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

SHADOW MAPPING: FILTRACE STÍNŮ V OPENGL

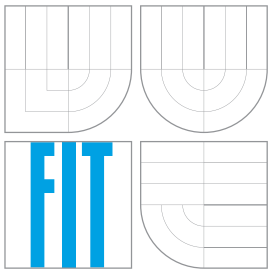
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JÁN BRÍDA

BRNO 2015

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

SHADOW MAPPING: FILTRACE STÍNŮ V OPENGL

SHADOW MAPPING: SHADOW FILTERING IN OPENGL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JÁN BRÍDA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JOZEF KOBRTEK

BRNO 2015

Abstrakt

Obsah této bakalářské práce pojednává o generování a způsobech filtrování stínů v 3D aplikacích. Popisuje možnosti potlačení jevu diskretizace některými technikami a dokumentuje artefakty pro ně typické. Čtenář se v něm zároveň dozví o jejich výkonnostích a kvalitativních rozdílech, čímž bude schopen zhodnotit nabízené implementace.

Abstract

The content of this bachelor thesis discusses generation, and filtration ways of shadows in 3D applications. It describes possibilities to surpress the discretization phenomenon using certain techniques, and documents the artifacts typical for them. The reader will also learn about their performance, and quality differences, thereby being able to judge offered implementations.

Klíčová slova

Mapování stínů, aliasing stínů, filtrování stínů, Procentně-bližší filtrování, Variantní stínové mapy, Konvoluční stínové mapy, Exponenciální stínové mapy

Keywords

Shadow Mapping, shadow aliasing, shadow filtering, Percentage-Closer Filtering, Variance Shadow Maps, Convolution Shadow Maps, Exponential Shadow Maps

Citace

Ján Brída: Shadow Mapping: filtrace stínů v OpenGL, bakalářská práce, Brno, FIT VUT v Brně, 2015

Shadow Mapping: filtrace stínů v OpenGL

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jozefa Kobrtka. Další informace mi poskytli Ing. Tomáš Starka a Ing. Tomáš Milet. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ján Brída
20. května 2015

Poděkování

Rád bych zde poděkoval panu Ing. Jozefovi Kobrtkovi a také jeho kolegům z UPGM za cenné rady a skvělý přístup k problémům, které se během práce na tomto projektu vyskytli.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Zobrazování stínů pozitivně přispívá při tvorbě realisticky vypadajících scén ve 3D, jelikož jsou díky nim tvary a relativní pozice objektů lépe rozpoznatelné lidským okem. V grafických efektech představují stále velmi diskutované téma, vzhledem k nedostatkům běžně používané metody jejich vytváření – Shadow Mappingu. Tato technika se v dnešní době stala prakticky standardním řešením vrhání stínů v trojrozměrném prostoru, od které se odvíjejí různé modifikace. Je oblíbená především kvůli jednoduchosti implementace a také lineárnímu růstu výpočtového času. Naneštěstí ale často způsobuje diskretizační artefakty, kvůli kterým mají přechody mezi osvětlenou a zastíněnou oblastí nepřirozeně výrazné schodovité hrany. Z tohoto důvodu byly navrženy různé varianty, aplikující anti-aliasing, občas za cenu velkých hardwarových požadavků anebo nových nežádoucích artefaktů. To, která technika má lepší výsledek, jak kvalitativní, tak co se nároků na výpočet/nutnou paměť týče, je podstatná znalost při použití těchto metod, či už jsou cílovou oblastí počítačové hry nebo animované filmy.

Rozsáhlá kapitola 2 vysvětluje fundamentální algoritmus Shadow Mappingu, popisuje aliasing i ostatní nedostatky. To jakým způsobem se dá vypořádat s ostrými hranami stínů je následně rozebráno v rámci dalších sekcí, představujících zkoumané filtrační metody: Percentage-Closer Filtering, Variance Shadow Maps, Convolution Shadow Maps a Exponential Shadow Maps.

Kapitola 3 popisuje demonstrační aplikaci, tzn. jaké byly cíle a požadavky, načež později ujasňuje, jak byly splněny. Zároveň probírá některé implementační detaily jednotlivých stínovacích technik.

V kapitole 4 se lze setkat s porovnáním implementovaných metod. Zde se dá získat rychlý přehled nabízených způsobů filtrování a doporučení v jejich volbě.

Konečne závěr 5 konstatuje dosažené výsledky a navrhuje možná vylepšení.

Kapitola 2

Teorie

Shadow Mapping je velmi působivým algoritmem, protože staví na triviální myšlence viditelnosti objektů z místa, kde se nachází světelný zdroj. Ta určuje nezastíněné oblasti scény, tudíž zbylé povrchy lze považovat za neosvětlené [7]. Ovšem kvůli diskretizační povaze důležité komponenty této techniky a zároveň způsobu vyhodnocení viditelnosti dochází k nechtěnému aliasingu stínů. Proto je dále nutné řešit tyto potíže, aby bylo možné nakonec získat dobře vypadající výsledek.

2.1 Shadow Mapping

V roce 1978 byla představena metoda stínování zvaná Shadow Mapping (SM). Popsána L. Williamsem v [7], stala se postupem času nejvíce používanou technikou řešící vytváření stínů. Její autor navrhl, že běžný z-buffering může být spolehlivým základem generování stínů pro libovolné objekty. Myšlenkou je vykreslit scénu z hlediště zdroje světla. To může „sledovat“ viditelné povrchy v určitém směru, kterým se rozlišuje jeho typ. Jedná se buď o jednosměrné (typicky Slunce pro velké geometrie), kterého projekce je nejčastěji popsána ortogonálně, bodové (např. žárovka), kdy jsou fotony vysílány do všech směrů, anebo nějaký druh světlometu, jehož zorné pole je úhlově omezeno [1].

Moderní SM zakládá na použití dalšího framebufferu, kterého obsah je zkopírován do textury. Nazývá ji také *hloubkovou* (nebo *stínovou*) mapou. Z ní se později vzorkuje uchovaná hloubka při provádění testu viditelnosti fragmentu.

2.1.1 Algoritmus

Tato technika pro každý texel hloubkové mapy uchovává hloubku povrchu nejblíže od světelného zdroje. Je použita při vykreslování scény z pozice pozorovatele. Během renderování jednotlivých primitiv je vzdálenost d od světla na každém fragmentu testována se z hodnotou uloženou v hloubkové mapě. Pokud je tato vzdálenost větší než vzorek z-bufferu, je fragment klasifikován jako zastíněn, protože se před ním musí nacházet nějaký okluzor, jehož vzdálenost od světla byla definována právě hloubkou v textuře. Vzorkování v tomto testu je dáno projekcí souřadnic zpracovávaného bodu (vykresleného znovu z pozice světla) do prostoru hloubkové mapy [7].

Rekapitulací se dá SM rozdělit do dvou fází:

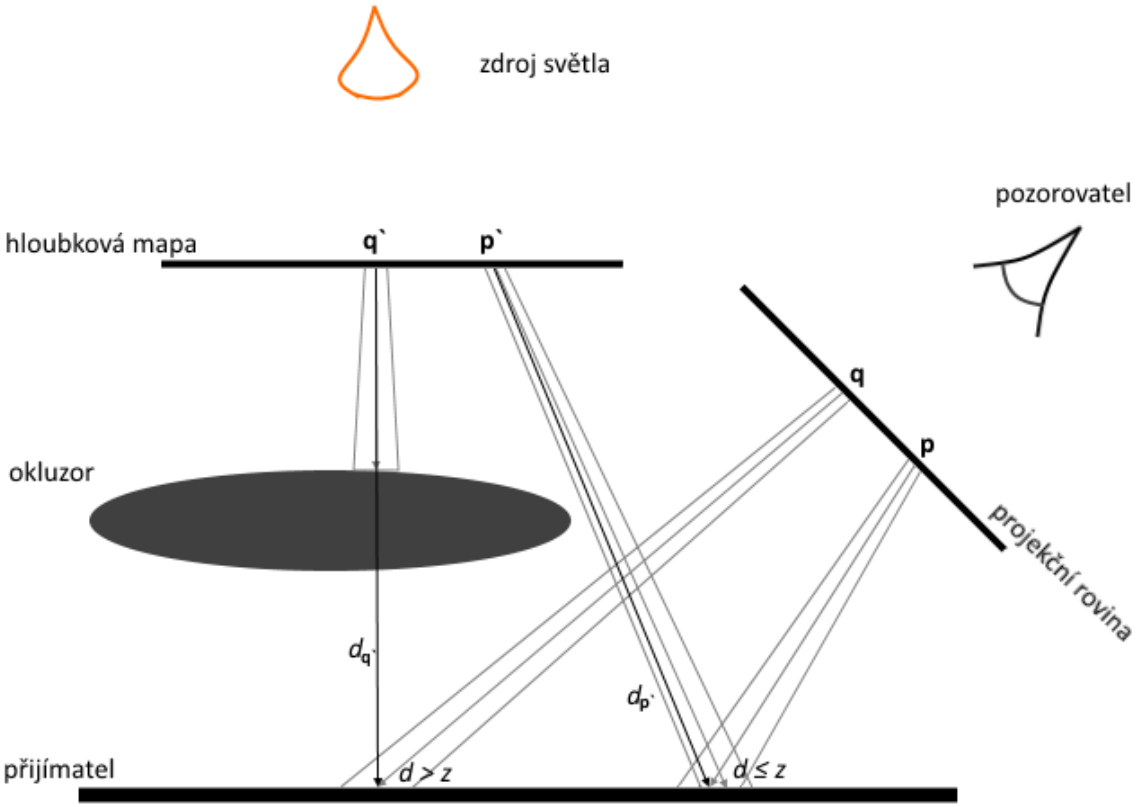
1. První fáze zaznamená všechny povrchy okluzorů viditelnými z místa vysílání světelných loučů do speciální textury během vykreslování do dodatečného framebufferu.

Hloubka na jednotlivých texelech je poté využita při vyhodnocení binární porovnávací funkce $f(d, z)$.

2. Při vizualizaci scény na zobrazovací zařízení se provede transformace souřadnic vykreslovaných bodů, aby odpovídali pozicím z pohledu zdroje světla. Poté je provedeno porovnání hloubky d transformovaného bodu se souřadnicemi $[x, y]$ a hloubkou z uchovanou v texelu hloubkové mapy na souřadnicích $[x, y]$. Výsledkem testu $f(d, z)$ je binární reprezentace viditelnosti:

$$f(d,z) = \begin{cases} 0 \Rightarrow \text{fragment je ve stínu} & d > z \\ 1 \Rightarrow \text{fragment je osvětlen} & d \leq z \end{cases} \tag{2.1}$$

Pro lepší představu této fáze viz obr. 2.1.



Obrázek 2.1: Tento obrázek popisuje druhou fázi SM algoritmu. Souřadnice bodů ve vztahu k pozici světelného zdroje jsou znovu rekonstruovány pro hloubkové porovnání d s hodnotou z uloženou v texelu, pokrývajícím určené okolí. Fragment $\mathbf{p}$, pro nějž platí, že $d_{\mathbf{p}'}$ je (přibližně) stejnou vzdáleností od místa zdroje světla, jaká byla uchována na odpovídajícím texelu $\mathbf{p}'$, je osvětlen, zatímco fragment $\mathbf{q}$ je ve stínu, protože pro něj není porovnání $d \leq z$ vyhodnoceno jako pravdivé. Zde je možno všimnout si i jak vypadá zarovnání: $\mathbf{q}$ a $\mathbf{q}'$ jsou na své pozici dostatečně zarovnány a na tomto místě nevzniká aliasing. Druhý fragment je však kvůli nedostatečnému rozlišení textury mapován na texel, kam bude naneseno i jeho okolí. Samozřejmě, v tomto případě k viditelnému efektu nedochází, protože daná oblast není nikde zastíněna.

