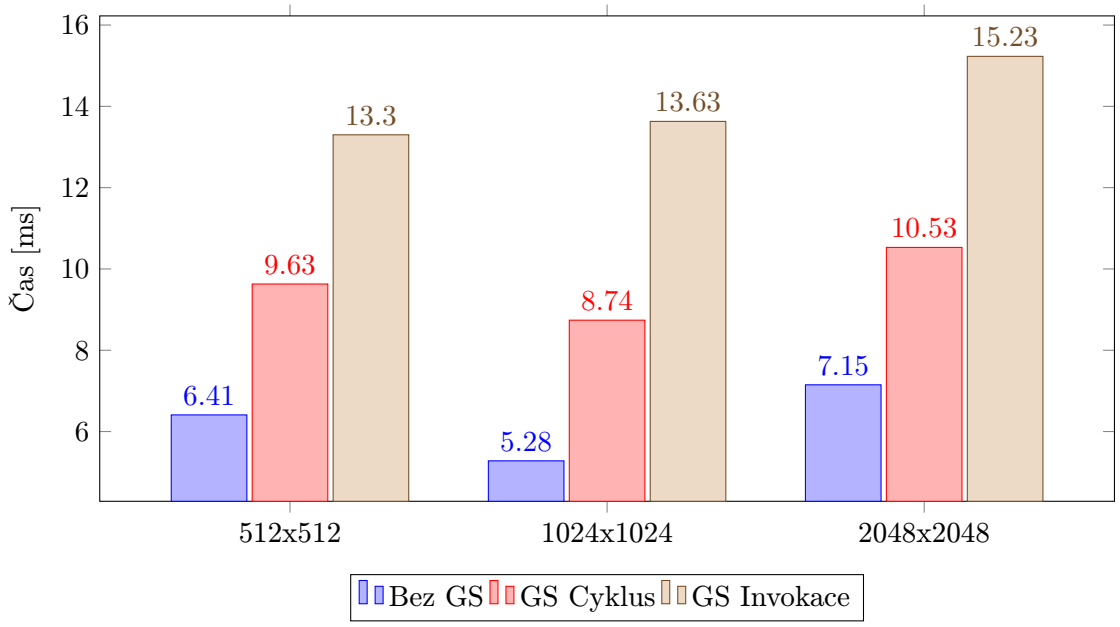
```
layout (invocations = 1, triangles) in;
layout (triangle_strip, max_vertices = 3) out;

uniform mat4 mvp[6];

void main(){
    for(int face = 0;face < 6;face++){
        gl_Layer = face; //nastaveni vrstvy
        for(int i = 0; i < gl_in.length(); i++){
            gl_Position = mvp[face]*gl_in[i].gl_Position;
            EmitVertex();
        }
        EndPrimitive();
    }
}
```

Algoritmus 5.1: Geometry shader využívající cyklus pro přepínání stěn cube mapy.

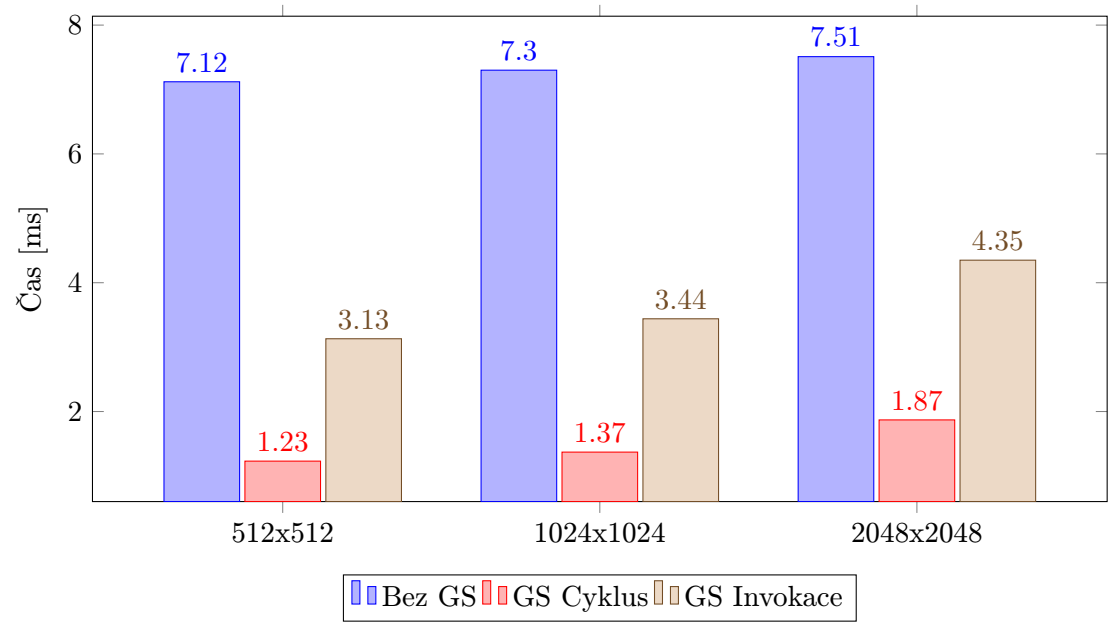
```
layout (invocations = 6, triangles) in;
layout (triangle_strip, max_vertices = 3) out;

uniform mat4 mvp[6];

void main(){
    gl_Layer = gl_InvocationID; //nastaveni vrstvy
    for(int i = 0; i < gl_in.length(); i++){
        gl_Position = mvp[gl_InvocationID]*gl_in[i].gl_Position;
        EmitVertex();
    }
    EndPrimitive();
}
```

Algoritmus 5.2: Geometry shader využívající invokace pro přepínání stěn cube mapy.

