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 STÍNŮ SKRZE PRŮSVITNÉ MATERIÁLY

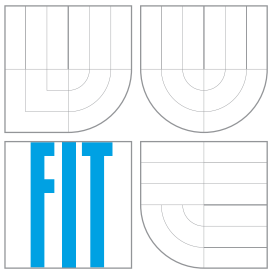
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ROMAN MÁTL

BRNO 2013

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 STÍNŮ SKRZE PRŮSVITNÉ MATERIÁLY

SHADOW RENDERING THROUGH TRANSLUCENT MATERIALS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ROMAN MÁTL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. NAVRÁTIL JAN

BRNO 2013

Abstrakt

Tato práce se zabývá výpočtem stínů skrze průsvitné materiály pomocí metody zvané Barevné stochastické stínové mapy a její praktickou implementací v knihovnách OpenGL. Na začátku této práce je představena metoda Stínových map. V další části jsou zkráceně popsány vybrané způsoby řešení zobrazování průhledných objektů ve scéně. Následuje detailnější vysvětlení metody Barevných stochastických stínových map. V poslední části je popsána implementace demonstrační aplikace v jazyce C++, zhodnocení dosažených výsledků a návrh možností pokračování projektu.

Abstract

This paper describes the calculation of shadows through translucent materials using a method called Colored Stochastic Shadow Maps and its practical implementation in OpenGL libraries. At the beginning of this work a Shadow mapping method is introduced. The next section shortly describes selected ways of dealing with rendering of transparent objects in the scene. This is followed by detailed explanation of the method Colored Stochastic Shadow Maps. The last section describes the implementation of demonstration application in C++ language, evalution of the results and proposal of options to continue the project.

Klíčová slova

CSSM, stínové mapy, stínování, OpenGL

Keywords

CSSM, shadow maps, shading, OpenGL

Citace

Roman Mátl: Zobrazení stínů skrze průsvitné materiály, bakalářská práce, Brno, FIT VUT v Brně, 2013

Zobrazení stínů skrze průsvitné materiály

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Navrátila. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Roman Mátl
12. května 2013

Poděkování

Touto cestou bych velice rád poděkoval vedoucímu mé bakalářské práce, panu Ing. Janu Navrátiovi, za hodnotné a pravidelné konzultace, cenné rady a bezproblémovou komunikaci.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Neodmyslitelnou součástí počítačové grafiky jsou stíny, které značně zvyšují úroveň realističnosti scény a vnáší informaci o hloubce a vzájemné poloze těles. K tomuto patří i autentické vykreslování barvy stínů, jejichž výsledná barva může být ovlivněna nejen barvou objektu, který jej vrhá, ale i jeho fyzikálními vlastnostmi. Zobrazení takovýchto stínů v reálném čase bývá často relativně hodně výpočetně i prostorově náročné. Jedno z řešení tohoto problému popisuje tato bakalářská práce a to metodu Colored stochastic shadow maps, která poskytuje velice zajímavé výsledky, ale také se potýká s různými nedostatky.

Tématem mé bakalářské práce je Zobrazení stínů skrze průsvitné materiály. Toto téma jsem si zvolil, protože mě velice zaujalo. Jedná se o velice diskutované téma a zároveň si myslím, že je také hodně aktuální. Dalším důvodem, který mě přiměl zaobírat se právě touto problematikou je ten, že bych se jednou chtěl uplatnit v oboru počítačové grafiky. Proto jsem rád, že jsem dostal možnost vybrat si právě toto téma. Má bakalářská práce vychází z odborných a známých publikací. Doufám, že tato práce přinese mnoho nových a zajímavých poznatků těm, kteří mají o problematiku stínování zájem.

Nejdříve představím metodu Stínových map, její přednosti i nedostatky, se kterými se potýká a nastíním některá jejich řešení. V další kapitole zkráceně popíši tři vybrané způsoby řešení zobrazování průhledných objektů ve scéně. Poté detailněji vysvětlím metodu Colored stochastic shadow maps, která je jádrem mě bakalářské práce.

V druhé polovině textu popíši implementaci jednoduché aplikace demonstrující metodu Colored stochastic shadow maps v praxi. Nakonec zhodnotím dosažené výsledky a navrhnu další možná rozšíření práce.

Kapitola 2

Stínové mapy

Metoda stínových map podle [15] je aktuálním způsobem k vykreslování dynamických stínů a je stále velmi řešeným tématem. Zejména kvůli její jednoduchosti a efektivnosti, ačkoliv se potýká s mnoha nedokonalostmi (artefakty, aliasy), které je nutné odstranit za pomocí různých vylepšení. V této kapitole popíši základy tohoto stínovacího algoritmu a jeho vybrané modifikace sloužící k dosažení kvalitnějších výsledků.

2.1 Základní algoritmus stínových map

Základní algoritmus stínových map se sestává ze dvou průchodů.

1. Při prvním průchodu je scéna zobrazována z pohledu zdroje světla. Přičemž se vzdálenosti fragmentů vrhajících stín od zdroje světla ukládají do textury (stínové mapy).

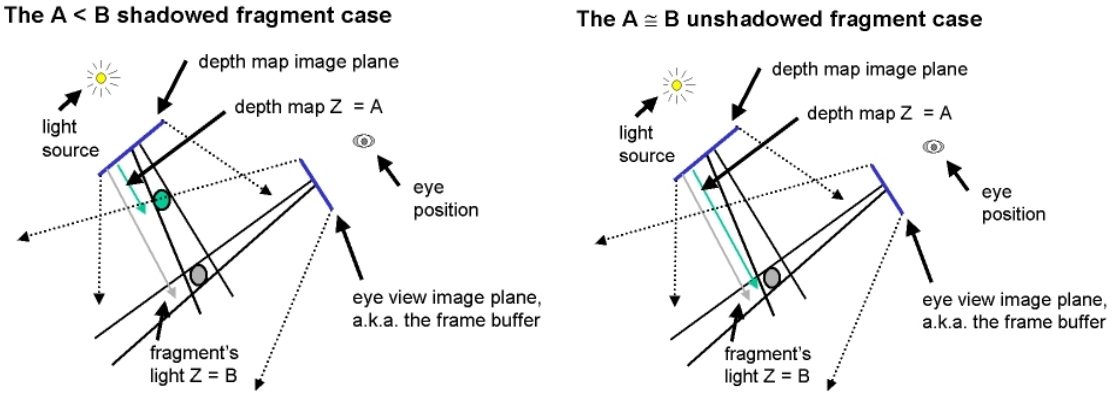
2. V druhém průchodu se scéna zobrazuje z pozice kamery a probíhá test, zdali je daný fragment ve stínu či nikoliv. Pozice zpracovávaného fragmentu je transformována pomocí modelové, pohledové a projekční matice do souřadnicového systému světla. Takto transformované souřadnice použijeme pro čtení hodnoty hloubky ze stínové mapy, kterou jsme získali v prvním průchodu. Tuto hodnotu dále porovnáme se složkou Z (hloubkou) aktuálního fragmentu. Pokud je hodnota získaná ze stínové mapy menší než Z-složka aktuálního fragmentu, tak je zpracovávaný fragment ve stínu. To, že je fragment ve stínu pro nás znamená, že budeme muset upravit jeho výslednou barvu. V opačném případě bude fragment osvětlen. Celé toto porovnání může být provedeno s hardwarovou akcelerací.

2.2 Problémy stínových map

Jak již bylo řečeno, při použití metody stínových map se výsledné stíny potýkají s různými artefakty a aliasy (t.j. různé nežádoucí vzhledové nepřesnosti). Některé tyto jevy se dají zcela odstranit a některé můžeme pouze zmírnit. Zde je stále prostor pro zdokonalení této metody.

2.2.1 Stínové akné

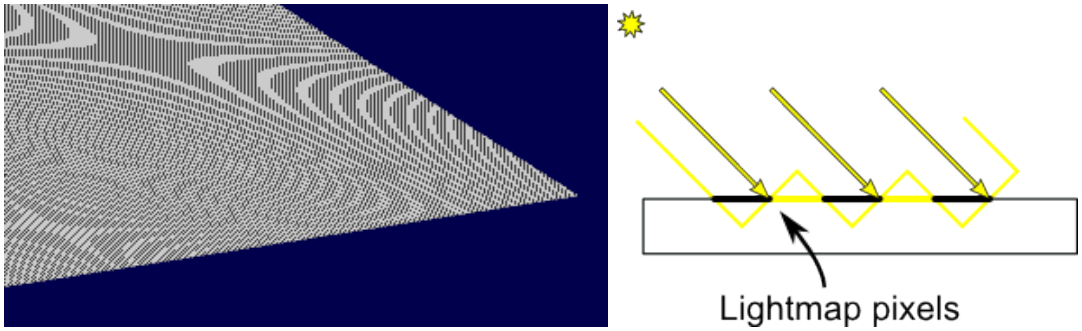
Stínové akné (shadow acne)[2][1] je nejzřetelnějším problémem stínových map, který může vznikat z více důvodů. Prvním důvodem je kvantizace hloubky na jeden pixel stínové mapy.