U typu světelného zdroje, kdy se světlo šíří ve všech směrech, se provádí test viditelnosti zvlášť na šesti hloubkových mapách, protože směr šíření světla je reprezentován pomocí kostky [1]; každá strana pro jednu projekci.

2.1.2 Nedostatky

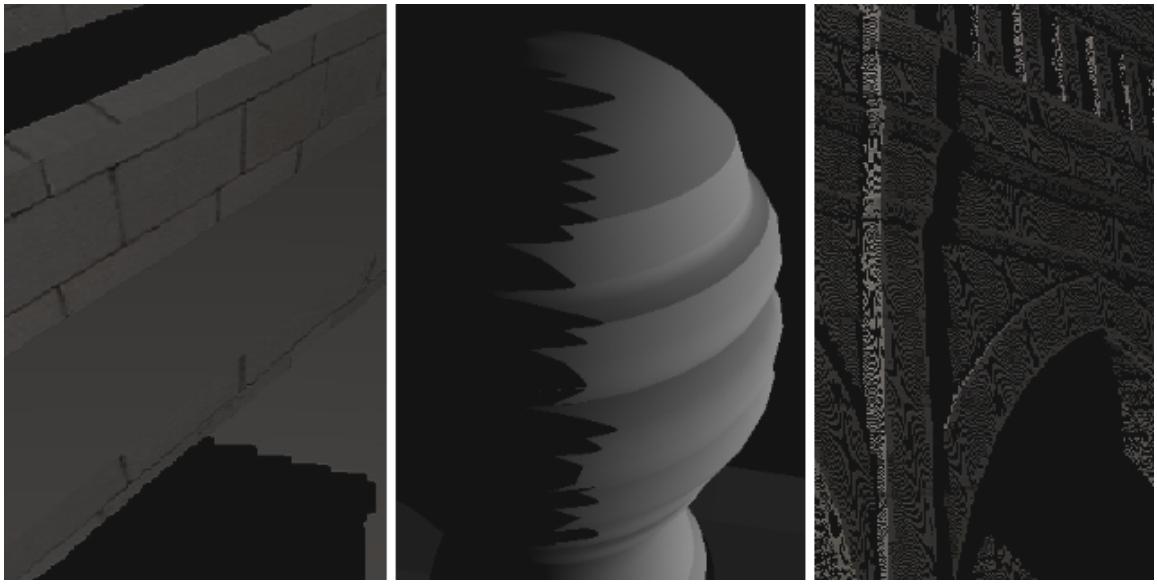
Přestože je implementačně SM velmi jednoduchá a cena vystavení hloubkové mapy je lineární podle počtu renderovaných primitiv (také nelze opomenout konstantní přístupový čas při vzorkování), přináší tato metoda vytváření stínů zásadní nedostatky (viz obr. 2.2), kterých řešení nebývá jednoduché.

Doprovodnou vadou bývá aliasing hran stínů. Výsledkem jsou ošklivé zubaté hrany mezi oblastmi přechodů intenzity světla. Rozpoznáváme jeho dva významné druhy. První je *perspektivní*, dán mapováním texelů hloubkové mapy na pixely zobrazovacího zařízení. Nízké rozlišení způsobuje, že část pixelů blíže roviny Near je nanesena na stejný texel, zatímco fragmenty, prezentující vzdálenější body, se mapují lépe, protože nevyžadují vyšší rozlišení textury [1]. Ostré hrany nevznikají, pokud jsou texely hloubkové mapy a pixely zobrazovacího zařízení zarovnány 1:1. Texely jsou také roztažené na místech, kde je směr dopadu světla paralelně s povrchem primitiva, které je však natočeno kolmo k pozorovateli, problém zvaný *projekční* aliasing. Nastává v situacích, kdy je celý polygon pokryt jediným texelem [1].

Povaha stínovacího testu zabraňuje aplikací běžných filtrovacích technik, jako mipmapping nebo anizotropie, vymítit popsané jevy [1]. Interpolace tedy přináší minimální výhodu v podobě přesnějšího vzorku, kterého hodnota je ale stále porovnávana s hloubkou d, stejnou pro celý filtrovaný region. Hrany se sice stávají zaoblenějšími, na druhou stranu je však stále patrný nežádoucí schodovitý efekt. Jelikož jsou tedy vzdálenosti od světelného zdroje vykreslovaných bodů v druhé fázi počítány až za běhu, je nemožné anti-aliasing aplikovat metodami zabudovanými v grafických čipech – binární výsledek funkce $f(d, z)$ je znám až po vzorkování a jeho interpolace musí probíhat nějakým jiným způsobem.

Další nevýhodou je granularita z-bufferu, která narušuje kvalitu scény při jevu zvaném *z-fighting*. z hodnoty fragmentů se poté kvůli nepřesnostem bufferu, který bývá typicky 16-bitový, zle odlišují. SM algoritmus je tímto negativně ovlivněn projevováním *povrchového akné* [7], které se dá popsat jako moaré vzor na osvětlených oblastech. Zdrojem těchto potíží může být nejen numerická nepřesnost. Kvůli faktu, že hloubka určité oblasti je reprezentována jedním texelem, vzorky získané světelnou projekcí během první fáze nejsou téměř nikdy na stejných místech jako vzdálenost od světla na zpracovávaném fragmentu. Diskretizace textury tedy bývá důvod, proč jsou některé rekonstruované hloubky menší/větší než jejich reprezentace ve stínové mapě [7]. Tento jev bývá také nazýván sebe-zastínění[1].

[1] Volný překlad z *self-shadowing*.

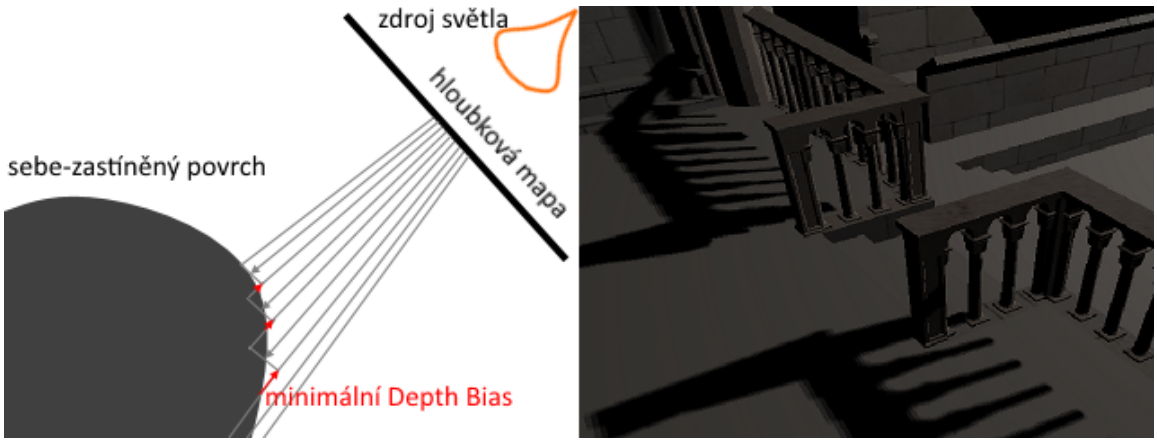


Obrázek 2.2: Zleva doprava: aliasing perspektivní, projekční, sebe-zastínění.

2.1.3 Vylepšení

Zaobýváním se potlačení sebe-zastínění, programátor je schopen tomuto negativu částečně vyhýbat vybráním dokonalejší reprezentace vzorkovaných hloubek. Bitový rozsah kanálu textury silně ovlivňuje přesnost při porovnání [1]. Lineární z-buffer ji umožňuje zachovat pro vzdálenější objekty, ovšem použití ortogonální projekce při nasvětlování velkých scén je těžkopádným způsobem jak tento problém zmírnit, protože čím větší je rozpětí projekce, tím více jsou bitovým rozsahem ovlivněny i blízké objekty. Perspektivní projekce, produkující nelineární hloubku, však určitý typ řešení poskytuje a to i pro objekty blíže zdroje světla. Přesnost v tomto případě je ovlivněna vzdáleností mezi rovinami Near a Far. Čím menší je tento rozdíl, tím lepší výsledky perspektivní projekce poskytuje. Upravením jejich pozic na pomezí nasvětlované scény tedy vylepšuje samotný porovnávací test [1]. Rovina Near by měla být nastavena co nejdále od světla, zatímco Far co nejblíže k poslednímu objektu scény. Metodou upravení frusta lze také docílit lepších výsledků během magnifikace texelů. Jak ale bylo vysvětleno v předchozí sekci 2.3.2, přesnost naneštěstí není jediným (a zcela ne tak významným faktorem) způsobujícím akné na osvětlených površích. Existují mnohem lepší modifikace, které uměle posouvají hranici hloubky vzorkovaného povrchu. Principem odsazení z hodnot texelů od povrchu snímaného během druhé fáze je aplikace tzv. *Depth Bias* [7]. Jedná se o konstantu b s minimální hloubkou, potřebnou pro úspěšné porovnání $f(d, z)$, které nebude mít za následek sebe-zastínění. Může být aplikována během vytváření stínové mapy $z + b$, nebo během testu $d - b$. Její velikost je dána nejvyšší strmostí povrchu ve scéně. Má to ovšem negativní vliv na kvalitu stínu, který je nazýván libozvučně *Peter Panning* (PP) [1], viz také obr. 2.3. Pro povrchy, kde by stačilo použití mnohem nižšího b než je vyžadováno pro nejstrmější povrch dopadne $f(d, z)$ na některých místech, které mají být správně zastíněny, v prospěch $d \leq z$. Následkem toho je, že lidskému oku se některé objekty jeví jako plovoucí, důsledkem ztráty stínu na kontaktních bodech s přijímatelem stínu. Jinak řečeno: objekt vypadá, jakoby se vznášel. Řešením by bylo aplikovat variabilní b pro jednotlivé povrchy. Naštěstí byly takéto způsoby eliminace akné již vymysleny, ovšem existuje ještě metoda na principu uložení hloubky zadních stran okluzerů během

fáze jedna [1]. Je to velmi jednoduchá úprava SM, zvaná *Second-Depth Shadow Mapping*. Není bez určitých omezení. Geometrii scény je nutné pozměnit, pokud jsou některé objekty tvořeny pouze jednou stranou. Může to být například stěna nebo obvykle terén. V také situaci je informace o hloubce ztracena, což vytváří dojem nekonzistentní scény, kdy některý přijímatel stín má a jiný ne. Dostatečná tloušťka všech těles tuto potíž redukuje, což má ale nežádoucí vliv na výkon, protože bývá potřeba renderovat více primitiv. Druhým nedostatkem je akné nacházející se právě na zadních stranách okluzerů. Problém sebe-zastínění tedy není pomocí této metody vyřešen úplně, protože se v podstatě jen přesune dál od zdroje světla. Zřejmou vadou na této úpravě stínování je také zmizení stínů na těchto místech. Porovnávaná hloubka při rekonstrukci je téměř totožná s hodnotami v textuře, takže výsledek porovnávací funkce je 1. Samozřejmě předpokládaným vyhnutím se tohoto problému je navržené okamžité stínování všech odvrácených polygonů, což zároveň zrychluje chod aplikace. Dynamičtější praktikou ofsetování hloubky je *Slope-Scaled Depth Bias* [1] při zaznamenávání primitiv nejblíže světelnému zdroji. z je posunuta dle toho, jak moc je na daném místě velký hloubkový sklon. Ten je pro polygony, které jsou paralelně s rovinami Near a Far roven nule, zatímco v ostatních případech je hloubka ofsetována podle naklonění[2]. I v tomto případě se však ne zcela dá vyhnout PP. Na GDC[3] 2011 byl poprvé prezentován elegantní způsob, jak proti němu efektivně bojovat: *Normal Offset*[4]. Aby se vyhlo použití minimálního Depth Bias, který by negativně působil na ostatní povrchy, použije se pro každý fragment přesně takový, jaký udává příslušná normála rasterizovaného bodu. Dá se o tom uvažovat jako o trojrozměrném škálování, kdy se každá souřadnice mění jedinečně. Laicky řečeno; připočítáním normál k pozicím vrcholů se objekt uměle posouvá dle jejich směrů (typicky se toto posunutí částečně redukuje použitím konstanty). Takto se ve vertex shaderu upravují vzorkovací souřadnice při rekonstrukci stínů.