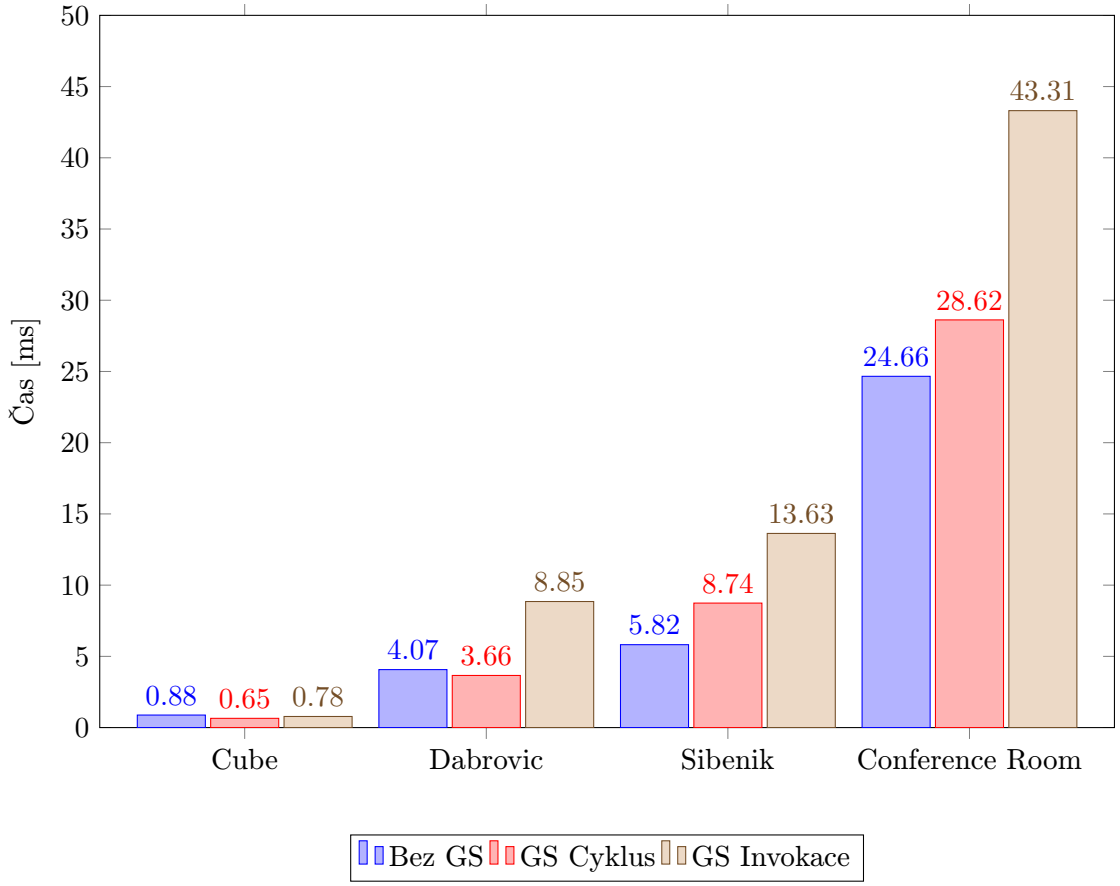
Následující vyhodnocení se zabývá vytvářením hluboké stínové mapy. V modelech byla použita průhledná okna s texturou i homogenní barvou, z pohledu světla vždy maximálně tři za sebou. Rozdíl doby vykreslování mezi monochromatickou a barevnou verzí je zanedbatelný. Hluboká stínová mapa vyžaduje velké množství paměti. Při rozlišení 2048x2048, 4 vrstvách v každé ze šesti stěn cube mapy, 4 složkách pro barevné stíny, z nichž každá je reprezentována 32-bitovým číslem s plovoucí řádovou čárkou je potřebná paměť 1 610 612 736 bytů. Potřebnou paměť lze snadno snížit na polovinu použitím 16-bitové re-