Obrázek 2.1: Schéma znázorňující rozhodování, zda je fragment ve stínu [10].

Jelikož máme omezenou velikost stínové mapy, tak do ní nemůžeme zapsat všechny hodnoty hloubky ze z-bufferu. Při porovnávání této kvantizované hodnoty s aktuální hloubkou fragmentu, který je na pomezí toho, zda bude ve stínu či ne, je větší pravděpodobnost, že tento fragment bude označen jako zastíněný, i když by neměl být. Jednoduchý nákres 2.2 nabízí názorné vysvětlení tohoto jevu. Na obrázku 2.2 můžeme vidět, jak se tento jev projevuje při zobrazení scény.

Dalším důvodem ke vzniku stínového akné může být to, že hodnota hloubky zpracovávaného fragmentu se blíží k její odpovídající porovnávané hodnotě ve stínové mapě. Zde může dojít při transformaci ze souřadnicového systému pozice kamery do souřadnicového systému zdroje světla k zaokrouhlovací/výpočetní chybě. K této chybě může dojít kvůli tomu, že hodnoty ve stínové mapě jsou počítány vestavěnou funkcí rasterizačního hardwaru, zatímco hodnota hloubky fragmentu z pohledu kamery je vypočítána a porovnána v shaderu[1].



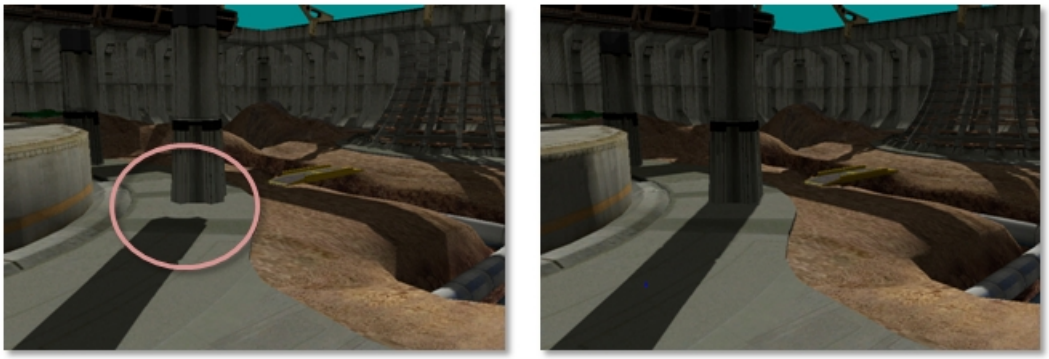
Obrázek 2.2: Schéma příčiny vzniku stínového akné [2].

Stínové akné můžeme zredukovat, či úplně odstranit. K tomuto účelu lze použít například takzvané chybové rozpětí (bias). Fragment bude ve stínu, pouze když bude hodnota hloubky daného fragmentu větší minimálně o určitou hodnotu. Tu udává právě bias. Velikost biasu je velice důležitá a nelze jednoznačně určit, jaká hodnota je nejvhodnější. Nemusíme použít pouze konstantní velikost biasu [2]. Toto řešení však může prohloubit další problém a to Peter Panning, který je popsán dále.

2.2.2 Peter Panning

Název Peter Panning pochází z dětské knížky od autora J. M. Barrie. Tímto názvem označujeme artefakt, který se projevuje tak, že je výsledný stín oddělen od objektu. Objekt potom vypadá, jakoby se vznášel, i když tomu tak není. Tento jev můžete vidět na obrázku 2.3. Peter Panning může být způsoben nevhodnou velikostí biasu nebo malým rozlišením stínové mapy podobně jako u stínového akné.

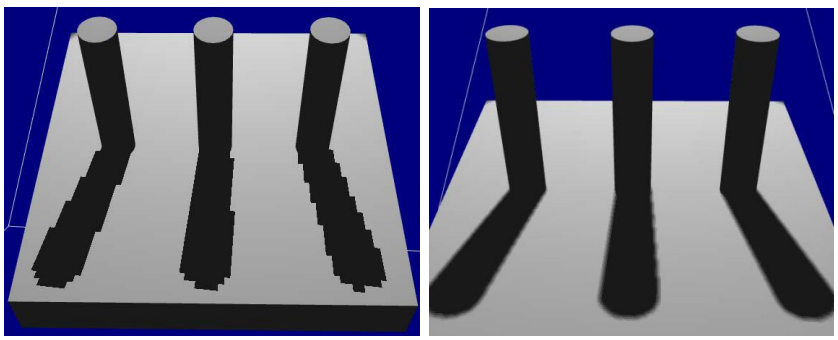
Peter Panning můžeme opět řešit více způsoby. Například vyhýbáním se tenké geometrie modelu. Tedy, aby měli objekty vrhající stín větší hloubku než bias, který jsme nastavili. Tento přístup zcela vyřeší Peter Panning, ale naopak musíme zobrazovat mnohem více dat pro jeden objekt.



Obrázek 2.3: Na levém obrázku zobrazen jev Peter Panning a napravo je scéna zobrazena správně [1].

2.2.3 Aliasing na okrajích stínů

Při výpočtu stínu se určuje, jestli je fragment zastíněn nebo ne. Kvůli tomuto ostrému porovnání vznikají na okrajích stínů ostré přechody mezi zastíněnou a nezastíněnou oblastí. Tento jev je zobrazen na obrázku 2.4. Zde se potom projeví i důsledky kvantizace hloubky fragmentů při tvorbě stínové mapy a to z důvodu omezeného rozlišení stínové mapy. Můžeme zvýšit rozlišení stínové mapy nebo lze takovéto přechody dle libosti zjemnit, nějakou vhodnou vzorkovací metodou, jako například Percentage Closer Filtering (PCF), která je popsána v podkapitole 4.4.1.

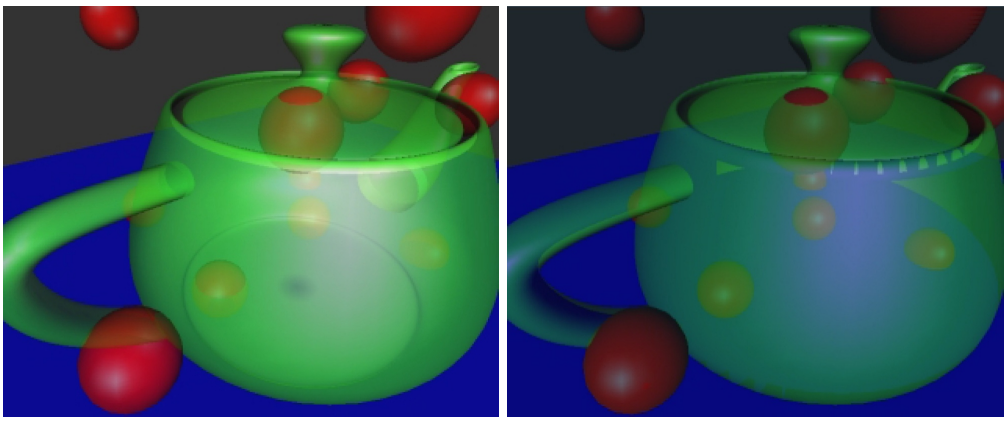


Obrázek 2.4: Nalevo aliasing na okrajích stínů, na pravém obrázku je aliasing na okrajích stínů po aplikaci PCF filtru 4x4 vzorky [9].

Kapitola 3

Algoritmy pro zobrazování průhledných materiálů

Pokud chceme korektně zobrazovat průhledné materiály, potřebujeme mít seřazeny vykreslované polygony podle hloubky, které by se neměli vzájemně protínat. Avšak splnit tento požadavek není vůbec snadné hned z několika důvodů. Například můžeme mít velice složité objekty scény a navíc jich tam může být umístěných mnoho, a to vzájemných průnicích. Polygony v takovéto scéně určitě nebudou splňovat předchozí požadavky. Proto si musíme polygony sami seřadit, ale obyčejné seřazení takového množství polygonů by bylo až příliš výpočetně náročné. Proto je potřeba tento přístup zefektivnit a zdokonalit. Následně stručně popíši některé z metod, s kterými lze řešit zobrazování průhlednosti.



Obrázek 3.1: Na obrázcích lze vidět korektní (vlevo) a nekorektní (vpravo) zobrazení transparentních objektů. Obrázky pochází z článku [7].

3.1 Malířův algoritmus

Malířův algoritmus[1] je založen na vykreslování všech ploch postupně od zadu dopředu. Přes vzdálenější objekty se kreslí bližší objekty. Podobně, jako při práci malíře, který také nejdřív nakreslí pozadí obrazu a postupně přikresluje další vrstvy. Tím dodá svému obrazu na prostorovosti.