Obrázek 2.3: Peter Panning způsobený konstantním Depth Bias. U nelineárního z-bufferu má na vzdálenějších objektech větší efekt, kvůli aritmetické nepřesnosti.

Nejkomplikovanější problematikou SM je aliasing hran. Vyšší rozlišení textury uchovávající hloubky nejbližších okluzorů poskytuje lepší mapování na zpracovávané fragmenty, ale za cenu vyšších paměťových nároků. Neustále však platí, že zarovnání pixelů zobrazovacího zařízení s texely hloubkové mapy není ve většině případech perfektní a protože

[2] V OpenGL možno pro implementaci použít funkci `glPolygonOffset`.
[3] Game Developers Conference.
[4] Normal Offset plakát, vysvětlující princip modifikace:
http://www.dissidentlogic.com/old/images/NormalOffsetShadows/GDC_Poster_NormalOffset.png.

d musí být vypočtena za běhu během druhé fáze, není možné výsledky testů, které by se dali interpolovat, uložit do používané textury [7]. Vyžadovanou funkčností je provést tento proces právě na samotných výstupech porovnání, případně změnit základní povahu SM, aby bylo možné využívat linearity. Následující sekce probírají techniky, které se pokusili o vytváření těchto modifikací.

2.2 Percentage-Closer Filtering

Představena studiem Pixar v roce 1987 [6], zabývá se problémem aliasingu hloubkových map, který je způsoben jejich diskretizací (popsáno v sekci 2.1.2). Je to velmi elegantní řešení změkčení stínů, kterého efektivita je ovšem omezena velikostí filtračního kernelu. Místo porovnání zkoumané hloubky d se z na jediném odpovídajícím texelu je zvolen filtrovaný region, na kterém probíhá akumulace výsledků $f(d, z)$, načež je suma následně vydělena celkovým počtem vzorkovaných texelů. Zbavuje SM nepříjemných ostrých hran stínů, jelikož výsledkem je plynulý přechod mezi osvětlenou a zatemnělou oblastí. Percentage-Closer Filtering (PCF) je ovlivněno více faktory, jako například rozlišením stínové mapy.

2.2.1 Algoritmus

Na velikosti PCF kernelu pak závisí konečná měkkost stínu. Menší jádro ponechává stín tvrdší a naopak [6]. Množství dopadajícího světla je funkcí míry viditelnosti světelného zdroje z osvětlovaného místa. PCF produkuje měkké stíny podle množiny míst, které sousedí se snímaným bodem, vytvářejíc dobře vypadající penumbru[5]. Tedy najít procentuálně množství vzorků, které jsou nezastíněny, je hlavním principem tohoto algoritmu.

Uvažujeme-li velikost kernelu K^2, pak tento algoritmus vypadá jednoduše, viz algoritmus 1. Průměrované hodnoty $f(d, z)$ jsou závislé na ofsetování vstupních $[u, v]$ souřadnic použitých při vzorkování hloubkové mapy. Toto posunutí $[x, y]$ v prostoru hloubkové mapy je průběžně inkrementováno a je nutné ho normalizovat velikostí jednoho texelu textury pro korektní ofsetování na daném regionu.

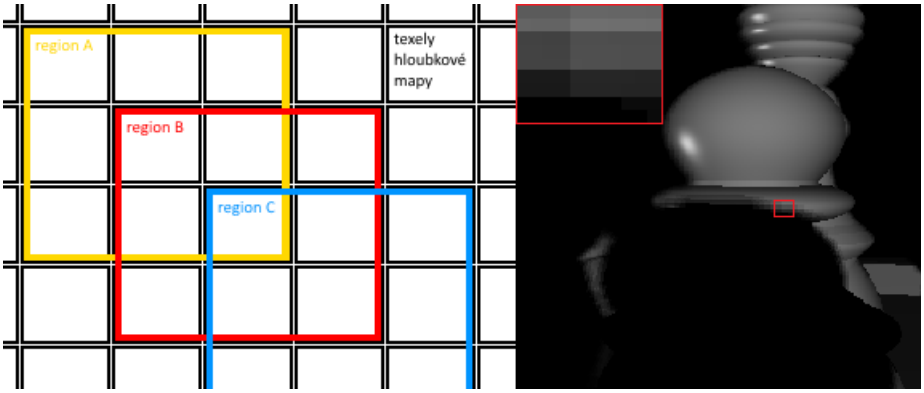
Algoritmus 1: Percentage-Closer Filtering

Input: zkoumaná hloubka d, výchozí souřadnice $[u, v]$
Output: intenzita světla I

1: $I \leftarrow 0$
2: $e \leftarrow (K - 1)/2$
3: **for** $y = -e$ **to** $y = e$ **do**
4: | **for** $x = -e$ **to** $x = e$ **do**
5: | | $I \leftarrow I + f(d, z_{[u+x, v+y]})$
6: | **end for**
7: **end for**
8: **return** I/K^2

Jak je vidět na obrázku 2.4, výsledkem je velmi dobré eliminovaní nechtěných ostrých hran. Vše funguje na principu překrývání filtračních regionů [6], díky čemuž vznikají mezi aplikací na sousedících pixelech malé rozdíly, hlavně při magnifikaci. Čím více je textura magnifikována, tím „hladší“ je přechod mezi osvětlenou a zastíněnou oblastí.

[5] Penumbrou rozumíme místa v zastíněné oblasti, odkud je viditelná část světelného zdroje – zde je stín jemnější, narozdíl od umbry (nelze odtud vidět zdroj světla), kde je intenzita nejnižší.



Obrázek 2.4: Překrývání regionů mající za následek plynulý přechod a efektní anti-aliasing. Vpravo je vidět ukázka $K = 3$.

Výrobci grafických karet již poskytují hardwarovou implementaci PCF, typicky přes použití speciálního typu sampleru[6], sice jen v podobě bilineární interpolace výsledků $f(d, z)$. Necelá část $\theta = u - i$ a $\phi = v - j$ udává vzdálenost od texelu na pozici $[i, j]$. V případě oblasti 2×2 pak PCF možno popsat matematicky jako (2.2).

$$\begin{array}{rcl} f(d, z_{[u,j]}) & \approx & (1-\theta) \cdot f(d, z_{[i,j]}) + \theta \cdot f(d, z_{[i+1,j]}) \\ f(d, z_{[u,j+1]}) & \approx & (1-\theta) \cdot f(d, z_{[i,j+1]}) + \theta \cdot f(d, z_{[i+1,j+1]}) \\ f(d, z_{[u,v]}) & \approx & (1-\phi) \cdot f(d, z_{[u,j]}) + \phi \cdot f(d, z_{[u,j+1]}) \end{array} \tag{2.2}$$

2.2.2 Nedostatky

PCF fundamentálně nemění princip SM, ze kterého vychází, což znamená, že se v praxi potýká i se stejnými obtížemi, známými pro tento algoritmus. Nejvýraznějším bývá povrchové akné, jeho eliminace však není až tak složitá, jak bylo popsáno dříve v sekci 2.1.3.

Hlavní komplikací PCF je, že násobné vzorkování pro determinizaci zastínění jednoho bodu má negativní vliv na výkon aplikace – přístup do textury je drahá operace. Také uniformnost penumbry, která je dána konstantní šířkou K kernelu PCF má za následek její stejný rozsah na všech místech [6], včetně těch kontaktních okluzoru a přijímatele.

2.2.3 Vylepšení

Kvalitu PCF řešení udává K^2, ovšem je praktické omezovat velikost filtrovaného regionu, nejlépe na místech, kde nedochází k přechodům. Jedním ze způsobů, jak tohoto docílit je kontrolovat rozdíl vzorkovaných hodnot v sousedství. Pokud není dostatečně velký, lze se uspokojit s výsledkem vypočteným podle prvních texelů, v opačném případě se zahrnou další okolní body, viz str. 363 [1]. Celý proces se opakuje dokud není dosaženo K^2. Efekt PCF benefituje pouze na místech penumbry, pro je výhodné takéto vylepšení aplikovat.

2.3 Variance Shadow Maps

Tato varianta hloubkových map se zaobírá problematikou hardwarové filtrace textur, která kvůli povaze výchozího algoritmu nezbavuje stíny zubatých hran. Vestavěné způsoby anti-aliasingu jako mipmapping a anizotropní filtrování bývají v tomto případě nepoužitelné,

[6] V OpenGL se lze setkat se zástupcem této techniky `sampler2DShadow`.

protože interpolují pouze zaznamenané z hodnoty okolitých texelů. Principem Variance Shadow Maps (VSM) [4] je plně využít filtrování textur na dostupných grafických kartách pro eliminaci aliasingu.

2.3.1 Algortimus

Narozdíl od klasického SM, VSM při renderování do textury, která má tentokrát dva barevné kanály, zapisuje na fragment hloubku okluzoru a její druhou mocninu – tyto hodnoty jsou definovány jako momenty. Texel má za úkol reprezentovat (přibližnou) distribuci hloubek na daném místě. Výhodou této reprezentace je, že průměr dvou distribucí možno získat zprůměrováním jejich momentů [4]. Při konstrukci stínů jsou momenty využity při zjišťování meze na té části distribuce, která je dále od vzorkovaného bodu[7]. Tato meze poskytuje dobrou aproximaci množství dopadajícího světla pro vytváření hladkých hran za použití běžných filtrovacích možností daného hardware, jelikož momenty lze interpolovat.

V první fázi se VSM od konvenčního Shadow Mappingu liší zápisem z^2 do druhého kanálu zložky textury[8]. Před konstrukcí stínů je doporučeno nechat vygenerovat mipmapu pro usnadnění filtrování, případně aplikovat odstranění šumu pro zjemnění stínů.

Druhá fáze začíná získáním uložených momentů ve variantní hloubkové mapě. Ty jsou v [4] definovány jako

$$M_1 = E(z) = \int_{-\infty}^{\infty} zp(z)\mathrm{d}z \tag{2.3}$$

$$M_2 = E(z^2) = \int_{-\infty}^{\infty} z^2p(z)\mathrm{d}z \tag{2.4}$$

a z nich je vypočtena střední hodnota μ a rozptyl σ^2:

$$\mu = E(z) = M_1 \tag{2.5}$$

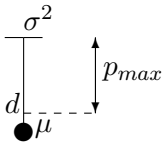
$$\sigma^2 = E(z^2) - E(z)^2 = M_2 - M_1^2 \tag{2.6}$$

Rozptyl σ^2 vyjadřuje šířku distribuce a udává mez, která určuje kolik distribuce může být koncentrováno od střední hodnoty, načež je poté využita v Čebyševově nerovnosti I. typu [4]. Pro $d \geq \mu$ se tak aplikuje:

$$P(z \geq d) \leq p_{max}(d) \equiv \frac{\sigma^2}{\sigma^2 + (d - \mu)^2} \tag{2.7}$$

Pravděpodobnost $P(z \geq d)$ udává část pixelů filtrovaného regionu, pro které porovnání $z \geq d$ s konstantní[9] hloubkou d dopadne negativně. Toto množství je přibližně stejné, jako v případě algoritmu PCF [4].

[7] U původní techniky stínování byla tato situace identifikována jako $d > z$.
[8] [4] v kapitole 3 zároveň navrhuje použití anti-aliasingu objektů.
[9] Při porovnání v regionu.