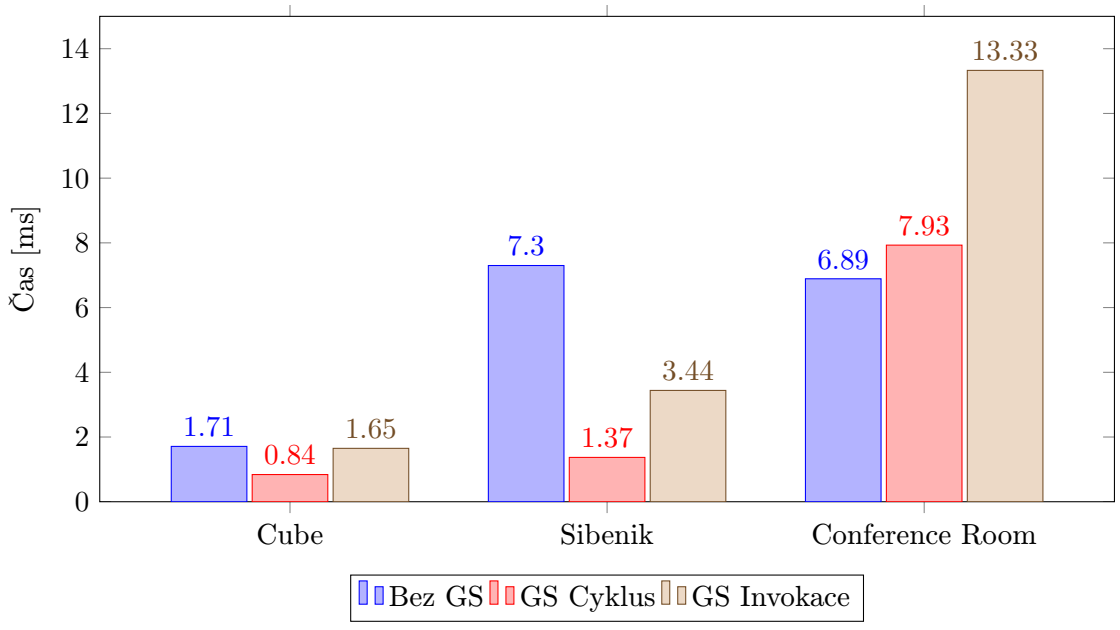
Graf 5.1: Porovnání doby vytváření stínové mapy v závislosti na využití geometry shaderu pro přepínání stěn cube mapy při různých rozlišeních stínové mapy. GeForce, Sibenik Cathedral.



Graf 5.2: Porovnání doby vytváření stínové mapy v závislosti na využití geometry shaderu pro přepínání stěn cube mapy při různých rozlišeních stínové mapy. Radeon, Sibenik Cathedral.



Graf 5.3: Porovnání doby vytváření stínové mapy v závislosti na využití geometry shaderu pro přepínání stěn cube mapy pro různé modely. GeForce, 1024x1024.

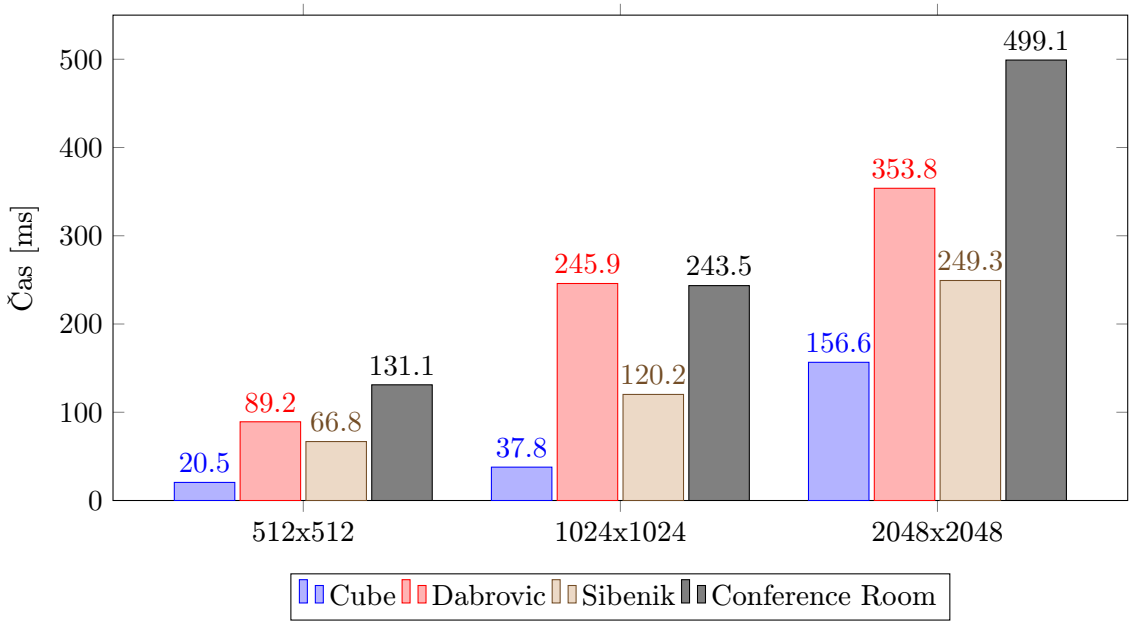


Graf 5.4: Porovnání doby vytváření stínové mapy v závislosti na využití geometry shaderu pro přepínání stěn cube mapy pro různé modely. Radeon, 1024x1024.

prezentace složek hluboké stínové mapy. Reprezentace viditelnostních funkcí pomocí 16 bitů je dostatečná. Problém nastává se složkou obsahující hloubku, kde je potřeba vyšší rozlišovací schopnost hodnot. Pokud to dovoluje charakter scény, postačí jako řešení tohoto problému vyšší hodnota posunu (biasu). Snížení paměťových nároků o další polovinu lze dosáhnout použitím pouze monochromatických stínů, pokud nejsou vyžadovány barevné. Implementace v demonstrační aplikaci používá stejnou velikost hluboké stínové mapy pro monochromatické i barevné stíny, aby bylo možné rychlé přepínání mezi těmito variantami.