[1] http://cs.wikibooks.org/wiki/Geometrie/Viditelnost.

Předpokladem pro tento algoritmus je, že povrch objektů je reprezentován rovnými plochami. Nejprve je potřeba pro každý polygon nalézt jeho nejmenší z-souřadnici a podle této hodnoty seřadit do seznamu všechny polygony ve scéně. První polygon v tomto seznamu musíme otestovat, zdali nepřekrývá některý z ostatních polygonů. Pokud polygon opravdu leží za všemi ostatnímy, tak je vykreslen a vyřazen ze seznamu. V opačném případě se na první místo v seznamu přesune plocha, u které dopadly všechny testy pozitivně.

Nevýhodou malířova algoritmu je, že když se polygony protínají, může dojít ke špatnému zobrazení. Dále může dojít k zacyklení algoritmu a to tehdy, pokud se polygony navzájem překrývají. Tento problém může být řešen rozdělováním ploch na menší části, v důsledku čehož klesá efektivita algoritmu.

3.2 Depth Peeling

Depth peeling je order-independent (průhledné objekty nemusí být seřazeny, před tím než jsou vykresleny[2]) metoda pro zobrazování průhledných objektů. Poprvé byla představena v [7]. Standardním hloubkovým testem dostaneme pouze hloubku nejbližších fragmentů. Jenže my potřebujeme hloubky fragmentů, které jsou druhé nejbližší, třetí nejbližší atd. Depth peeling řeší právě tento problém. Podstatou této techniky je, že při n průchodech scénou dostaneme n hloubkových vrstev scény. Například ve dvou průchodech dostaneme nejbližší a druhou nejbližší vrstvu scény. Pro obě vrstvy budeme mít hodnotu hloubky a barvy včetně alfa kanálu (RGBA) všech fragmentů. Každou vrstvu si uložíme zvlášť do textur. Znázornění jednotlivých vrstev můžete vidět na obrázku 3.2.

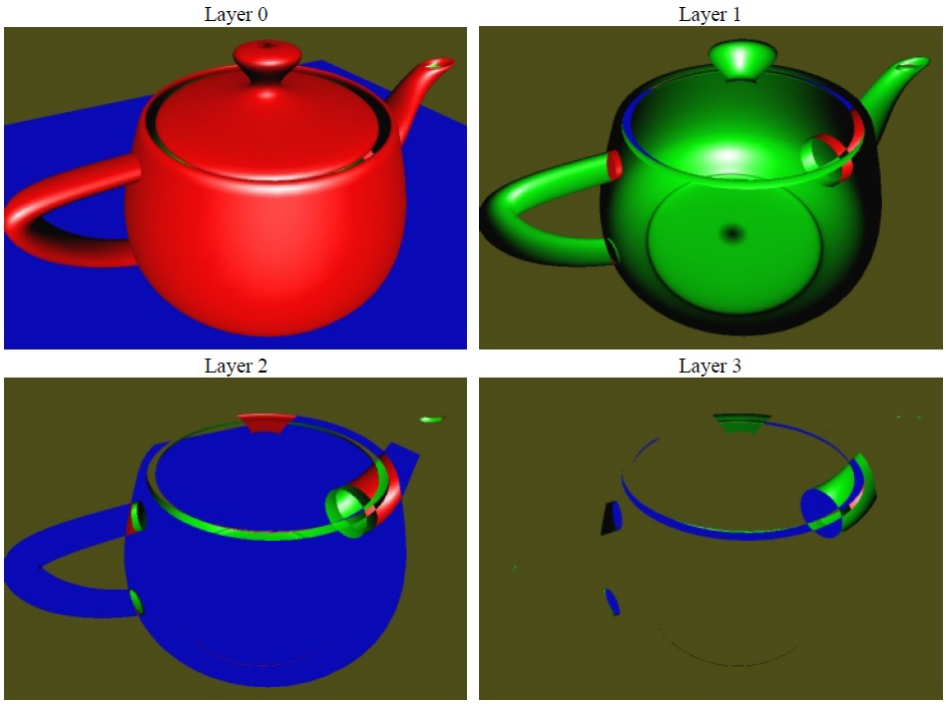
Nakonec jen zobrazíme jednotlivé textury obsahující vrstvy od nejvzdálenější po nejbližší tak, že je namapujeme na čtverec velikosti výřezu scény (viewport) a použijeme přitom funkci pro míchání barev (blendig function). Tím dosáhneme efektu průhlednosti.

Rychlost depth peelingu je omezena počtem průhledných vrstev objektů. Další nevýhodou je, že předem nevíme, jaký počet průchodů budeme potřebovat. Existuje efektivnější varianta tohoto algoritmu zvaná Dual Depth Peeling [3].

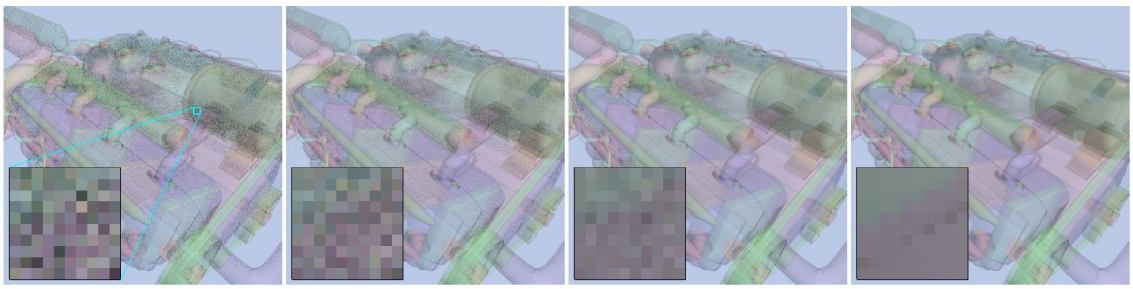
3.3 Stochastic Transparency

Stochastic transparency je order independent metoda, která je představena v článku [6]. Metoda vychází ze Screen-door transparency[3] a Deep shadow maps (hloubkové stínové mapy) [11]. Algoritmus je založen na stochastickém vzorkování překrývajících se fragmentů. Konkrétně při zvýšeném rozlišení na základě alfa složky barvy jednotlivých povrchů. Díky převzorkování (downsampling) dostaneme korektně zobrazené barvy průhledných objektů v jediném průchodu geometrií a to bez setřiďování fragmentů a velikost potřebné paměti nenarůstá. Ale výsledný obraz je zatížen stochastickým šumem. Na obrázku 3.3 můžete vidět výsledky použití metody při různém počtu vzorků. K redukci tohoto nežádoucího jevu se používá několik technik. Nejdůležitější technikou k redukci šumu je alpha correction pass. Metoda znatelně zjemní šum, ale potřebujeme kvůli ní další průchod geometrií.

[2] http://www.codermind.com/answers/What-is-order-independent-transparency.html.
[3] http://www.bluevoid.com/opengl/sig00/advanced00/notes/node211.html.



Obrázek 3.2: Znázornění jednotlivých vrstev (layer) získaných metodou depth peeling. Layer 0 zobrazuje nejbližší vrstvu, Layer 1 druhou nejbližší vrstvu, atd. Pro názornost jsou použity pestré barvy a 2 zdroje světla. Jeden uvnitř konvičky a jeden ze strany [7].

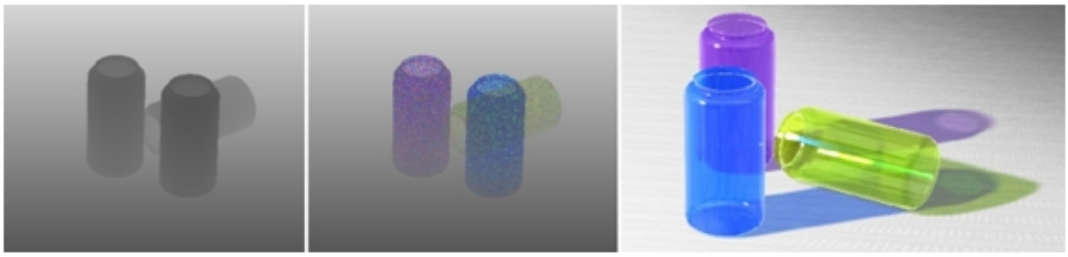


Obrázek 3.3: Zobrazení scény za použití základního algoritmu stochastic transparency při 8, 16 a 64 vzorcích na jeden pixel. Na posledním obrázku je scéna zobrazena metodou depth peeling [6].