Obrázek 2.5: Čebyševova nerovnost I. typu procentně určuje, jak moc může být daný bod nezastíněn. Zde je vyobrazen případ, kdy je střední hodnota μ mírně menší než hloubka d testovaného fragmentu: $p_{max} = 0{,}8$. Situace s $d = \mu$ by dopadla $p_{max} = 1$.

2.7 je však jen horní mezí, naštěstí postačující ke správné aproximaci skutečného výsledku p, jak je dokázáno v sekci 3.1 [4]. Z ní vyplývá, že p, které je tedy definováno jako nezastíněná část filtrovaného regionu, může být rovno hodnotě p_{max}, typicky v situacích rovinných útvarů, čímž se z Čebyševovy nerovnosti prakticky stává rovnost. Jak bylo poznamenáno, jedná se o poměrně specifické případy, avšak běžně se lze setkat se situacemi, kdy se ve vzorkovaném okolí hloubka okluzoru a přijímače jeví přibližně konstantní, což uspokojuje renderovací účely.

Pro lepší pochopení následuje definice nabývání hodnot intenzity světla I:

$$I = \begin{cases} 1 & d < \mu \\ \frac{\sigma^2}{\sigma^2 + (d-\mu)^2} & d \geq \mu \end{cases} \tag{2.8}$$

Celý algoritmus nakonec možno zesumarizovat do několika bodů:

1. Ať je scéna renderována ze zorného pole zdroje světla. Výstup fragment shaderu zapisuje momenty z a z^2 do dvou-kanálové textury s floating point aritmetikou.

2. Generace mipmap[10] a předtím případné rozmazání variantní hloubkové mapy konvolučním kernelem (například 7×7 Gaussovské rozmazání).

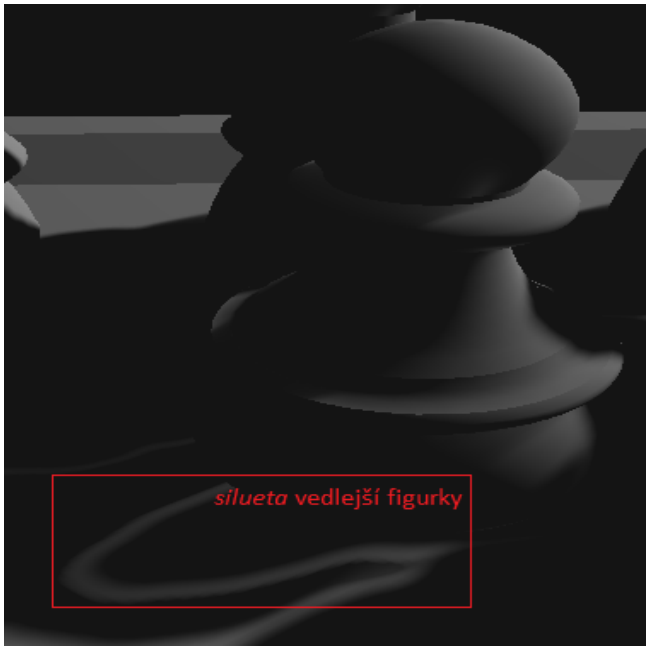
3. Renderuje se z pohledu pozorovatele. Po obdržení zaznamenaných momentů se provede test na $d < \mu$. Pokud platí, je fragment osvětlený. V opačném případě se vypočte rozptyl σ^2 a intenzita světla je škálována podle p_{max}.

2.3.2 Nedostatky

Při použití nizké bitové šířky pro reprezentaci distribuce bývá rovnice (2.6) numericky nestabilní, dle sekce 5.3 [4] již u 16-bitové aritmetiky, zatímco v případě dvounásobné reprezentace nejsou artefakty patrné. Vzhledem k tomu, že v té době byly možnosti grafických karet více omezené než je tomu dnes, není již tento problém při použití běžných 32-bitových textur aktuální. Pokud ale nejsou k dizpozici, je doporučeným postupem rozdělit 32-bitová čísla a každé uchovat ve dvou kanálech 16-bitové délky. Také projekce světla by měla produkovat lineární z-buffer pro vylepšení přesnosti vzdálenějších objektů (ne vždy je to vyžadováno).

Nejzávažnějším problémem VSM je tzv. *Lightbleeding*. Nastává v situacích kdy rozptyl σ^2 bývá velmi velký [5], běžně ve scénách s velkou hloubkovou rozmanitostí. Projevuje se viditelnými artefakty na místech zastíněných oblastí, které by měly být součástí (často umbry) stínu, ale jsou nekorektně osvětleny 2.6.

[10] Dotatečne je výhodné zapnout anizotropní filtrování.



Obrázek 2.6: Lightbleeding.

2.3.3 Vylepšení

V [5] popisuje spoluautor VSM Andrew Lauritzen metody pro zlepšení výsledků této techniky.

První je způsob reprezentace hloubkových rozsahů na texelech variantní stínové mapy pomocí druhého momentu, která elegantně řeší problém biasingu, typický pro PCF.

$$M_2 = \mu^2 + \frac{1}{4}\left(\left[\frac{\partial z}{\partial x}\right]^2 + \left[\frac{\partial z}{\partial y}\right]^2\right) \tag{2.9}$$

Pokud jsou parciální derivace rovny nule, je pravá strana redukována na μ^2, tedy původní druhá mocnina střední hodnoty hloubky. To se děje v situacích, kdy je povrch paralelně s projekční rovinou světla. Vhodné je také limitovat parciální derivace, pokud je povrch okluzoru rovnoběžný se směrem dopadajícího světla, čímž se lze vyhnout potencionálně vysokým rozptylům. Povrchové akné se ale dá efektivně eliminovat jen použitím minimálního rozptylu při konstrukci stínů.

Další vyžadovanou modifikací je redukce Lightbleeding artefaktů. Ty se nejvíce projevují, když je rozdíl hloubek vícero za sebou jdoucích okluzerů velmi vysoký. Důležitým poznatkem je, že pokud je povrch v hloubce t plně zastíněn v rámci filtrovaného regionu se střední hodnotou M_1, pak $t > M_1$ a $(t - M_1)^2 > 0$. p_{max} (2.7) bude potom vždy menší než 1. Z toho vyplývá, že nekorektní penumbra nikdy nedosáhne plné intenzity. Toto lze využít aplikováním dolní meze pro p_{max}, čímž určité hodnoty pod touto mezí budou mapovány na 0, tzn. kompletně ve stínu, a ostatní hodnoty škálovány tak, aby byli mapovány od 0 k 1. V algoritmu 2 nastavuje první parametr agresivitu potlačení Lightbleedingu. H bývá

v případě stínování maximální možná intenzita světla.

Algoritmus 2: LinStep

Input: dolní mez L, horní mez H, ovlivňovaná hodnota v
Output: intenzita I

1: $I \leftarrow (v - L)/(H - L)$
2: $I \leftarrow \text{clamp}(I, 0, 1)$
3: **return** I

Referovaná kapitola [5] ještě dodává, že filtrování lze ulehčit použitím *Summed-Area Table* (SAT), sestavenou z elementů $a[i, j]$ zdrojové textury jako:

$$t[i, j] = \sum_{x=0}^{i} \sum_{y=0}^{j} a[x, y] \tag{2.10}$$

Každý $t[i, j]$ je sumou vstupních texelů nad a vlevo od tohoto elementu. Použitím SAT jde zjemnit stíny při průměrování, resp. distribuovat momenty ve variantní stínové mapě, čímž lze omezit numerickou nestabilitu, pokud by se jí nedalo vyhnout. Takto před-filtrovaná textura produkuje gradient při bilineární interpolaci a přispívá k redukci Lightbleedingu.

2.4 Convolution Shadow Maps

Klíčem postupu této techniky (CSM) je zakódování binární funkce (2.1) do samostatných funkcí, které jsou použity k aproximaci viditelnosti [2]. Ty mohou být zase filtrovány libovolnými konvolučními metodami grafické karty. CSM konvergují k reálným výsledkům mnohem lépe než VSM 2.3 a komplexnost scény nemá tak velký vliv na případné artefakty. Navíc poskytuje možnost simulace měkkých stínů při aplikováním širokého konvolučního filtru, nicméně rozsah penumbry zůstává poměrně stejný.

2.4.1 Algoritmus

Stínovací test označen jako

$$s(\mathbf{x}) = f(d(\mathbf{x}), z(\mathbf{p})) \tag{2.11}$$

je závislý na proměnných d a z. První z těchto proměnných je udávána bodem $\mathbf{x} \in \mathbb{R}^3$. $\mathbf{p}$ je získáno projekcí $T\colon \mathbb{R}^3 \to \mathbb{R}^2$ podle převodu $\mathbf{p} = T(\mathbf{x})$ do prostoru hloubkové mapy. Aby bylo možné získat při konstrukci anti-aliasované stíny, je nutné filtrovat $s(\mathbf{x})$:

$$s_f(\mathbf{x}) = \sum_{\mathbf{q} \in \mathcal{N}} w(\mathbf{q}) f(d(\mathbf{y}), z(\mathbf{p} - \mathbf{q})) \tag{2.12}$$

Rovnice (2.12) obsahuje novou proměnnou $\mathbf{y}$. Jedná se o bod ležící blízko $\mathbf{x}$ (definováno $T(\mathbf{y}) = \mathbf{p} - \mathbf{q}$, kde $\mathbf{q}$ je sousedící texel) a je významný při konvoluci. Neexistuje však inverzní převod zpět na $\mathbf{y}$, protože se s ním pracuje jen v prostoru hloubkové mapy [2]. Z toho vyplývá předpoklad $d(\mathbf{x}) \approx d(\mathbf{y})$ nutný při dalším odvozování.

$$s_f(\mathbf{x}) = [w * f(d(\mathbf{x}), z)](\mathbf{p}) \tag{2.13}$$

Znamená to, že $d(\mathbf{x})$ je reprezentativní vzdáleností pro celé okolí $\mathcal{N}$, což sice neplatí tak často (pouze rovinné útvary), ale postačuje k uspokojivé aproximaci [2].

Jelikož není možné přímo aplikovat filtrování na $z(\mathbf{p})$ — takáto konvoluce není ekvivalentní s filtrováním výsledků porovnávací funkce (viz vysvětlení povahy Shadow Mapping algoritmu v sekci 2.1.2) — je nezbytné použít transformaci z hodnot, načež $f(d, z)$ bude výsledkem sumy a porovnávací test vypadá následovně:

$$s(x) = \sum_{i=1}^{N} a_i(d(\mathbf{x}))B_i(z(\mathbf{p})) \tag{2.14}$$

Expanze v (2.14) je zkrácena podle N a je lineární vzhledem k množině $B_{i=1\ldots N}$. Aby bylo možné použít tuto řadu, je hloubková mapa konvertována na tzv. *bázové obrazy*, označeny právě jako B_i. Tato řada se pak dosadí do konvoluce v (2.13), viz kapitolu 3 [2].

$$\begin{aligned} s_f(\mathbf{x}) &= [w * \sum_{i=1}^{N} a_i(d(\mathbf{x}))B_i](\mathbf{p}) \\ &= \sum_{i=1}^{N} a_i(d(\mathbf{x}))[w * B_i](\mathbf{p}) \end{aligned} \tag{2.15}$$

Je vidět, že libovolná konvoluční operace na stínovací funkci je ekvivalentní s konvolucí individuálních bázových obrazů $B_i(z(\mathbf{p}))$. Takovýmto způsobem (osamostatněním $d(\mathbf{x})$ a $z(\mathbf{p})$) lze umožnit filtrování stínovací funkce. Úspěšně je toho docíleno využitím Fourierovy řady [2].