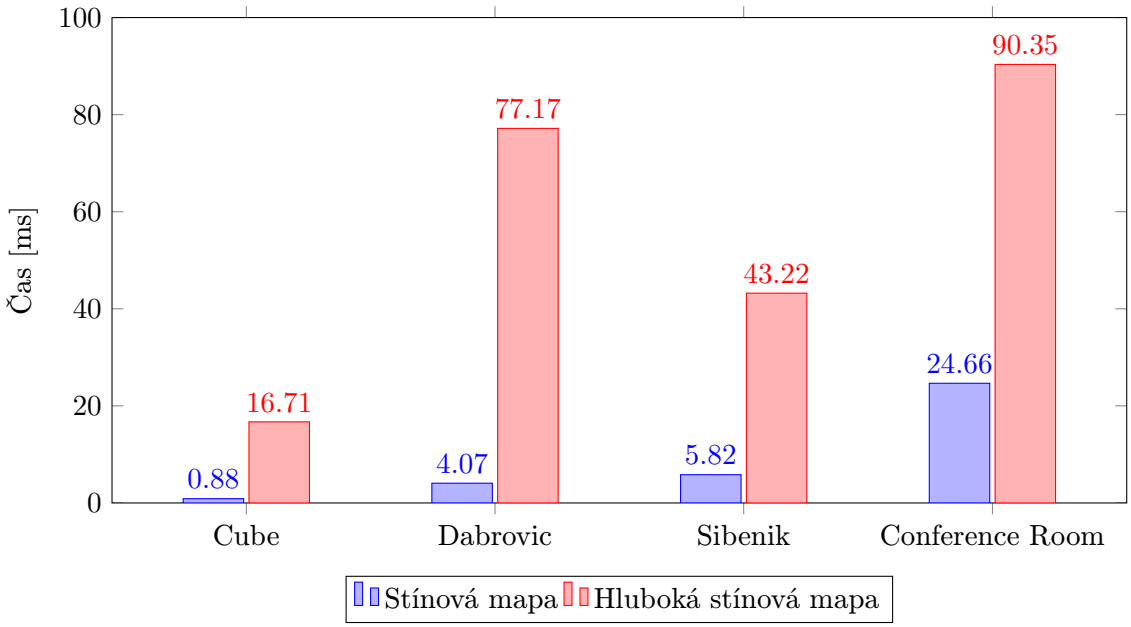
Graf 5.5 znázorňuje závislost doby vytváření hluboké stínové mapy na počtu trojúhelníků a rozlišení. Doba vytváření roste s počtem trojúhelníků. Záleží však i na jejich uspořádání, což je vidět na rychlejším vykreslení modelu Sibenik Cathedral i přesto, že obsahuje větší množství trojúhelníků. Dále je možné porovnat nárůst doby v závislosti na rozlišení. Doba vykreslení běžné stínové mapy se s vyšším rozlišením prodlužovala jen mírně, u hluboké stínové mapy je však nárůst mnohem strmější, a to pro všechny modely.

Graf 5.6 zobrazuje porovnání doby vytváření běžné stínové mapy a hluboké stínové mapy s jednou vrstvou. Vykreslení hluboké stínové mapy trvá podle očekávání několikrát déle. V případě běžné stínové mapy je zapotřebí pouze depth attachment, u hluboké stínové mapy je navíc potřeba color attachment pro uložení viditelnostní funkce. Dalším zdržením je výpočet viditelnostní funkce. Na rozdíl od běžné stínové mapy je potřeba nastavení příslušných uniform hodnot a přepnutí subrutin pro správné předání parametrů materiálu.