Kapitola 4

Colored Stochastic Shadow Maps

Colored Stochastic Shadow Maps (dále CSSM) je datová struktura, ale i metoda představena v publikaci [12]. Její název je odvozen jak z toho, že produkuje barevné stíny, tak pro svůj vzhled po zobrazení. Metoda vychází z tradičního algoritmu stínových map podle publikace [15]. Díky tomu je CSSM algoritmus poměrně rychlý na výpočet a lze na něj aplikovat většinu rozšíření a antialiasingových úprav aplikovatelných na stínové mapy. Proto je vhodná pro real-timeové aplikace. Nevýhodou je, že se metoda potýká se stejnými druhy artefaktů, jako stínové mapy. CSSM je také rozšířením algoritmu Stochastic transparency [6].



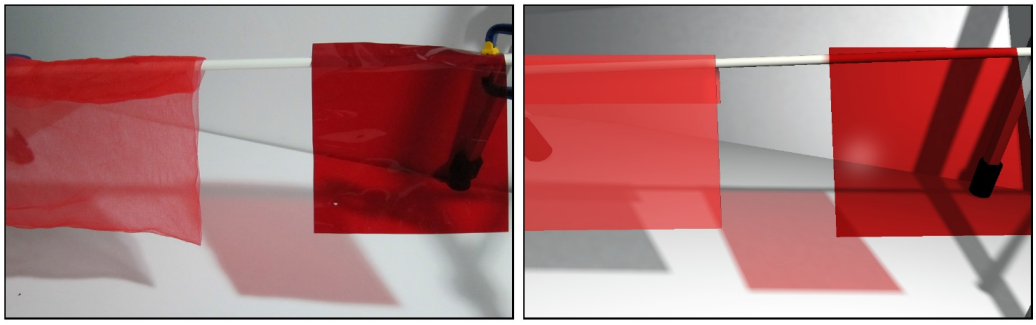
Obrázek 4.1: Klasická stínová mapa, Colored Stochastic Shadow Map a zobrazovaná scéna pomocí metody CSSM [12].

4.1 Kombinace překrytí a přenosu

Pro výpočet barevných stínů se pánové McGuire a Enderton snaží nalézt přístup ke vztahu mezi pokrytím a přenosem. Jedná se o tři statistické akce mezi fotonem o vlnové délce lambda a povrchem, který leží uvnitř trojúhelníku:

A = „daný foton zasáhne trojúhelník v okolí povrchu“
S = „daný foton zasáhne samotný povrch“
T = „daný foton je přenášen skrz povrch“

Rozdílu mezi povrchem objektu a trojúhelníkem může být například objekt list stromu, který je modelován jako trojúhelník větší než list a vnější oblast vlastního listu je odříznuta oblastí, kde je hodnota $\alpha = 0$. Takto označená oblast bude považována jako nepřítomná (zcela průsvitná). Můžeme si představit, že průsvitnost skla je dána určitým množstvím



Obrázek 4.2: Nalevo fotografie demonstrující jev různých stínů červené látky a červeného divadelního gelu. Vzhled stínu látky je způsoben částečným překrytím (α) neprůhledných červených vláken. U gelu je vzhled stínu způsoben prosvícením červené složky světla. Tento jev může být simulován pomocí různých vektorů $\vec{\rho_\lambda}$. Napravo je scéna zobrazena pomocí algoritmu Colored Stochastic Shadow Maps [12].

otvorů rozmístěných po celém jeho povrchu, skrz které mohou pronikat jednotlivé fotony. Statistická reprezentace tohoto jevu pro jednotlivé povrchy je částečné překrytí.

Nechť je pravděpodobnost, že foton dané vlnové délky zasáhne povrch, dána pravděpodobností, že foton zasáhne trojúhelník, ve kterém se tento povrch nachází, vyjma oblasti za okrajem povrchu je dána vztahem:

$$P(S|A) \quad = \quad \alpha \tag{4.1}$$

Nechť je pravděpodobnost, že je foton o vlnové délce λ prosvícen (přenesen) skrz povrch dána vztahem:

$$P(T|S) \quad = \quad \vec{t_\lambda} \tag{4.2}$$

Několik příkladů, jak bychom mohli modelovat jednotlivé povrchy, můžete vidět v tabulce 4.1. Jak vektor $\vec{t_\lambda}$, tak i hodnota α mohou být namapovány na texturu materiálu.

Materiál	α	$\vec{t_r}$	$\vec{t_g}$	$\vec{t_b}$
Zelené sklo	1,00	0,1	0,9	0,1
Průsvitný nylon	0,25	0,5	0,5	0,5
Cihla	1,00	0,0	0,0	0,0
Černý nylon	0,25	0,0	0,0	0,0

Tabulka 4.1: Příklady vlastností materiálů pro výpočet $\vec{\rho_\lambda}$. Pravděpodobnosti fotonů zasáhnout povrch materiálu (α) a pravděpodobnosti jednotlivých vlnových délek světla na prosvícení materiálu $\vec{t_\lambda}$ podle publikace [12].

Dále pravděpodobnost fotonu, který zasáhl trojúhelník, že bude pohlcen nebo odrazen povrchem můžeme zredukovat na výraz:

$$\begin{aligned} \vec{\rho_\lambda} &= (\overline{T}|A) \\ \vec{\rho_\lambda} &= 1 - P([(S \cap T) \cup \overline{S}]|A) \\ \vec{\rho_\lambda} &= (1 - \vec{t_\lambda})\alpha \end{aligned} \tag{4.3}$$

Jinými slovy, vektor $\vec{\rho_\lambda}$ je zlomek světla pro každou vlnovou délku λ, který zasáhne povrch materiálu a není prosvícen skrz. Tuto materiálovou konstantu využijeme v následujícím algoritmu.

4.2 Obecný algoritmus CSSM1

Obecný algoritmus Colored Stochastic Shadow Maps zvaný CSSM1 je rozšířením algoritmu Stochastic transparency [6]. Tento přístup vyžaduje pole složené z více stínových map. Pro každou vlnovou délku budeme potřebovat jednu stínovou mapu, například tři pro barevný model RGB.

CSSM1 ALGORITHM

generateShadowMap():

1. For each wavelength λ:
   (a) Bind and clear depth texture shadow[λ]:
   (b) Set the projection matrix from the light's viewpoint
   (c) Render all surfaces; discard fragments with $\vec{\xi_\lambda} < \vec{\rho_\lambda}$
2. Return the shadow array

shadowedLightColor():

1. Let $\vec{s}_{xyz}$ be the projected shadow map texture coordinate and depth value (as specified by GLSL `shadow2D`)
2. For each wavelength λ:
   (a) Let $X_\lambda = 0$
   (b) For each sample offset Δ (of n total):
      i. $\vec{X_\lambda}$+=(`texture2D`(shadow[λ], $\vec{s}_{xy} + \vec{\Delta}$).r $> \vec{s_z}$)
   (c) $\vec{X_\lambda} = \vec{L_\lambda}\vec{X_\lambda}/n$
3. Return $\vec{X}$

Obrázek 4.3: Pseudokód algoritmu CSSM1 podle publikace [12].

Algoritmus se skládá ze dvou částí. V první části probíhá vytvoření CSSM na základě vektoru $\vec{\rho_\lambda}$. Ve druhé části provedeme hloubkový test podobně jako u stínových map, rozšířený pro potřeby algoritmu CSSM. Pseudokód tohoto algoritmu můžete vidět na obrázku 4.3.

V prvním kroku budeme zobrazovat scénu z pohledu světla. Porovnáváme náhodné číslo v rovnoměrném rozložení pravděpodobnosti ξ s konstantní hodnotou ρ_λ, kterou jsme získaly výpočtem v předchozí kapitole. Pokud je hodnota ξ menší než hodnota ρ_λ, tak do textury reprezentující stínovou mapu pro danou vlnovou délku zapisujeme hloubku fragmentu. V opačném případě do textury zapisujeme hodnotu reprezentující danou vlnovou délku (v našem případě hodnotu vyhodnocovaného barevného kanálu v intervalu [0,1]). Tento krok musíme zopakovat pro každou vlnovou délku λ.

V dalším kroku zobrazujeme scénu z pohledu kamery. Souřadnice zpracovávaného fragmentu transformujeme do souřadnicového systému zdroje světla. S takto transformovanými souřadnicemi přistupujeme do jednotlivých stínových map, které jsme vytvořili v předešlém kroku. Získanou hodnotu porovnáme s hloubkou fragmentu a na základě výsledku porovnání upravíme barevný kanál fragmentu. Tento postup aplikujeme na stínové mapy pro všechny vlnové délky světla. Porovnání bychom měli provádět v rámci nějaké vzorkovací metody (rekonstrukční filtr). Tuto problematiku dále rozepíšu v kapitole 4.4.

Druhý krok musí být aplikován v kontextu libovolného algoritmu pro zobrazování průsvitných materiálů viz. kapitola 3.1.

Existuje několik nevýhod algoritmu CSSM1. První nevýhodou je, že musíme vytvářet více stínových map pro jeden zdroj světla. To nám markantně zvýší čas potřebný k vytvoření stínové mapy, kvůli několika násobnému průchodu. Také zabereme více paměti a musíme přenášet více dat potřebných k vykreslení do grafické karty.