Porovnávací funkce f může být reprezentována jako Heavisideova funkce $H(t)$ následovně: $f(d, z) = H(d - z)$. Požadavkem při aplikování Fourierovy řady je periodičnost. Autoři CSM toho docílili definováním obdélníkové funkce $S(t)$ s periodou $2 \Rightarrow t \in (-1, 1)$. Dostáváme $H(t) = \frac{1}{2} + \frac{1}{2}S(t)$. Fourierova řada pro stínovací funkci se potom stává

$$f(d, z) \approx \frac{1}{2} + 2\sum_{k=1}^{M} \frac{1}{c_k} \sin[c_k(d - z)] \tag{2.16}$$

s $c_k = \pi(2k - 1)$ (viz sekci 3.1 [2]). Pomocí trigoniometrické identity je pak suma převedena na:

$$f(d, z) \approx \frac{1}{2} + 2\sum_{k=1}^{M} \frac{1}{c_k} \cos(c_k d) \sin(c_k z) - 2\sum_{k=1}^{M} \frac{1}{c_k} \sin(c_k d) \cos(c_k z) \tag{2.17}$$

Nyní zpět na (2.14), je vidět, že termy jde osamostatnit podle závislosti na d, resp. z.

$$\begin{array}{ll} a_{(2k-1)}(d) = \frac{2}{c_k} \cos(c_k d) & B_{(2k-1)}(z) = \sin(c_k z) \\ a_{(2k)}(d) = \frac{-2}{c_k} \sin(c_k d) & B_{(2k)}(z) = \cos(c_k z) \end{array} \tag{2.18}$$

Výhodou takto upravené řady[11] je, že bázové obrazy se nachází v intervalu $\langle -1, 1 \rangle$, který nevyžaduje širokou aritmetiku (postačujících je 8 bitů).

Implementace samotná požaduje určité množství paměťového prostoru. Závisí to na zvolené délce řady pro aproximaci porovnávací funkce. Z uchování bázových obrazů při $M = 16$ vyplývá použití osmi 4-kanálových textur. OpenGL poskytuje možnost *Multiple-Render Targets*; renderování do několika textur zároveň, tedy postačuje jedno dodatečné vykreslení scény (znovu z pozice světla). Jednotlivé kroky jsou popsány následovně:

[11] Pozn.: $k = 1 \ldots M$, $N = 2M$, $\frac{1}{2}$ je připočtena samostatně.

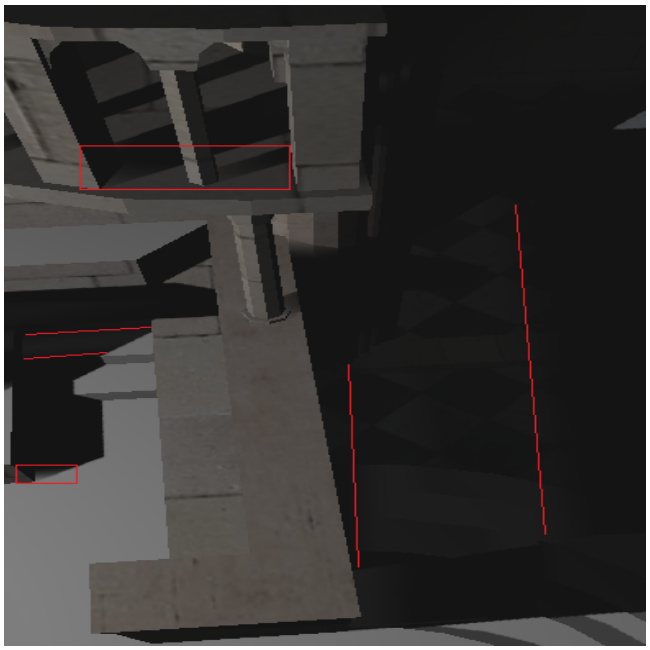
1. Po sestavení hloubkové mapy se z hodnoty konvertují na odpovídající bázové obrazy (viz rovnice (2.18)) v druhém render průchodu. Lepší přesnosti výsledných B_i lze dosáhnout použitím lineárního z-bufferingu.

2. Jakmile byla sada bázových obrazů vypočtena, na odpovídající textury se může aplikovat filtrování. To zahrnuje mipmapping a rozmazání, případně lze zapnout pokročilejší techniky při konstrukci stínů pro potlačení výskytu ostrých hran.

3. Ve fázi vykreslování stínované scény jsou dopočítány koeficienty a_i dle aktuální hloubky d zkoumaného bodu, načež je pak výsledek testu aproximován rovnicí (2.15).

2.4.2 Nedostatky

Fourierova řada je náchylná ke zvonění[12], především když je zkrácena na malý počet termů M [2]. To má za následek přetečení funkce za rozsahy minimální a maximální intenzity světla. Produkuje to především světlé pruhy (obr. 2.7) pokrývající (většinou) celou šířku stínu. Také kontaktní místa okluzoru a přijímatele jsou ovlivněna.



Obrázek 2.7: Zvonění a změna intenzity na kontaktních místech.

Dalším záporem při použití Fourierovy řady pro vyhodnocení schodové funkce je pomalý přechod mezi výslednými intenzitami. Tzn. pro plně osvětlené povrchy ($d \approx z$) se stínovací funkce vyhodnotí jako 0,5. Vyúsťuje to v nižší jas v celé scéně [2].

CSM strácí hlavně výkonově. Od délky Fourierovy řady se odvíjí počet použitých textur nesoucích báze B_i. Dopad lze vidět u větších scén, kde je potřeba vykreslovat obrovské množství primitiv. Výpočet termů pak trvá jistý čas, nemluvě o generaci mipmap B_i v každém průchodu. Omezení M ale vytváří vhodný kompromis: cenou je více možných artefaktů.

[12] Známo též jako Gibbsův jev.

2.4.3 Vylepšení

Redukce zvonění bývá dosažena [2] útlumem každého k-tého termu: $e^{-\alpha\left(\frac{k}{M}\right)^2}$. Sílu útlumu kontroluje parametr α[13]. Použitím této metody se však třeba dopustit kompromisu – výsledná funkce je pak totiž více strmá a tedy přechody mezi zastíněnou a nezastíněnou oblastí bývají méně jemné.

Jev 50% zastínění všech povrchů je potlačen jednoduchým ofsetováním Heavisideovy funkce [2]. Potom při $d \approx z$ je výsledek korektně na (přibližně) maximální hodnotě intenzity světla. Posunutí se ale negativně projevuje na kontaktních místech, kde nastává zvýšení jasu. Škálováním s_f možno dosáhnout podobné výsledky, ale lze pozorovat návrat aliasingu [2]. Běžně stačí strmost funkce upravit násobením dvou. Poté musí být hodnota limitována v oboru hodnot viditelnosti, tedy $\langle 0, 1 \rangle$.

Dostatečně velké M potlačuje artefakt nesprávné intenzity stínu na kontaktních místech. Ve většině případech stačí $M = 16$.

V rámci práce na projektu byla vyzkoušena modifikovaná verze tohoto algoritmu. Původní navrhovaná implementace popisuje vytváření B_i dodatečným vykreslením scény a uložením vypočtených bázových obrazů do speciálních textur, kterých počet je $\frac{M}{2}$. Velikost zabrané paměti a zápisu do ní drasticky vplývá na rychlost celé aplikace, především kvůli použití trilineárního filtrování, mající za následek vygenerování mipmap v každém cyklu. Přesunutím výpočtu termů B_i do shaderu, kterému jsou konceptuálně na vstupu, se lze vyhnout obsazení většího adresního prostoru na grafické kartě a tím také šetřit dostupnými prostředky.

2.5 Exponential Shadow Maps

Exponenciální stínové mapy (ESM) [3] byly poprvé představeny jako algoritmus inspirovaný konvoluční variantou 2.4 Shadow Mappingu. Narozdíl od ní používají k aproximaci pouze jediný term. Tato funkce stojí na předpokladu, že hloubky bodů filtrovaného regionu zpracováváného fragmentu při konstrukci stínů neleží mimo rozsah vzdáleností uložených v textuře z hodnot. Faktem je, že ve většině případů tento požadavek není narušen. Pokud ano, je pro tyto fragmenty zvolen vlastní PCF kernel. Naštěstí takových případů nebývá mnoho.

ESM nabízejí velmi dobré výsledky za opravdu minimální cenu jak výpočtového času, tak oproti CSM i potřebné paměti. Zároveň tolik netrpí artefakty typickými pro předchozí techniky. Principem stále zůstává využití hardwarových možností.

2.5.1 Algoritmus

Algoritmus je konceptuálně podobný CSM tím, že se snaží aproximovat porovnávací test použitím řady. Je založen na jednoduchém postřehu, vztahujícím se ke klasické porovnávací funkci $f(d, z)$: existuje-li bod $\mathbf{x}$, viditelný z pozice světla, je jeho hloubka dána jako d – potom d musí být větší nebo rovno z-tové hodnotě uložené v hloubkové mapě, protože jsou v ní uchovány jen hloubky objektů nejblíže světlu [3]. Logicky tedy platí nerovnost $d(\mathbf{x}) - z(\mathbf{p}) \geq 0$, kde $\mathbf{p}$ je texel na souřadnicích daných $\mathbf{x}$ po projekci do 2D. V praxi se však setkáváme se situacemi kdy toto neplatí a to nejen kvůli číselným nepřesnostem textury.

[13] Typicky postačuje $\alpha = 1$.

Za předpokladu, že $d > z$, definuje se porovnávací funkce $f(d, z)$ jako,

$$f(d, z) = \lim_{\alpha \to \infty} e^{-\alpha(d-z)} \tag{2.19}$$

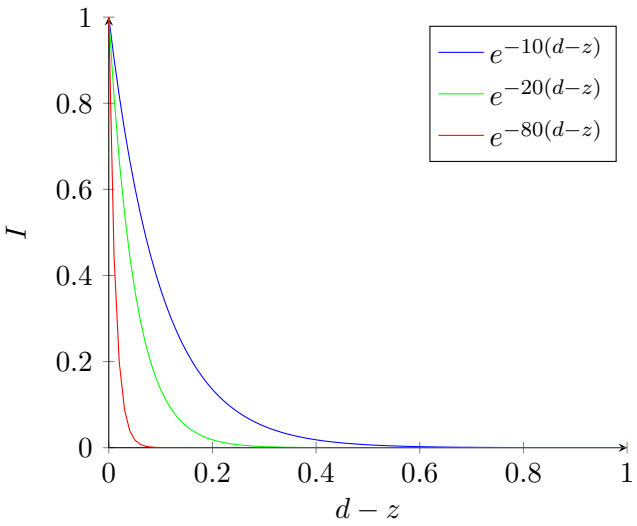
která může být aproximována po nahrazení α konstantou c, reprezentující dostatečně velké číslo. Pravou stranu lze rozdělit na nezávislé části,

$$f(d, z) = e^{-cd} e^{cz} \tag{2.20}$$

aby před aproximací $s_f(\mathbf{x})$ (2.21) bylo možné z hodnoty předfiltrovat některým konvolučním kernelem w [3]:

$$s_f(\mathbf{x}) = e^{-cd(\mathbf{x})}[w * e^{cz}] \tag{2.21}$$

Volba správné c je velmi důležitá, protože od ní se odvíjí aproximace. Podle sekce 3.1 [3] se pro 32-bitové kanály volí c = 80. V případě aritmetiky 16 bitů je c = 30. Efekt lze pozorovat na grafu 2.8.