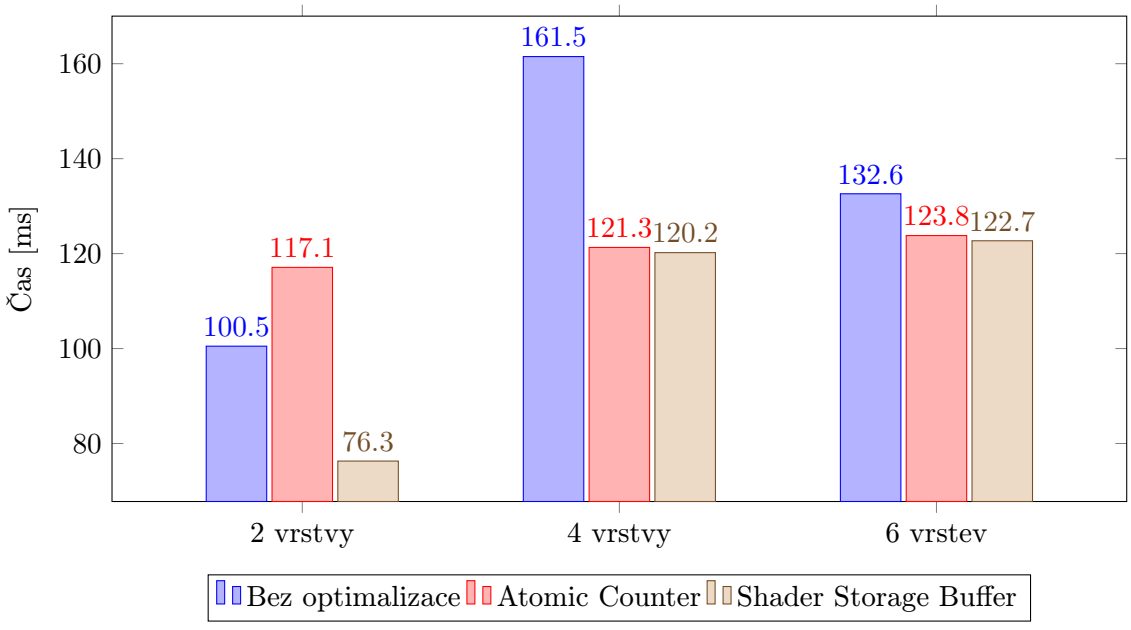


Graf 5.5: Porovnání doby vytváření hluboké stínové mapy v závislosti na rozlišení pro různé modely. GeForce, 4 vrstvy, SSB.

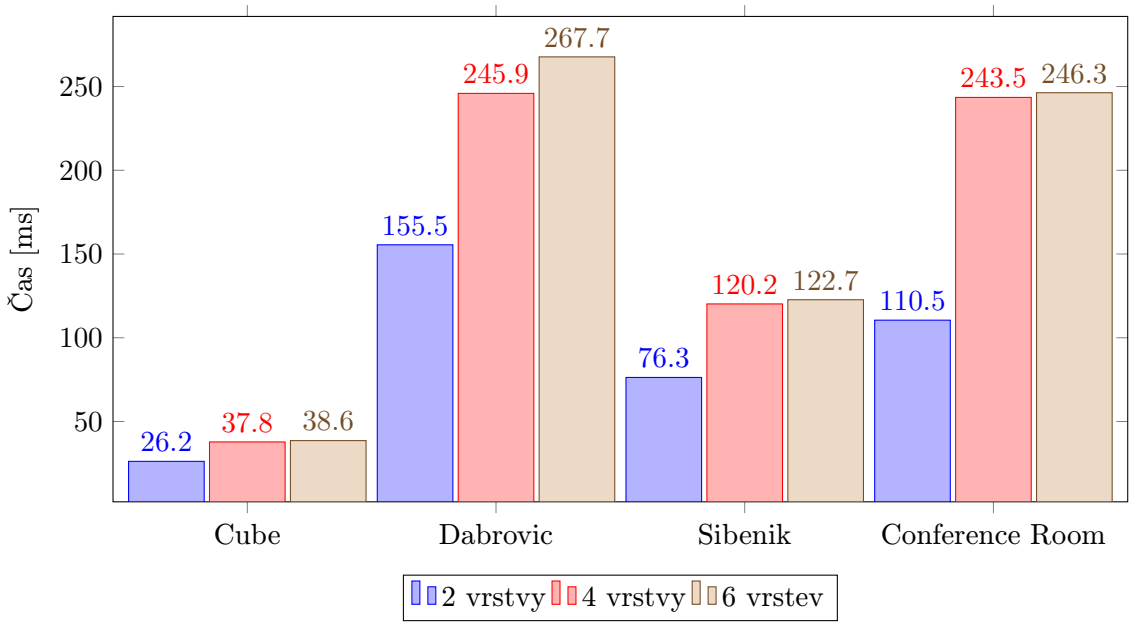
Vytváření hluboké stínové mapy bylo vylepšeno pomocí dvou variant optimalizace vykreslování. Optimalizace je založena na ukončení vykreslování následných vrstev v případě, že v aktuální vrstvě nebyly zpracovány žádné průhledné fragmenty. První varianta zaznamenává výsledek pomocí atomických čítačů, druhá varianta pomocí shader storage bufferu. Porovnání doby vykreslování bez optimalizace a se zmíněnými variantami je možné vidět v grafu 5.7. Naměřené hodnoty se pro různé modely značně lišily. V některých případech byla rychlejší varianta s atomickými čítači než varianta se shader storage bufferem, i přesto že čítače vyžadují atomický přístup. Pravděpodobně je to způsobeno mírou souběhu operace



Graf 5.6: Porovnání doby vytváření hluboké stínové mapy a běžné stínové mapy pro různé modely. GeForce, 1 vrstva, 1024x1024.