Další nevýhodou je, že metoda vyžaduje větší rozlišení stínové mapy k tomu, aby byl výsledný stín přijatelně jemný (bez zřetelnějšího stochastického šumu, zrnění). Tento jev je důsledkem stochastického vzorkování při tvorbě stínové mapy. S tímto problémem se potýká většina stochastických metod. Dále se můžeme potýkat nejen s problémem šumu intenzity stínu, ale i s odstínovými odlišnostmi barvy stínu, které jsou opět způsobeny stochastickým vzorkováním jednotlivých vlnových délek světla.

4.3 Efektivní algoritmus pro barevný model RGB (CSSM2)

Algoritmus CSSM2 je časovou a prostorovou optimalizací algoritmu CSSM1 pro případ běžného RGB barevného modelu. Abychom se vyhnuli vytváření tří stínových map ve třech průchodech, tak použijeme jednu barevnou texturu. Do barevné textury lze zapsat hodnoty pro jednotlivé barevné kanály RGB, což využijeme jako naše tři stínové mapy. Tento přístup nám okamžitě přináší trojnásobné zvýšení výkonu při generování stínové mapy. Dále můžeme zvektorizovat a tudíž zefektivnit nejen výpočet hodnot, které budeme do textury zapisovat, ale také výpočet výsledné barvy fragmentu při stínování.

Je potřeba zakomponovat hodnoty hloubky fragmentů do naší barevné stínové mapy, abychom neztratili její funkci pro určování zastíněných fragmentů. K tomuto účelu budeme potřebovat dočasnou tradiční stínovou mapu pro neprůhledné materiály. Tuto dočasnou stínovou mapu použijeme při vytvoření CSSM textury místo hodnot hloubky podobně jako u algoritmu CSSM1. K nasimulování hloubkového testu pro průhledné materiály použijeme minimální míchání barevných kanálů (`min-blending`)[1]. Na obrázku 4.4 je zobrazen pseudokód tohoto algoritmu.

4.4 Rekonstrukční filtr

Při vytváření stínu metodou CSSM narážíme na problém. Tímto problémem jsou odstínové odlišnosti barvy výsledného stínu, které jsou způsobeny stochastickým vzorkováním hodnot jednotlivých barevných kanálů při tvorbě CSSM textury. K dosažení slušných výsledků je potřeba tento jev co nejvíce zredukovat. Za tímto účelem potřebujeme kvalitní rekonstrukční filtr, který může využívat různé vzorkovací metody při vytváření stínů. Při tvorbě rekonstrukčního filtru musíme docílit toho, abychom při co nejmenším počtu vzorků dosáhly

[1] `http://www.opengl.org/sdk/docs/man/xhtml/glBlendEquation.xml`.

CSSM2 ALGORITHM

generateShadowMap():

1. Set the projection matrix from the light's viewpoint
2. Bind and clear the depth buffer and shadow color buffer
3. Disable color write, enable depth write
4. Render all opaque surfaces
5. Enable color write, disable depth write
6. Copy depth to all color channels by rendering large quad
7. Set MIN blending
   (i.e., `glBlendEq(BLEND MIN)`; `glBlendFunc(ONE, ONE)`)
8. Render all translucent surfaces; let each fragment's color be $max(z, (\vec{\xi} > \vec{\rho}))$, where z is the fragment's depth value
   (i.e., `glFragCoord.z`)
9. Return the shadow color buffer texture

shadowedLightColor():

1. Let $\vec{s}_{xyz}$ be the projected shadow map texture coordinate and depth value
2. Let $\vec{X} = \vec{0}$
3. For each sample offset $\vec{\Delta}$ (of n total):
   (a) $\vec{X}$+=(`texture2D`(shadow, $\vec{s}_{xy} + \vec{\Delta}$).rgb $> \vec{s_z}$)
4. Return $\vec{X}\vec{L}/n$

Obrázek 4.4: Pseudokód algoritmu CSSM2 podle publikace [12].

co nejlepších výsledků. Obvykle se využívají různé poznatky a teoretické úvahy o lidském vnímání a charakteristik specifických šumových funkcí.

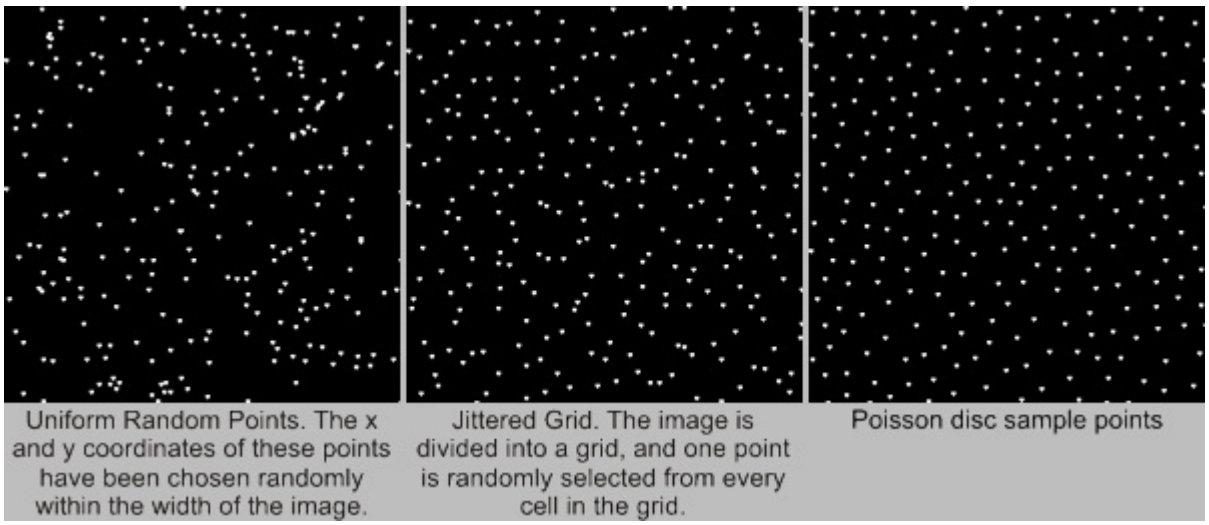
4.4.1 Percentage Closer Filtering (PCF)

Metoda Percentage Closer Filtering [9] pomáhá zmírnit aliasing na okrajích stínů při použití stínovací metody stínových map. Je to vzorkovací metoda. Nejdřív se vyhodnotí test zastínění okolních fragmentů, což jsou vybrané vzorky. Šířku filtru a počet vzorků lze libovolně měnit, k docílení ideálního poměru vzhledu stínů a dobou výpočtu stínů. Také lze měnit způsob volby pozice vzorků. Na základě hodnot těchto vzorků vypočítáme aritmetickým průměrem výslednou hodnotu míry zastínění zpracovávaného fragmentu.

4.4.2 Poisson disk vzorkování

Tento způsob tvorby vzorků pro rekonstrukční filtr má skvělou vlastnost. A to že výsledné body jsou blízko u sebe, ale nejsou blíže k sobě, než udává minimální vzdálenost vzorků. Minimální vzdálenost vzorků nemusí být konstantní. Na obrázku 4.5 lze pozorovat rozdíl ve vytváření bodů za použití různých funkcí. Konkrétní způsob vytváření jednotlivých bodů

je popsán v publikaci [4].



Obrázek 4.5: Ukázka bodů generovaných různými způsoby. Na levém obrázku je použito rovnoměrné rozložení pravděpodobnosti. Na dalším obrázku je vždy náhodně vybrán jeden bod z každé sekce vzniklé po rozdělení celé plochy mřížkou. Na pravém obrázku je použito Poisson disk vzorkování. Zdroj: [14].

4.4.3 CSSM rekonstrukční filtr

Filtr pro metodu CSSM2 musí být širší než bilineární kvůli lepší redukci stochastického šumu. U rekonstrukčního filtru uvedeného v publikaci [12] můžeme pozorovat relativně dobré výkonnostní a aliasingové vlastnosti. Filtr má tvar čtverce kombinovaného s křížem. Skládá se ze třinácti pozic, umístěných relativně k pozici zpracovávaného texelu, na kterých budeme odebírat vzorky.