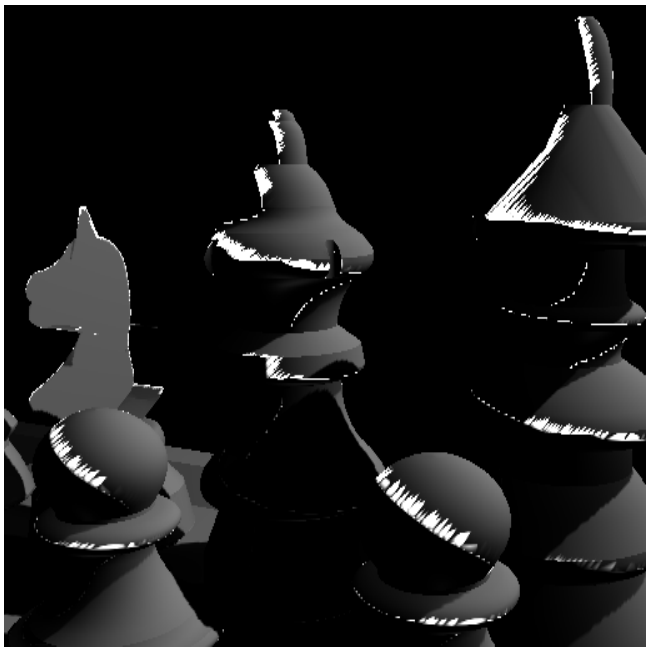


Obrázek 2.8: Graf exponenciální stínovací funkce: použitá konstanta c prudce zvyšuje strmost funkce.

2.5.2 Nedostatky

Jediným značným záporem ESM je, že staví na předpokladu hloubky bodů zpracovávaného regionu v intervalu daném hodnotou uloženou v textuře $\Rightarrow d \in \langle z, \infty)$ [3]. Problém nastává především při bilineárním filtrování, jak je názorně vyobrazeno obr. 2.9. Hloubkový rozdíl může být definován jako $\Delta_{\mathbf{x}} = d(\mathbf{x}) - z(\mathbf{p})$. Pokud $\Delta_{\mathbf{x}} < 0$, pak s_f vrací velká čísla, protože tato exponenciální funkce nekonverguje k 1. Výsledky jsou ovlivněny podle použití filtrování. V případě nearest neighbor filtrování nemusí předpoklad platit vzhledem na omezení hloubkové mapy a to jak numerické tak prostorové (mapování na šikmé povrchy). Tyto problémy u výchozího algoritmu způsobují již dobře známé akné. Pro ESM to má za následek přetečení funkce s_f, což lze řešit horní mezí nastavenou na 1. Během filtrování mohou nastat situace, kdy okolité texely budou obsahovat hloubku větší než referovaná hloubka d. Použití horní meze nepřichází v úvahu, protože by se tím porušil princip ESM, jelikož by se musely texely vzorkovat jako u PCF. Naštěstí však bývají zkoumané d hodnoty větší. Během

vzorkování bodů, které se nacházejí dále od okrajů stínů platí $\Delta_{\mathbf{x}} \geq 0$. Toto je typicky velmi časté protože většina fragmentů scény je buď plně osvětlena nebo ve stínu. Dokonce i v případech kdy je podmínka porušena, nemusí být tento jev patrný. Kupříkladu, šikmé nezastíněné povrchy mohou být vzorkovány nad uloženou z hodnotu. Nicméně, protože má být povrch osvětlen, limitováním maximální intenzity je tento problém úplně eliminován.



Obrázek 2.9: Porušení předpokladu: má za následek přeteční funkce s_f. Projevuje se nekonzistentními místy o vyšší intenzitě.

Zde je ještě dobré upozornit, že dodatečné před-filtrování má za následek komplikování úspěšného ESM testu [3]. Čím větší je filtrační kernel, tím pro vícero bodů běžně neplatí nutný předpoklad pro jeho vykonání a uchyluje se ke klasické filtrační technice jako například bilineární interpolace $f(d, z)$.

2.5.3 Vylepšení

Potížím popsaným v 2.5.2 je možné se vyhnout klasifikací porušení předpokladu $\Delta_{\mathbf{x}} \geq 0$. Pokud je fragment zařazen mezi nevalidní, uchýlí se algoritmus k upravené PCF metodě zjemňování stínů. Autoři ESM navrhli dva způsoby klasifikování těchto situací [3]: *z-max* klasifikace a práhování.

Z-max klasifikace závisí na dodatečné textuře, ve které je uchovávana maximální z-tová hodnota sousedství daného bodu – z běžných zaznamenaných hloubek specifický filtr vytvoří mipmap strukturu, uchovávajíc maximální hodnoty mipmappovaného okolí. Ověřením $d(\mathbf{x}) < z_{max}$, lze klasifikovat zda-li je pro daný fragment předpoklad platný. Je doporučeno polygonové ofsetování $d(\mathbf{x})$ kvůli možné špatné klasifikaci osvětlených povrchů.

Druhou možností je použití jednoduchého práhování. Výsledek funkce s_f je testován, zdali překračuje $1+\epsilon$, kde ϵ je zvolený práh, čímž se indikuje, že došlo k porušení podmínky. Pokud k tomu dojde, zprůměrují se výsledky ze sousedství se současnou hodnotou, podobně jako u PCF.

Porovnáním nabízených možností se dá dojít k závěru, že z-max klasifikace, i když poskytuje lepší výsledky, je horší metodou [3], načež potlačuje potenciál ESM jako výkonného algoritmu pro aproximaci stínů. Práhování totiž jeví přibližně stejné výsledky, nicméně může docházet k nesprávné klasifikaci: nelze totiž pro všechny pixely filtračního kernelu určit, zda-li pro ně platí diskutovaný předpoklad, poněvadž pouze výsledná hodnota je kontrolována.

Kapitola 3

Demonstrační aplikace

Popsání jednotlivých technik v předchozích kapitolách mělo za cíl uvést čtenáře do problematiky s dostatečným porozuměmím o tom, jak každá metoda funguje, jaké jsou její nedostatky a co lze udělat pro zlepšení. Cílem demonstrační aplikace je prezentovat praktické využití probíraných algoritmů a poskytnout uživateli dostatečnou zpětnou vazbu, na základě které bude schopen zvolit si vyhovující variantu. Program má za úkol na testovací scéně prokázat uváděné předpoklady, vztahujícími se k daným technikám, t.j. představení artefaktů, zobrazit aktuální frame rate a samozřejmě vizuálně ukázat jak kvalitní jsou výsledné stíny.

3.1 Návrh

Demonstrační software *shadow* musí uživateli umožnit zhodnotit modifikace SM algoritmu PCF, VSM, CSM a ESM. Benchmark bude probíhat na komplexní scéně, která poskytne jistou zátěž, aby výsledky byly patrné již během renderování, pokud by výkon razantně klesl během použití některé z technik. Toto musí být možno okamžitě pozorovat, nejlépe za použití grafu, který bude reagovat na změny hladiny FPS. Detailnější informace pak program vypíše na standardní výstup, kde uživatel nalezne informace o průměrném frame rate u zvolené techniky. Interaktivita bude zprostředkována pomocí klávesnice a myši, aby byla možná volnost pohybu, pokud by uživatel chtěl opustit přednastavenou trasu a prohlédnout si artefakty. Také schopnost nastavovat pozici světla v podobném stylu, prostřednictvím přepínání ovládané kamery.

Samotná aplikace bude využívat OpenGL. Na straně procesoru bude vykonáván kód přeložený z jazyka C++. Pro zrychlení práce na projektu přibylo zahrnutí několika volně dostupných knihoven:

- GLFW[1] – vytvoření a správa kontextu, umožnění aplikovat ovládání pomocí klávesnice a myši,
- GLEW[2] – poskytuje přístup ke standardním funkcím OpenGL, které nemusí být součástí jistých operačních systémů,
- GLM[3] – užitečná sada datových typů, pro práci s vektory a maticemi (jako v GLSL),

[1] GLFW: http://www.glfw.org/
[2] GLEW: http://glew.sourceforge.net/
[3] GLM: http://glm.g-truc.net/0.9.6/index.html

- Assimp[4] – tato knihovna je schopna nahrát z mnoha formátů 3D modely,
- SOIL[5] – použitá při vytváření textur z obrázků uložených na pevném disku.

3.2 Stínovací techniky

Všechny varianty SM algoritmu je výhodné optimalizovat automatickým zastíněním fragmentů, kterých normála n míří odvráceným směrem od šíření světelných loučů r. Takýto test lze provést na skalární součin $n \cdot r$. Pokud je výsledek menší než nula, může být daný bod ihned považován za neosvětlený. Nejen zlepšení výkonu, ale i lepší stíny na površích téměř paralelně se směrem šíření světla jsou docíleny použitím této jednoduchého způsobu kontroly orientace vzhledem ke zdroji světla. Pro zlepšení dojmu byl zároveň implementován Phongův osvětlovací model.

Výsledek stínovacího algoritmu PCF je dán zprůměrováním výsledků funkce (2.1) jednotlivých bodů filtrovaného regionu, jehož velikost je určena šířkou K. PCF poskytuje ještě lepší stíny v případě použití bilineární interpolace mezi určenými texely. Detaily tohoto vylepšení a implementace techniky jako takové jsou k nalezení v souboru object-pcf.frag. Povrchové akné je eliminováno použitím konstantního Depth Bias.

VSM využívá texturu s interním formátem `GL_RG32F`, kde každý kanál má rozsah 32 bitů, kvůli zamezení numerické nestability, která se u této metody často projevuje. Do variantní hloubkové mapy jsou pomocí shaderu shadow-vsm.frag uloženy momenty z a z^2, reprezentující distribuci hloubek filtrovaného regionu. Jejím rozmazáním lze docílit měkkších stínů. Pro texturu je pak vytvořena mipmap struktura, aby šlo aplikovat trilineární filtrování. Ve fragment shaderu object-vsm.frag se pak intenzita světla vyvodí podle rovince (2.8). Potlačení Lightbleedingu probíhá podle algoritmu 2 a vypočtením parciálních derivací z při zápisu do variantní hloubkové mapy (popsáno rovnicí 2.9) je omezeno povrchové akné.

CSM začíná klasicky uložením hloubek nejbližších objektů (relativně ke zdroji světla) do hloubkové mapy. Na ni lze následně před generací termů B_i aplikovat Gaussove rozmazání pro efektnější odstranění ostrých hran. Tato textura je na vstupu shader programu, definovaného zdrojovým kódem v souborech basis.vert a basis.frag, kterého fragment shader generuje osm `vec4` výstupů (zvolené M je tedy 16). Ty jsou vloženy do `GL_RGBA16F` textur, nesoucích bázové obrazy B_i. Shader object-csm.frag je po vytvoření mipmap použije pro aproximaci $f(d, z)$ podle rovnice (2.17). Jevu zvonění je bráněno útlumem termů (viz sekci 2.4.3) a vypořádávání se s celkově nízkou intenzitou základní verze algoritmu je ošetřeno škálováním výsledné hodnoty s_f, t.j. násobením dvou, aby Fourierova řada konvergovala rychleji.

Vlastní modifikace XCSM vznikla metodou pokus-omyl, během implementačních potížích, týkajících se pole samplerů pro textury bázových obrazů. Pro otestování funkčnosti algoritmu se výpočet termů B_i přesunul do programu, majícím na starost konečné generování stínů. Ukázalo se, že funkčně jsou si obě varianty velmi podobné (CSM však stále poskytuje lepší filtrování, protože se interpoluje již mezi B_i, ne pouze z hodnotami). Pokud je zvolena dodatečná konvoluce (rozmazání), rozdíly nejsou téměř patrné. Co se týká výkonnostních rozdílů, je jasné, že zabránění použití několika textur navíc a dalšímu průchodu scénou se hladiny FPS velmi liší, viz kapitola 4.

První fáze ESM se od běžného Shadow Mappingu liší zápisem e^{cz} (shadow-esm.frag). Stejně jako u předchozích dvou technik se může využít dodatečné rozmazání textury před

[4] Assimp: http://assimp.sourceforge.net/
[5] SOIL: http://www.lonesock.net/soil.html

vytvořením mipmapy. object-esm.frag shader poté aproximuje stínovací funkci podle rovnice (2.21), načež kontroluje porušení předpokladu jednoduchým prahováním (sekce 2.5.3). Pokud podmínka není splněna, je výsledkem stínovací funkce výstup upraveného PCF, které interpoluje mezi výsledky čtyř nejbližších texelů.