Graf 5.7: Porovnání doby vytváření hluboké stínové mapy v závislosti na počtu vrstev a použité optimalizaci. GeForce, Sibenik Cathedral, 1024x1024.



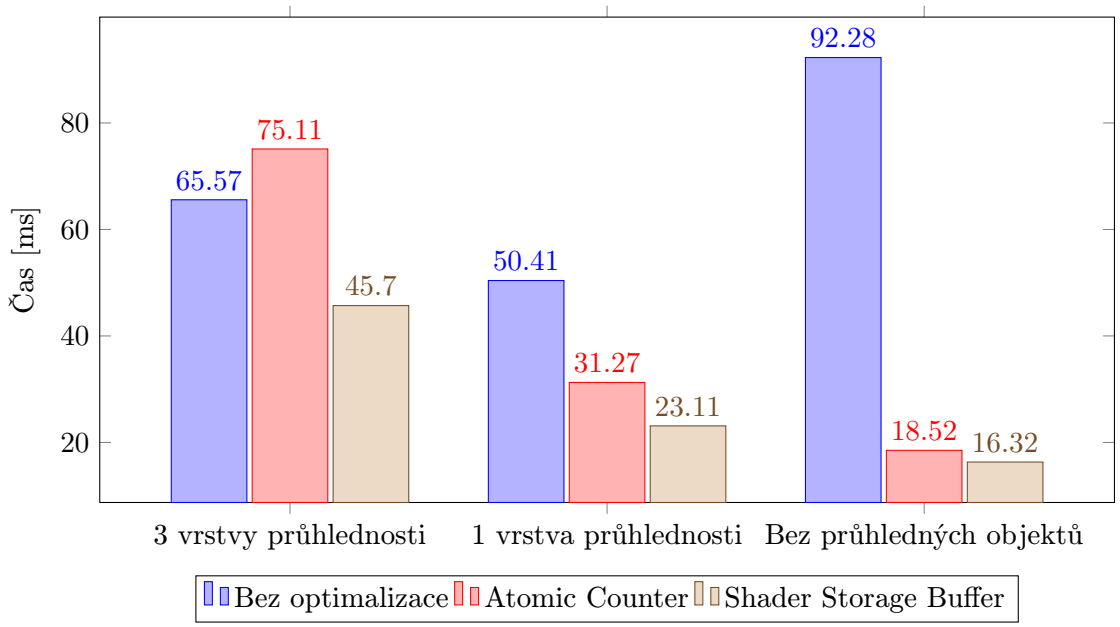
Graf 5.8: Porovnání doby vytváření hluboké stínové mapy v závislosti na počtu vrstev a použitém modelu. GeForce, SSB, 1024x1024.

inkrementace čítače, která může být různá. Výhoda optimalizace silně závisí na rozložení a komplexnosti scény a umístění světla. Pokud se nacházejí průhledné objekty pouze v jedné stěně cube mapy hluboké stínové mapy, lze dosáhnout razantního zrychlení. Naopak v případě, kdy se průhledné objekty nachází ve všech stěnách a je potřebné vykreslit většinu vrstev, může optimalizace vykreslování ještě zpomalit. To je způsobeno nutností kontroly a zápisu do atomického čítače nebo shader storage bufferu, nelze však přeskočit žádný vykreslovací průchod a investovaný čas se proto nezúročí.

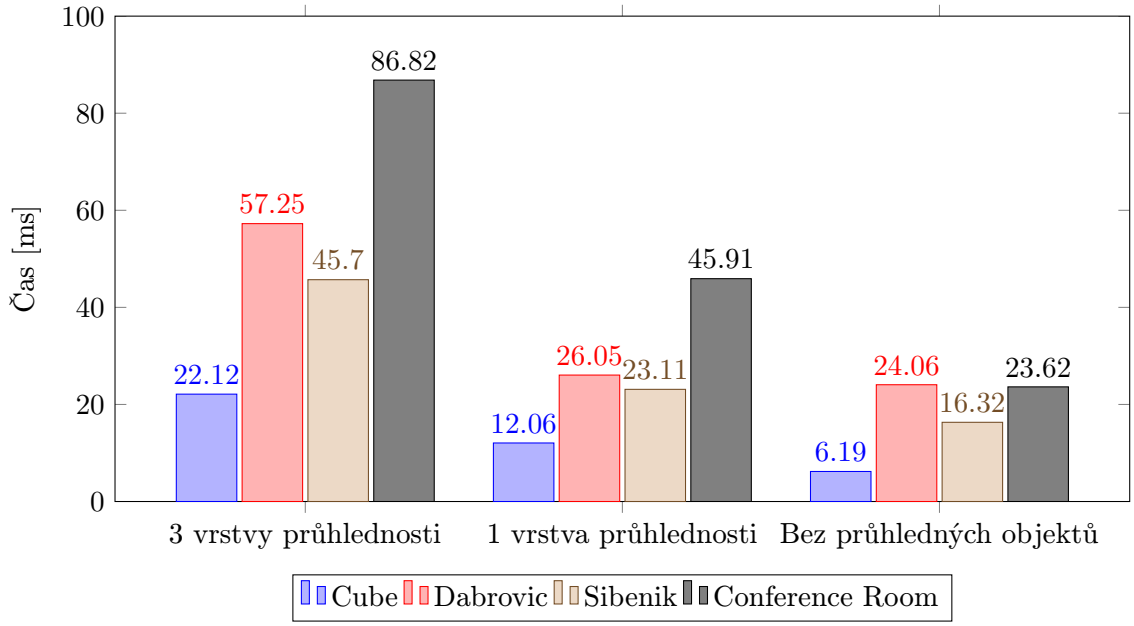
sGraf 5.8 zobrazuje dobu vykreslování hluboké stínové mapy v závislosti na počtu vrstev pro různé modely. Zde jo možné dobře vidět důsledek optimalizace, při třech oknech za sebou jsou 4 vrstvy stínové mapy dostatečné pro zachycení všech stínů. Přidáním dalších dvou vrstev již doba vykreslování témeř nenarůstá, protože tyto vrstvy nejsou nikdy využity. Pokud jsou vrsty využity, roste doba vykreslování úměrně počtu vrstev.