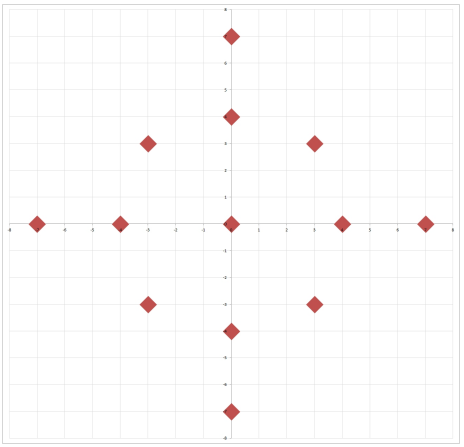
$$\begin{aligned} \vec{\Delta} &= \vec{x} + \vec{\delta}(\vec{s}_{xy}) \\ \vec{x} &\in \{(0,0),(\pm 3,\pm 3),(\pm 4,0),(0,\pm 4),(\pm 7,0),(0,\pm 7)\} \end{aligned} \tag{4.4}$$

Symbolem $\vec{\delta}$ je označen mikro-offset, který je aplikován na pozici každého vzorku ($\vec{x}$), to nám zajišťuje rozptýlení (jittering). Touto technikou dosáhneme lepších aliasingových výsledků a ve výsledném stínu nebude patrný tvar filtru. Pomocí následujícího vzorce určíme velikost mikro-offsetu.

$$\vec{\delta}(\vec{s}_{xy}) = \frac{[(5\vec{s}_{xy})mod(2,2)] - (1,1)}{6 \parallel \frac{\partial \vec{s}_{xy}}{\partial x} \parallel_1, \parallel \frac{\partial \vec{s}_{xy}}{\partial y} \parallel_1} \tag{4.5}$$

Operátor $\parallel \ . \ \parallel_1$ značí Manhattanskou vzdálenost[2]. Jmenovatel rovnice 4.5 vychází z toho, že Manhattanská vzdálenost prostorové derivace je podporována funkcemi OpenGL/DirectX API a GPU hardwarem.

[2] http://mathworld.wolfram.com/TaxicabMetric.html.



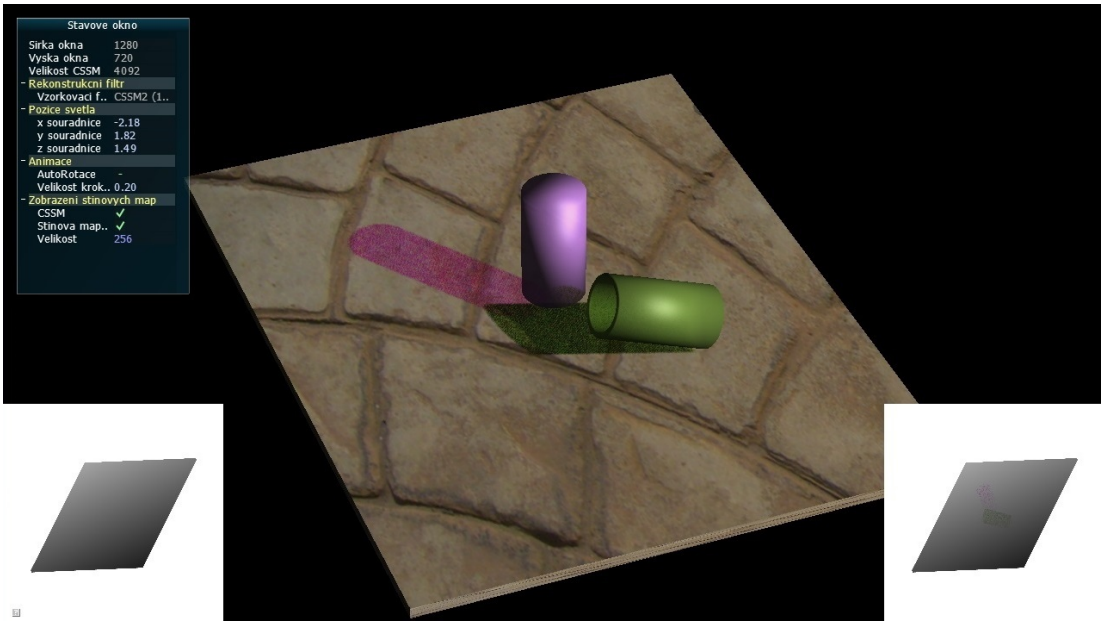
Obrázek 4.6: Schéma filtru CSSM2.

Kapitola 5

Implementace demonstrační aplikace

Vytvořil jsem jednoduchou aplikaci za účelem vyzkoušení a experimentování s algoritmem CSSM v praxi. K implementaci jsem použil jazyk C++ a grafickou knihovnu OpenGL verze 3.3 a výš. Pro psaní shaderů jsem použil jazyk GLSL (OpenGL Shading Language) verze 3.0. K vytvoření okna a zpracování vstupů klávesnice a myši jsem zvolil knihovnu SDL (Simple Directmedia Layer)[1]. Dále jsem použil knihovnu AntTweakBar pro vytvoření grafického uživatelského rozhraní. Modely objektů scény jsem vytvořil v programu Blender.

Výsledná aplikace po spuštění obsahuje osvětlenou scénu. Uživatel má možnost se scénou provádět různé transformace. Jedná se o rotaci, zvětšení a posun. Také je možné měnit pozici zdroje světla, zapnout jednoduchou animaci scény či zvolit vzorkovací metodu rekonstrukčního filtru.



Obrázek 5.1: Screenshot demonstrační aplikace.

[1] http://www.libsdl.org/.

Vytvoření stínů scény probíhá ve dvou průchodech. První průchod slouží k vygenerování stínové mapy a v druhém průchodu se vytváří vlastní stíny. V následujících podkapitolách popíši, co se provádí v jednotlivých krocích.

5.1 Vytvoření CSSM

Zvolil jsem ortogonální zdroj světla simulující sluneční svit. Výběr zdroje světla lze poměrně jednoduše změnit mírnou úpravou zdrojového kódu. Na začátku vykreslovací smyčky probíhá výpočet pozice a směr záření světla. Výsledkem je matice, kterou použiji ve vertex shaderu pro výpočet zobrazení scény z pozice světla. Nejdříve jsem vyrenderoval všechny neprůhledné objekty scény z pohledu světla a do textury typu `GL_RGB` jsem zapsal hodnoty vzdáleností jednotlivých fragmentů od zdroje světla. Čímž jsem vytvořil stínovou mapu neprůhledných objektů. Tuto stínovou mapu jsem zobrazil přes celou scénu a následně jsem nastavil minimální míchání barev (`glBlendEquation(GL_MIN)` a `glBlendFunc(GL_ONE, GL_ONE)`). Dále zobrazím všechny průhledné objekty scény. Pomocí datového typu `uniform` předám do fragment shaderu čtyřrozměrný vektor, který obsahuje materiálové konstanty α, $\vec{t_r}$, $\vec{t_g}$ a $\vec{t_b}$ simulující částečné překrytí a přenos. Bližší informace jsou v podkapitole 4.1.

Ve fragment shaderu provedu namapování textur na modely. Vytvořím vektor $\vec{R}$ složený ze tří různých náhodných čísel, které generuji pro každý fragment. Výběr dobrého a hlavně rychlého generátoru náhodných čísel v intervalu $\langle 0, 1 \rangle$ v rovnoměrném rozložení pravděpodobnosti je velice důležitý, protože by špatná volba mohla významně zpomalit výpočet nebo způsobit zvýšení výskytu artefaktů ve výsledném stínu.

V inicializační části aplikace jsem vytvořil CSSM texturu. CSSM je textura typu `GL_RGB16F`, což je textura, jejíž texely se skládají ze tří šestnáctibitových hodnot typu `float`. Velikost CSSM je 4096x4096. Dále provedu výpočet stochastické průsvitnosti. Na základě vztahu 4.3 vytvořím vektor `rho`.

```
color = max(vec3(greaterThan(R, rho)), vecDepth.x);
```

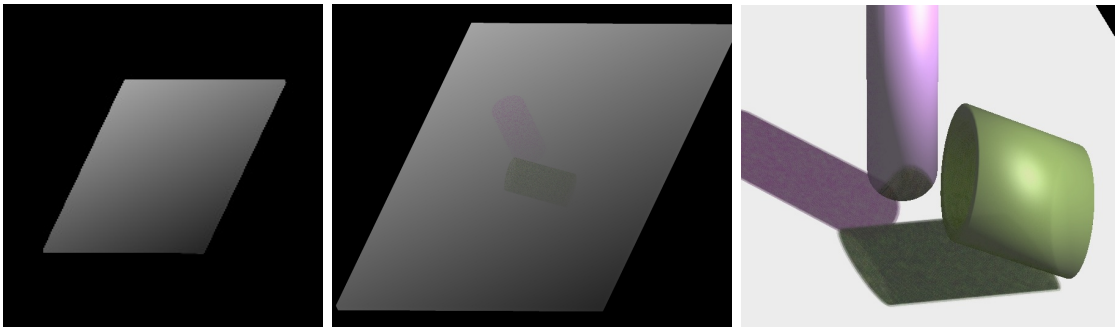
Porovnám tří složkové vektory `rho` a $\vec{R}$. Výsledkem porovnání je tří složkový vektor stejného typu. Porovnávají se jednotlivé složky. Pokud porovnání vyjde kladně, výsledek je roven jedné, v opačném případě je výsledná složka rovna nule[2]. Nakonec do CSSM textury zapíši tří složkový vektor, kde v každé složce je větší ze z-souřadnice a výsledku předešlého porovnání.