Dodatečná konvoluce představuje filtrování Gaussovským 7×7 kernelem, kdy je nutná dočasná textura pro uchování mezivýsledku – nejprve se rozmazání provádí na x-ové ose, pak y-ové. Scéna se tedy musí znovu vykreslit dvakrát.

3.3 Vybrané implementační detaily

Zvolený objektově-orientovaný model rozděluje aplikaci do několika modulů. Třída `Renderer`, seskupující jejich funkčnost, má na starosti hlavní smyčku renderování a v podstatě celý chod aplikace. Při tvorbě byl kladen důraz na znovupoužitelnost mnohých tříd, v některých případech se rozhodlo pro použití dědičnosti. Zajímavá je především třída `Scene`, která obsahuje členskou metodu `renderAll`. Ta na své atributy, reprezentující vykreslovanou scénu, aplikuje pomocí polymorfizmu speciální techniky, předány parametrem jako seznam ukazatelů na typ `Technique`. Ukazatel dereferuje úplnou virtuální metodu této bázové třídy, čímž se volá odpovídající funkce, aplikující nějakou vlastnost renderování, ve většině případů to znamená nahrání nějaké uniformní jednotky do momentálně používaného shader programu. Tímto způsobem lze jednoduše přepínat mezi implementovanými stínovacími technikami. V konstruktoru `Scene` jsou objekty inicializovány prostřednictvím knihovny Assimp, vytvářejíc uzly `Node`, ze kterých scéna sestává. Jistěže se ověřuje, zda-li některá data už nebyla předtím nahrána, což se může stát u materiálů jako textury.

Techniky předvedeny při průletu scénou mají zaznamenávány dosažené hladiny FPS v seznamu třídy `Benchmark`, která z těchto hodnot sestavuje graf, vykreslovaný v pravém horním rohu. Když uživatel přepne na jinou stínovací metodu nebo ukončí program, na standardní výstup je vypsán průměrný frame rate i frame time. Z těchto prezentací výkonu si dokáže odvodit a velmi pravděpodobně již za běhu představit, která varianta je rychlejší.

Algoritmy diskutovány v předchozích kapitolách představují instance odvozené od dříve zmíněné třídy `Technique`. Také dědí z další třídy zvané `Offscreen`, zapouzdřující implementaci framebufferu, používaného při vykreslování.

Průběh pohybování (neovládané) kamery pozorovatele je definován interpolací mezi tzv. *keyframe* daty, nacházejících se ve speciálních souborech složky keyframes/. Při vykonávání programu lze kdykoliv stisknout klávesu `K`, která uloží na konec odpovídajícího[6] souboru informace o vektorech nutných k sestavení matice pro transformaci pozic vrcholů do souřadného systému kamery. Celá implementace se nachází ve třídách `Camera` a `Spectator` (tato je odvozena od první jmenované).

Nezbytně nutné detaily pro použití lze dohledat v souboru README.

[6] Kamery jsou číslovány od 1 a první představuje kameru pozorovatele. Podle pořadí se odvozují názvy keyframe souborů.

Kapitola 4

Výsledky a porovnání

Testování proběhlo na platformě Windows 7. Počítač sestával z grafické karty AMD Radeon HD 6630M s 1 GB pamětí, 4 GB RAM a CPU Intel® Core™ i5-2430M (2,40 GHz, 4 logické procesory). Aplikace byla přeložena kompilátorem G++ (MinGW-w64).

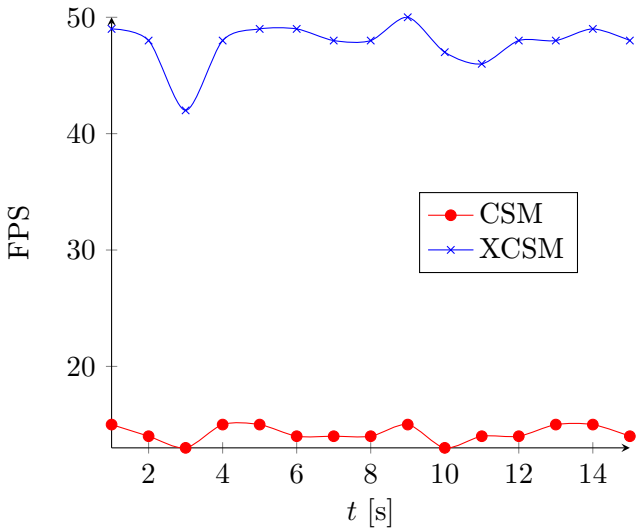
R	**H**	PCF		VSM	CSM	XCSM	ESM
		3	5				
640 × 480	256^2	63	59	62	21	60	62
	512^2	59	55	59	19	58	60
	1024^2	56	47	55	16	55	56
	2048^2	50	35	53	15	53	53
800 × 600	256^2	60	54	60	20	59	60
	512^2	58	51	57	18	57	58
	1024^2	54	43	54	16	53	54
	2048^2	47	31	52	15	52	52
1024 × 768	256^2	55	46	57	20	57	58
	512^2	53	43	53	18	53	54
	1024^2	49	37	52	17	51	52
	2048^2	41	27	50	15	48	50

Tabulka 4.1: FPS statistika: **R** označuje rozlišení zobrazovacího zařízení a **H** rozsah hloubkové mapy. Ve sloupci PCF vyjadřují čísla druhého řádku šířku kernelu K. Technika XCSM je mnou upravená verze původní CSM, jak bylo popsáno v posledním odstavci sekce 2.4.3.

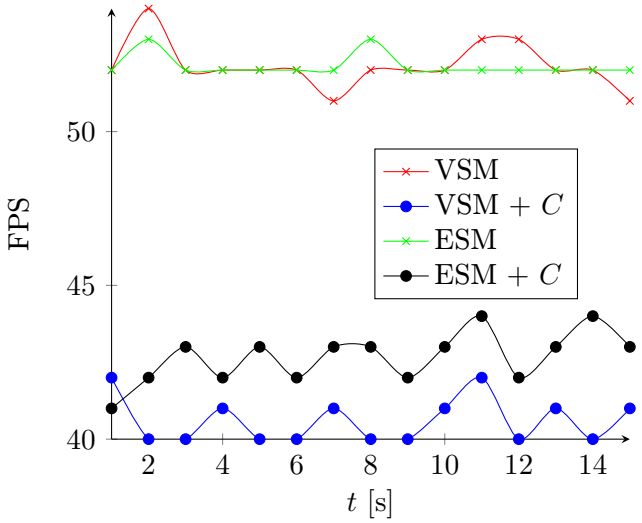
H	PCF	VSM		CSM		XCSM		ESM	
			C		C		C		C
256^2	0,342	0,683	+0,512	5,720	+0,512	0,342	+0,256	0,342	+0,256
512^2	1,366	2,731	+2,048	22,872	+1,024	1,366	+ 1,024	1,366	+ 1,024
1024^2	5,462	10,923	+8,192	91,480	+4,096	5,462	+4,096	5,462	+4,096
2048^2	21,846	43,691	+32,768	365,912	+16,384	21,846	+16,384	21,846	+16,384

Tabulka 4.2: Tabulka využité paměti (v MB): Rozlišení textury je dáno prvním sloupcem. Každá technika používá mipmapping. C indikuje pomocnou texturu během konvoluce.

Z tabulky 4.1 by měly být patrné výkonnostní rozdíly porovnávaných technik. Oproti ostatním metodám se PCF $K = 3$ z hlediska výkonu ukázala jako výhodnější varianta, pokud druhé způsoby filtrování používají dodatečné průchody pro rozmazání textury (detaily v tabulce 4.3 a ukázka na grafu 4.2). To sice produkuje lépe vypadající stíny, především u nižších rozlišení hloubkové mapy, ovšem negativně vplývá na celkový výkon i u šetrných algoritmů jako VSM a ESM. Avšak jestli se rozhodne tuto konvoluci vypustit, poskytují spomenuté stínovací techniky mnohem lepší výsledky, dokonce i v případě rozlišení hloubkové mapy $\mathbf{H} = 2048^2$. Vyšší rozlišení zároveň potlačuje nepřijemný jev perspektivního aliasingu, tedy dodatečné filtrování pomocí C není za použití trilineárního a anizotropního nutné. Také Lightbleeding u VSM je tímto částečně eliminován. PCF naneštěstí trpí zjevným PP, které je asi celkově nejhůře vypadajícím artefaktem. Velmi zajímavé je porovnání CSM a XCSM, viz graf 4.1. CSM jako takové je, co se hardwarových nároků týče, evidentně nejméně doporučeným algoritmem, protože oproti ostatním metodám je nejenže více paměťově náročné (tabulka 4.2), ale pokles hladiny FPS je až drastický. Jeho asi jedinou výhodou je, že poskytuje opravdu dobře vypadající stíny. Aplikací XCSM možno zbavit tuto metodu výše popsaného problému náročnosti na zdroje počítače. Daní jsou trochu více viditelné artefakty aliasingu, avšak víceméně lze tímto dosáhnout podobně kvalitní výsledky. Z daného sloupce 4.1 se dá hned usoudit, že výkonem se blíží VSM i ESM. Posledně jmenovaný je opravdu levným algoritmem s postačujícím anti-aliasingem, i když někdy šlo pozorovat artefakty prosakující intenzity, kterým se dá zamezit ohlídáním porušení předpokladu $d \geq z$. Kvalitativní rozdíly pak nejlépe vidět na obrázích 4.3.



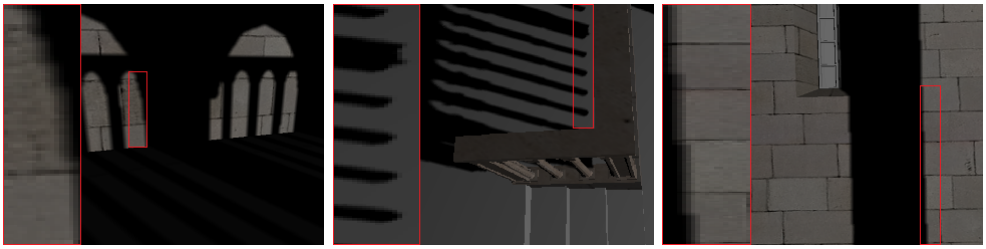
Obrázek 4.1: Výkonnostní rozdíl CSM a XCSM na časovém úseku při renderování stejné scény: hloubková mapa s rozsahem 1024^2 a použito rozmazání.



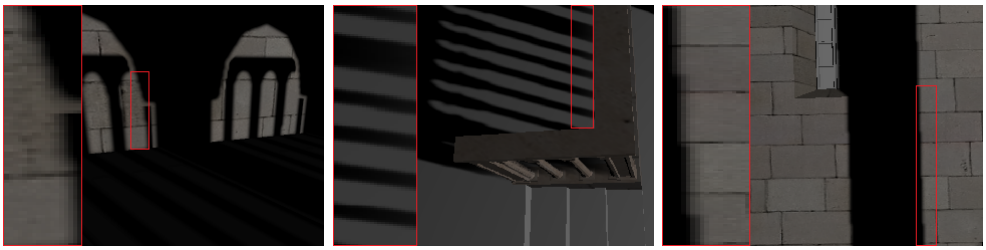
Obrázek 4.2: Výkonnostní rozdíl VSM a ESM s a bez konvoluce C, kde $\mathbf{H} = 1024^2$.