Optimalizace popsané výše pro urychlení tvorby hluboké stínové mapy byly použity i pro Dual Depth Peeling. Z grafu 5.9 je zřejmé, že optimalizace se nejlépe uplatní v případě, že v pohledu není žádný průhledný objekt. Díky optimalizaci se provede vždy vykreslení pouze první vrstvy nezávisle na počtu hloubkových vrstev viditelné geometrie. Při průchodu bez optimalizace se procházely všechny hloubkové vrstvy. Při vyšším počtu průhledných objektů a pozadí s nízkým počtem vrstev hloubky se optimalizace téměr neuplatní. Na rozdíl od vytváření hluboké stínové mapy, je v případě Dual Depth Peelingu pro všechny modely i typy pohledů rychlejší verze s shader storage bufferme oproti verzi s atomickým čítačem.

Graf 5.10 zobrazuje závislost doby vykreslování v závislosti na počtu trojúhelníků. Graf vykazuje podobné charakteristiky jako při vykreslování hluboké stínové mapy, s počtem trojúhelníků rosto potřebná doba. Model Dabrovic Sponza je však opět vykreslen pomaleji než Sibenik Cathedral, zřejmě z důvodu uspořádání trojúhelníků.



Graf 5.9: Porovnání doby vykreslování scény pomocí Dual Depth Peelingu v závislosti na počtu průhledných vrstev a typu optimalizace. Sibenik Cathedral. GeForce.



Graf 5.10: Porovnání doby vykreslování scény pomocí Dual Depth Peelingu v závislosti na počtu trojúhelníků pro různé pohledy. SSB. GeForce.

Kapitola 6

Závěr

V rámci práce byla prozkoumána řada metod souvisejících s vykreslováním průhledných stínů. Pro vytvoření demonstrační aplikace byly vybrány metody Depth Peeling (a varianta Dual Depth Peeling) a hluboké stínové mapy (Deep Shadow Maps). Vytvořená aplikace dokáže zobrazovat stíny průhledných objektů a to včetně barevných stínů. Lze použít objekty s homogenní částečně průhlednou barvou nebo objekty s průhlednou texturou. Díky Dual Depth Peelingu není nutné řazení průhledných objektů před vykreslením. Byly prozkoumány možné optimalizace pracující na principu přeskočení některých fází vykreslování, pokud nejsou nutné. Tyto optimalizace je možné využít při tvorbě hluboké stínové mapy i při vykreslování scény pomocí Dual Depth Peelingu, zrychlení však přináší pouze v určitých podmínkách. Z demonstrační aplikace vzniklo také ukázkové video.

Při řešení se vyskytl problém s načítáním některých modelů, pravděpodobně kvůli nedokonalosti použité knihovny, tento nedostatek však není pro účely aplikace příliš zásadní. Dalším problémem je nesprávné zobrazení průhledných objektů na sestavě Radeon, kvůli pozdnímu odhalení tohoto problému však již nebyla nalezene příčina problému.

Tvorba hluboké stínové mapy probíhá s jedním vzorkem na pixel. Proto se jako další možné směřování práce nabízí využití více vzorků na pixel a tím potenciálně dosáhnout vyšší kvality stínu. Další vylepšení kvality je možné dosáhnout implementací metody Percentage Closer Filtering při vyhodnocování zastínění objektů. Dále by bylo vhodné implementovat metodu Stochastic Transparency pro zobrazování scény, popřípadě i generování hluboké stínové mapy a porovnat její rychlost a kvalitu s metodu Dept Peeling. Realističnost zobrazení je možné vylepšit pomocí aplikace Fresnel efektu na průhledné povrchy.

Literatura

[1] Glassner, A. S.: *An introduction to ray tracing*. Morgan Kaufmann, 1989, ISBN 0-12-286160-4.

[2] Schmittler, J.; Wald, I.; Slusallek, P.: SaarCOR – A Hardware Architecture for Ray Tracing. *Graphics Hardware*, 2002: s. 1–11.