5.2 Vytvoření stínu

Ve vertex shaderu provedu přepočítání souřadnic do homogenního systému souřadnic pro správné adresování CSSM, k čemuž použiji MVP matici z pohledu světla. Dále ji vynásobím s bias maticí kvůli převodu do homogenního systému souřadnic. Výslednou matici vynásobím pozicí zpracovávaného fragmentu. Výsledkem je souřadnice, pomocí které získám data k danému fragmentu z CSSM.

Ve fragment shaderu provedu určení difúzní složky a barvy světla. Pomocí difúzní složky vypočítám ambientní složku světla, která slouží k nasimulování nepřímého osvětlení. Poté zvolím odrazovou složku světla potřebnou pro výpočet odražené části světla. Na základě těchto dat implementuji Phongův osvětlovací model [13].

[2] například pro vektor vec3 a = vec3(0.7, 0, 0.6); vec3 b = vec3(0, 0.2, 0.5); vec3(greaterThan(a, b)) = vec3(1.0,0,1.0)



Obrázek 5.2: Stínová mapa pro neprůhledné objekty scény, výsledná CSSM textura, zobrazovaná scéna.

V dalším kroku je potřeba vypočítat vektor osvětlení CSSM pomocí rekonstrukčního filtru (viz. podkapitola 5.3). Nakonec z osvětlení CSSM a z osvětlovacího modelu vypočítám výslednou barvu fragmentu, kterou následně vykreslím.

5.3 Aplikace rekonstrukčních filtrů

V cyklu, který se opakuje podle počtu vzorků, se provádí výběr hodnoty z CSSM textury pro daný fragment. V každém opakování je vyhodnoceno porovnání z-souřadnice zpracovávaného fragmentu s každou z hodnot uložených v CSSM (tří složkový vektor ve kterém je uložena informace pro stínování určena pro RGB barevné kanály). Pokud je tato hodnota větší než z-souřadnice, tak se do daného barevného kanálu výsledného vektoru přičte hodnota $1,0$. Pokud je menší nepřičítá se nic. Po dokončení cyklu se výsledný vektor podělí počtem opakování vzorku. Tímto jsem provedl vzorkování (sampling). K zajištění míry rozptýlení hodnot (rozmazání, vyhlazení výsledného stínu) je souřadnice pro přístup do CSSM modifikována offsetem. Bližší informace lze vidět v podkapitole 4.4. Offset je dvousložkový vektor, který lze určit mnoha způsoby a jeho výběr je velice důležitý pro kvalitu rekonstrukčního filtru. V to mto kroku se liší jednotlivé metody výběru offsetu, které dále popíši.

```
vec3 cssmColor;
for (int i = 0; i < pocet_vzorku; i++){
        vec2 Offset = vypocet_podle_zvolene_vzorkovaci_metody;
        cssmColor += (texture2D(coloredShadowMap,
                (ShadowCoord.xy + Offset)).rgb >
                        (ShadowCoord.z - bias))
}
cssmBarva = cssmBarva / pocet_vzorku;
```

Obrázek 5.3: Pseudokód rekonstrukčního filtru.

5.3.1 Konstantní okolí bodu (PCF)

Tento filtr je jednoduchý. Pro sběr vzorků jsem zvolil 9 bodů s pevnými souřadnicemi. Nevyužil jsem hardwarové optimalizace, protože by měly jen malý vliv na rychlost vykreslování.

Výsledná kvalita zobrazení by byla totožná.

5.3.2 Poisson disc

Do pole dvousložkových vektorů ve fragment shaderu jsem uložil již vygenerovaná čísla v rozložení poisson disc. Generování čísel při každém průchodu by bylo značně výpočetně náročné. Způsob generování těchto bodů je popsán v podkapitole 4.4.2. Z tohoto pole v každém cyklu vzorkování vezmu uložené souřadnice jako offset. Velikost filtru jsem zvolil 16 vzorků.

5.3.3 Rovnoměrné rozložení

Zde jsem zvolil třiceti vzorkový filtr. Jednotlivé body jsou při každém opakování generovány v rovnoměrném rozložení pravděpodobnosti. Použil jsem GLSL funkci `random` a jako seed jsem zvolil pozici fragmentu v rámci scény. Pozice bodů lze i vygenerovat předem ve větším množství a uložené do textury předat do shaderu pro zrychlení výpočtu. V tomto případě jsem použil první variantu.

5.3.4 CSSM rekonstrukční filtr

Do pole jsem uložil třináct bodů tak, aby byly umístěny relativně ke zpracovávanému fragmentu. Pro každý z těchto bodů vypočítám mikro-offset podle rovnice 4.5. Při tom využívám několik GLSL vestavěných funkcí. Pro výpočet Manhattanské vzdálenosti jsem zvolil funkci `manhattanDistance` a pro výpočet parciálních derivací funkce `dFdx` a `dFdy`. Výsledný offset je tedy součtem relativní souřadnice bodu a mikro-offsetu.

5.4 Implementační detaily

Při implementaci algoritmu jsem narazil na několik problémů. Některé z nich následovně popíši.

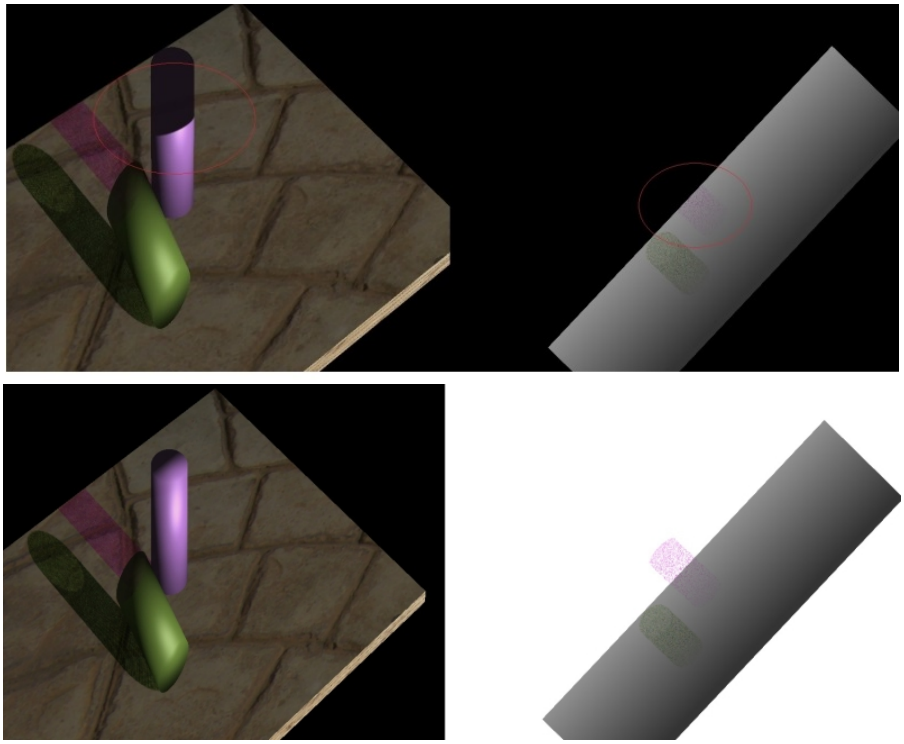
5.4.1 Řešení zobrazování průhledných objektů

Pro zajištění správného zobrazení průhledných objektů pro tvorbu CSSM a zobrazení scény jsem použil princip Malířova algoritmu. Pro každý objekt scény počítám přibližné těžiště, při načítání modelů. Před každým vykreslením scény provedu seřazení objektů podle vzdálenosti od bodu pozorování od nejvzdálenějšího po nejbližší. Při vykreslování se jednotlivé objekty vykreslují podle pořadí. Toto řešení není příliš ideální pro složitější scény, ale pro potřeby této demonstrační aplikace dostačující.

5.4.2 Nevhodné zastínění objektů

Další problém na který jsem narazil, nastává když je objekt umístěn tak, že z pohledu světla v jeho pozadí není žádný neprůhledný objekt. V tomto případě je ta část objektu nevhodně zastíněna. Tento jev je způsoben nastavením minimálního míchání barev (viz. kapitola 5.1) při tvorbě CSSM. Toto míchání způsobí, že výsledná hodnota v CSSM pro část fragmentů je rovna výchozí barvě při čištění framebufferu a to barvě černé. Zobrazení tohoto jevu lze vidět na obrázku 5.4.

K odstranění tohoto jevu jsem před tvorbou CSSM pomocí příkazu `glClearColor` nastavil barvu pro čištění framebufferu na bílou. Po vytvoření CSSM jsem tuto hodnotu opět nastavil na výchozí, aby finální vykreslení scény proběhlo správně. Tímto krokem jsem zajistil to, že při míchání barev nedojde ke ztrátě informace o stínu a objekt již nebude nevhodně zastíněn.