R	**H**	VSM	CSM	XCSM	ESM
640×480	256^2	50	19	48	50
	512^2	49	18	48	47
	1024^2	46	15	44	45
	2048^2	30	14	38	40
800×600	256^2	48	19	47	48
	512^2	46	17	48	47
	1024^2	46	15	46	46
	2048^2	30	13	38	40
1024×768	256^2	46	19	44	46
	512^2	44	16	43	44
	1024^2	41	14	42	42
	2048^2	29	12	35	37

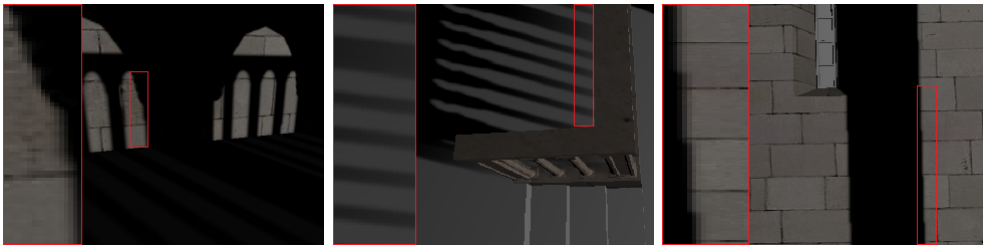
Tabulka 4.3: Dopad použití Gaussovského filtru rozmazání textury: jeho zahrnutí způsobuje přibližně 10-20% ztrátu výkonu. Jedná se o dva dodatečné render průchody, kdy se do dočasné textury uloží obsah hloubkové mapy rozmazaný v jednom směru, a následně se konvoluje v dalším, jehož výstup je vložen do původní (zdrojové) textury.



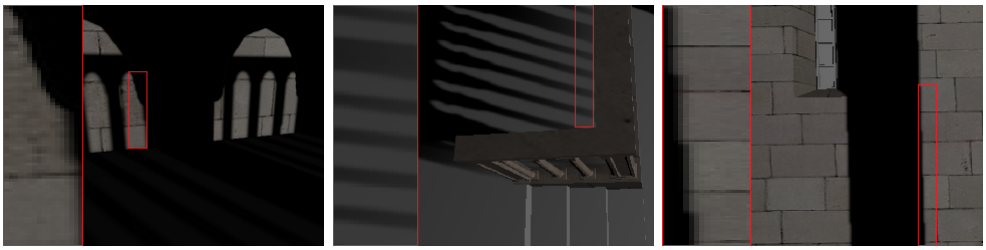
a) PCF



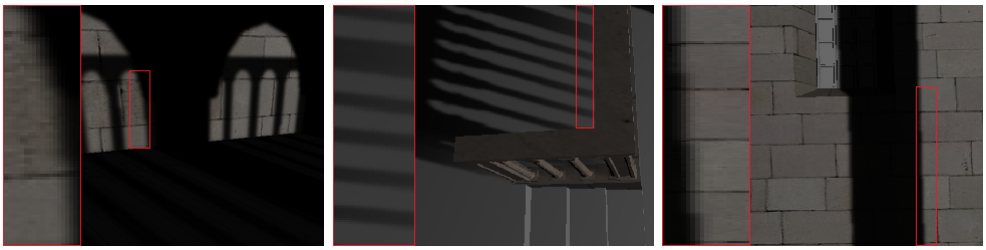
b) VSM



c) CSM



d) XCSM



e) ESM

Obrázek 4.3: Kvalitativní srovnání nabízených technik: rozlišení textur zvoleno 1024^2. S výjimkou PCF ($K = 3$) je u všech ostatních metod použito rozmazání 7×7 Gaussovým filtrem. U PCF si lze všimnout PP. VSM dává najevo svůj nejzávažnější nedostatek, t.j. Lightbleeding. Nekorektní intenzitu na kontaktních místech mají všechny techniky kromě PCF vzhledem k aproximační povaze funkcí společnou.

Kapitola 5

Závěr

Byl představen základní algoritmus generování stínů v 3D grafike a některé techniky, sloužící k potlačení popsaného aliasingu a sebe-zastínění. Ukázalo se, že nejefektivnější způsoby vytváření pěkných stínů jsou v tomto průzkumu VSM a ESM. Do všech je možné zakomponovat nějaké vylepšení pro zbavení se specifických artefaktů, tudíž z hlediska kvality se nachází na přibližně stejné úrovni jako CSM, které produkují nejhezčí výsledek, i když za cenu vysokých nároků na zdroje počítače. Volba filtrační metody tedy závisí víc na hardwarových požadavcích, minimálně pokud jsou stíny renderovány v real-time aplikacích.

Tato práce podrobně analyzovala, uvedla nedostatky a popsala dříve navržená vylepšení, a zhodnotila nabízené varianty Shadow Mapping algoritmu. Také se během implementace přišlo na způsob vylepšení CSM, který poskytuje přibližně stejnou kvalitu řešení filtrace stínů, ovšem s omnoho nižšími paměťovými a výpočetními nároky (viz kapitolu 4) – modifikace je podobně náročná jako právě doporučený algoritmus ESM.

Samotný demonstrační program má několik nedostatků. Především implementace CSM, protože používá 16-bitové kanály textur k uchování B_i, jelikož v OpenGL je k dispozici pouze 8-bitová varianta s normalizovanou aritmetikou. Hodilo by se tedy využít nějaký způsob zakódování/dekódování termů. U metody PCF by bylo vhodné změnit typ sampleru na `shadow2DSampler` (kvůli vestavěné bilineární interpolaci), i když podobný výsledek zprostředkovává vlastní GLSL funkce. VSM může být vylepšeno o SAT (viz sekci 2.3.3), aby se více omezil častý Lightbleeding.

Dalším směrováním projektu by mohlo být studium problematiky vytváření přesných měkkých stínů, tedy řešit generování korektní umbry/penumbry. Již existují techniky postavené na představených algoritmech a z hlediska výkonu i kvality by mohlo dojít na zajímavé porovnání.

Literatura

[1] AKENINE-MÖLLER, T.; HAINES, E.; HOFFMAN, N.: *Real-Time Rendering*. Natick, MA, USA: A.K. Peters, třetí vydání, 2008, ISBN 978-1-56881-424-7.

[2] ANNEN, T.; MERTENS, T.; BEKAERT, P.; aj.: Convolution Shadow Maps. In *EGSR '07 Proceedings of the 18th Eurographics Conference on Rendering Techniques*, Aire-la-Ville, CH: Eurographics Association, 2007, ISBN 978-3-905673-52-4, s. 51–60, dostupné z http://web4.cs.ucl.ac.uk/staff/j.kautz/publications/csmEGSR07.pdf.

[3] ANNEN, T.; MERTENS, T.; SEIDEL, H.-P.; aj.: Exponential Shadow Maps. In *GI '08 Proceedings of Graphics Interface 2008*, Toronto, CA: Canadian Information Processing Society, 2008, ISBN 978-1-56881-423-0, s. 155–161, dostupné z http://www.cad.zju.edu.cn/home/jqfeng/papers/Exponential%20Soft%20Shadow%20Mapping.pdf.

[4] DONNELLY, W.; LAURITZEN, A.: Variance Shadow Maps. In *I3D '06 Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, New York, NY, USA: ACM Press, 2006, ISBN 1-59593-295-X, s. 161–165, dostupné z http://www.punkuser.net/vsm/.

[5] LAURITZEN, A.: *GPU Gems 3*, kapitola 8. Summed-Area Variance Shadow Maps. Addison-Wesley, 2008, ISBN 0-321-51526-9, dostupné z https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_pref01.html.

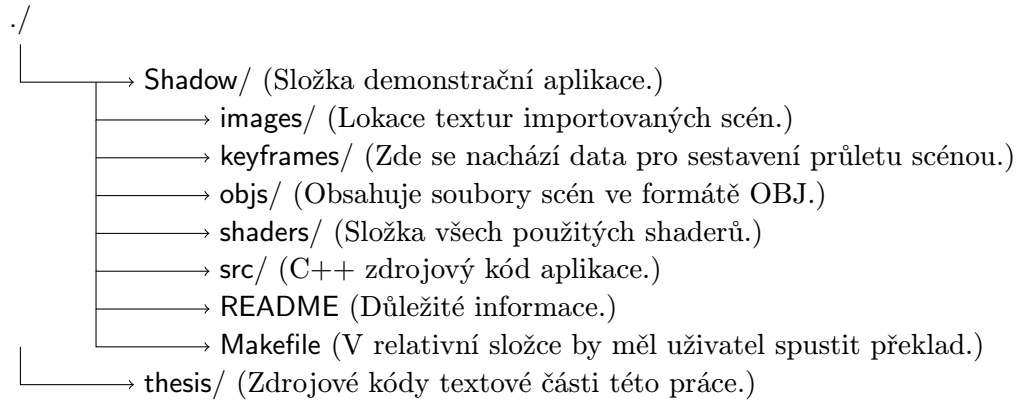
[6] REEVES, W.; SALESIN, D.; COOK, R.: Rendering Antialiased Shadows with Depth Maps. In *SIG-GRAPH '87 Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA: AMC Press, 1987, ISBN 0-89791-227-6, s. 283–291, dostupné z http://www.eng.utah.edu/~cs5610/handouts/reeves87.pdf.

[7] WILLIAMS, L.: Casting Curved Shadows on Curved Surfaces. In *SIGGRAPH '78 Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA: AMC Press, 1978, s. 270–274, dostupné z http://www.cs.berkeley.edu/~ravir/6160/papers/p270-williams.pdf.

Příloha A

Obsah CD

Přiložené CD poskytuje zdrojové soubory demonstrační aplikace *Shadow* a textu bakalářské práce. Pravidla modifikace/distribuce specifikuje přiložená licenční smlouva.



Obrázek A.1: Strom adresářové struktury, popisující pouze nejdůležitější části.

Příloha B

Manuál

Programování proběhlo na operačním systému Windows 7, s kompilátorem G++ (MinGW-w64[1]). Uživatel nechá přeložit poskytnutý zdrojový kód programem Make. Předtím by však měl změnit určité proměnné v souboru Makefile, aby překladač vědel, kde hledat zahrnuté hlavičkové soubory/knihovny. Jedná se konkrétně o `CPPFLAGS` a `LDFLAGS`. Po nahlédnutí do souboru by mělo by být jasné, co která proměnná obsahuje.

Demonstrační aplikaci po přeložení uživatel spustí ze souboru shadow (nebo stačí použít příkaz `make run`[2]). Velikost hloubkové mapy je dána jediným argumentem (číslo mocniny dvou). Implicitně má rozsah 1024^2. Při spuštění je prezentována scéna z objs/sibenik/sibenik.obj, druhé scény lze použít přepsáním 3. řádku metody `Renderer::loop` v renderer.cc[3]. V pravém horním rohu je k dispozici graf, kde lze sledovat změny hladiny FPS, v levém dolním rohu možno vidět scénu z pozice zdroje světla. Standardně se uživatel nestará o pohyb kamery pozorovatele, ani zdroje světla, ovšem pokud má o to zájem, lze využít klávesy **F1**, resp. **F2**. Pohyb je potom sprostředkován klasicky – přes **WSAD**.

Důležité – techniky se přepínají manuálně:

Technika	Klávesa	
PCF	**P**	(implicitní při startu)
VSM	**V**	
CSM	**C**	
XCSM	**X**	
ESM	**E**	

Dodatečnou konvoluci u metod VSM, CSM, XCSM a ESM možno přepínat klávesou **G**. Šířka PCF kernelu je specifikována konstantní proměnnou `width` v souboru shaders/object-pcf.frag ve funkci `pcf`. Při použití jiných objektů by se měl uživatel ujistit, že odpovídající textury se nachází v adresáři images/. Restart průchodu scénou se učiní pomocí **R**. Zastavit lze **H**. Pokud si chcete vytvořit vlastní dráhu, tak pomocí klávesy **K** možno přidat na konec souboru v keyframes/ informace o nutných vektorech, které má momentálně ovládaná kamera.

[1] MinGW-w64: http://sourceforge.net/projects/mingw-w64/

[2] Upozornění: v distribuci MinGW-w64 se program Make spouští příkazem `mingw32-make`.

[3] objs/ je automatický prefix.