[3] Assarsson, U.; Dougherty, M.; Mounier, M.; aj.: An Optimized Soft Shadow Volume Algorithm with Real-Time Performance. *Graphics Hardware*, 2003.

[4] Stamminger, M.; Drettakis, G.: Perspective Shadow Maps. *ACM Trans. Graph.*, ročník 21, č. 3, Červenec 2002: s. 557–562, ISSN 0730-0301, doi:10.1145/566654.566616.
URL http://doi.acm.org/10.1145/566654.566616

[5] Reeves, W. T.; Salesin, D. H.; Cook, R. L.: Rendering Antialiased Shadows with Depth Maps. *SIGGRAPH Comput. Graph.*, ročník 21, č. 4, Srpen 1987: s. 283–291, ISSN 0097-8930, doi:10.1145/37402.37435.
URL http://doi.acm.org/10.1145/37402.37435

[6] Engel, W.: Cascaded shadow maps. *ShaderX*, ročník 5, 2007: s. 197–206.

[7] Lloyd, D. B.; Govindaraju, N. K.; Quammen, C.; aj.: Logarithmic Perspective Shadow Maps. *ACM Trans. Graph.*, ročník 27, č. 4, Listopad 2008: s. 106:1–106:32, ISSN 0730-0301, doi:10.1145/1409625.1409628.
URL http://doi.acm.org/10.1145/1409625.1409628

[8] Brabec, S.; Annen, T.; Seidel, H.-P.: Shadow mapping for hemispherical and omnidirectional light sources. 2002: s. 397–407.

[9] Crow, F. C.: Shadow Algorithms for Computer Graphics. *SIGGRAPH Comput. Graph.*, ročník 11, č. 2, Červenec 1977: s. 242–248, ISSN 0097-8930, doi:10.1145/965141.563901.
URL http://doi.acm.org/10.1145/965141.563901

[10] Brabec, S.; Seidel, H.-P.: Shadow Volumes on Programmable Graphics Hardware. *Computer Graphics Forum*, ročník 22, č. 3, 2003: s. 433–440, ISSN 1467-8659, doi:10.1111/1467-8659.00691.
URL http://dx.doi.org/10.1111/1467-8659.00691

[11] Heidmann, T.: Real shadows, real time. *Iris Universe*, ročník 18, 1991: s. 28–31.

[12] Everitt, C. W.; Kilgard, M. J.: Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. *CoRR*, ročník cs.GR/0301002, 2003.

[13] Bergeron, P.: A General Version of Crow's Shadow Volumes. *Computer Graphics and Applications, IEEE*, ročník 6, č. 9, 1986: s. 17–28, ISSN 0272-1716, doi:10.1109/MCG.1986.276543.

[14] Williams, L.: Casting Curved Shadows on Curved Surfaces. *SIGGRAPH Comput. Graph.*, ročník 12, č. 3, Srpen 1978: s. 270–274, ISSN 0097-8930, doi:10.1145/965139.807402.
URL http://doi.acm.org/10.1145/965139.807402

[15] Fernando, R.; Kilgard, M. J.: *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003, ISBN 0321194969.

[16] Phong, B. T.: Illumination for Computer Generated Pictures. *Commun. ACM*, ročník 18, č. 6, Červen 1975: s. 311–317, ISSN 0001-0782, doi:10.1145/360825.360839.
URL http://doi.acm.org/10.1145/360825.360839

[17] Dachsbacher, C.; Stamminger, M.: Reflective Shadow Maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, New York, NY, USA: ACM, 2005, ISBN 1-59593-013-2, s. 203–231, doi:10.1145/1053427.1053460.
URL http://doi.acm.org/10.1145/1053427.1053460

[18] Sloan, P.-P.: Stupid spherical harmonics (sh) tricks. In *Game Developers Conference*, ročník 2008, Microsoft Corporation, 2008.

[19] Kaplanyan, A.; Dachsbacher, C.: Cascaded Light Propagation Volumes for Real-time Indirect Illumination. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '10, New York, NY, USA: ACM, 2010, ISBN 978-1-60558-939-8, s. 99–107, doi:10.1145/1730804.1730821.
URL http://doi.acm.org/10.1145/1730804.1730821

[20] Everitt, C.: Interactive order-independent transparency. *White paper, nVIDIA*, ročník 2, č. 6, 2001: str. 7.

[21] Bavoil, L.; Myers, K.: Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK*, 2008.

[22] Enderton, E.; Sintorn, E.; Shirley, P.; aj.: Stochastic transparency. *Visualization and Computer Graphics, IEEE Transactions on*, ročník 17, č. 8, 2011: s. 1036–1047.

[23] Lokovic, T.; Veach, E.: Deep Shadow Maps. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, ISBN 1-58113-208-5, s. 385–392, doi:10.1145/344779.344958.
URL http://dx.doi.org/10.1145/344779.344958

[24] McGuire, M.: Computer Graphics Archive. August 2011.
URL http://graphics.cs.williams.edu/data

[25] Hearn, M.: Transparent Textures. [online], [cit. 2014-5-24].
URL http://www.transparenttextures.com/