Obrázek 5.4: Na horním obrázku je zobrazen jev, kdy je objekt nevhodně zastíněn. Na spodním obrázků je zobrazena scéna po aplikaci zvolené úpravy.

Kapitola 6

Vyhodnocení výsledků

Demonstrační aplikaci jsem testoval především na počítači s grafickou kartou NVIDIA GeForce GTX 650 pod operačním systémem Windows 7 při použití knihovny OpenGL 3.3. V tabulce 6.1 jsou zobrazeny výsledky mého testování. Pro výpočet jednotlivých časů jsem použil OpenGL časovače, které umožňují přesné a asynchronní vyhodnocení časů potřebných pro zpracování jednotlivých příkazů grafickým procesorem. Při testování jsem se zaměřil hlavně na srovnání algoritmu CSSM implementovaného v mé demonstrační aplikaci s algoritmem stínových map podle publikace [15] pro různě složité scény seřazené podle počtu trojúhelníků, ze kterých jsou složeny. Dále jsem sledoval vztah, mezi dobou výpočtu a velikostí stínové mapy a barevné stochastické stínové mapy.

Stínovací metoda		**SM**	**CSSM**	**SM**	**CSSM**	**SM**	**CSSM**
Rozlišení		**1928x1080**		**4092x4092**		**8192x8192**	
Scéna 12k tr.	tvorba	0,7	1,9	1,0	3,5	5,6	13,1
	vykreslení	0,5	1,6	0,6	3,2	0,8	3,1
	celkem	**1,2**	**3,5**	**1,6**	**6,7**	**6,4**	**16,2**
Scéna 37k tr.	tvorba	0,9	0,9	2,2	7,4	1,1	18,9
	vykreslení	0,9	1,6	2,2	3,9	6,5	9,4
	celkem	**1,8**	**3,5**	**4,4**	**11,3**	**7,6**	**28,3**
Scéna 786k tr.	tvorba	1,0	14,7	10,9	27,4	1,0	74,1
	vykreslení	5,3	25,7	6,7	29,5	7,9	21,7
	celkem	**6,3**	**40,4**	**17,6**	**56,9**	**8,9**	**95,8**

Tabulka 6.1: Tabulka udává dobu výpočtu GPU v milisekundách potřebnou pro vytvoření stínů pomocí algoritmu stínových map (SM) podle publikace [15] a algoritmu CSSM implementovaném v mé demonstrační aplikaci pro různé rozlišení SM a CSSM.

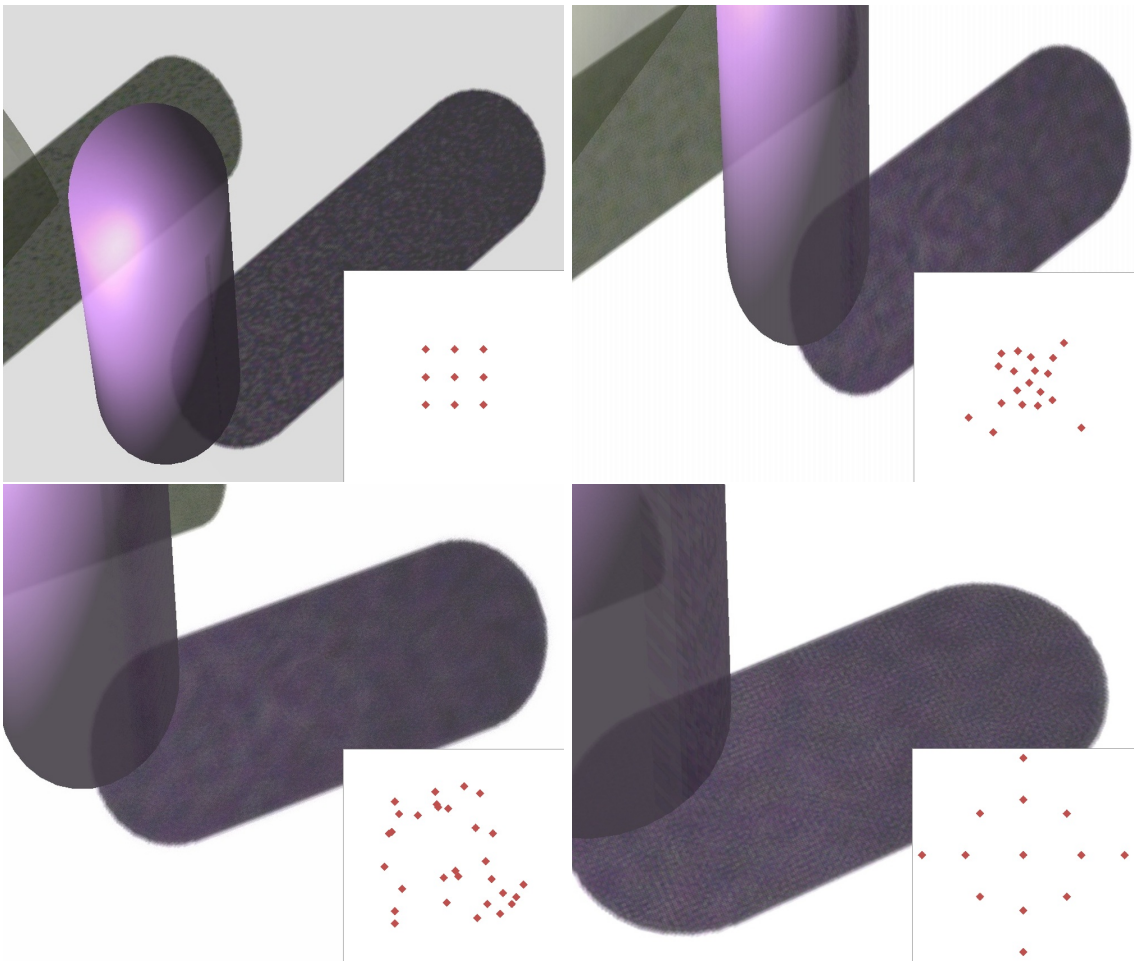
Z výsledků uvedených v tabulce 6.1 lze vidět, že demonstrační aplikace zobrazuje stíny v reálném čase při rozumné velikosti CSSM. Na základě vizuálních výsledků (viz. obrázek 5.1) posuzuji, že se barvy stínů i přes mírné odstínové odlišnosti blíží realitě. Míchání barev překrývajících se stínů probíhá realisticky. V porovnání se scénou vytvořenou autorem algoritmu (viz. obrázek 4.1, či 6.2) je však možné pozorovat několik odlišností. Hlavní odlišnost (tj. závislost sytosti stínů na počtu ploch průhledného materiálu, kterým projde paprsek světla) je způsobena tím, že jsem neimplementoval robustnější metodu na zobrazování průhledných materiálů. Hlavním důvodem bylo překročení rámce této práce, ale tento

nedostatek jsem částečně nahradil seřazením vykreslovaných objektů místo jednotlivých fragmentů. Jako pokračování projektu jsem v kapitole 6.2 navrhl kroky k dosažení velice podobných výsledků.

6.1 Porovnání výsledků rekonstrukčních filtrů

Při implementaci demonstrační aplikace jsem otestoval více druhů rekonstrukčních filtrů. Na obrázku 6.1 lze pozorovat kvalitu zobrazované scény při aplikaci různých rekonstrukčních filtrů. Například při zvolení devíti vzorků umístěných relativně v okolí zpracovávaného fragmentu probíhá výpočet osvětlení velice rychle, ale výsledná kvalita stínů není příliš dobrá.

Naopak při zvolení třiceti vzorků v rovnoměrném rozložení pravděpodobnosti počítaných přímo ve fragment shaderu jsou výsledné stíny poměrně kvalitní, ale zpracování trvá příliš dlouho.



Obrázek 6.1: Výsledné stíny při aplikaci různých rekonstrukčních filtrů a) PCF 9 vzorků (vlevo nahoře), b) poisson disc 20 vzorků (vpravo nahoře), c) rovnoměrné rozložení 30 vzorků (vlevo dole), d) upravený rekonstrukční filtr podle Mittringa [12] 13 vzorků (vpravo dole).

Dále jsem použil dvacet vzorků rozmístěných pomocí předem vytvořených souřadnic v rozložení Poisson disc. Poměr rychlosti zobrazení a kvality stínů je při použití tohoto filtru velice rozumný.

Nejzajímavější výsledky podával upravený rekonstrukční filtr podle Martina Mittringa, který se skládá z třinácti vzorků umístěných do tvaru kříže kombinovaného se čtvercem (viz. obrázek 4.6). Zde je ještě pro každý bod zvlášť počítán mikro-offset pomocí funkcí využívající hardwarové akcelerace. Tento rekonstrukční filtr podává velice dobré výsledky při nízkém počtu vzorků.

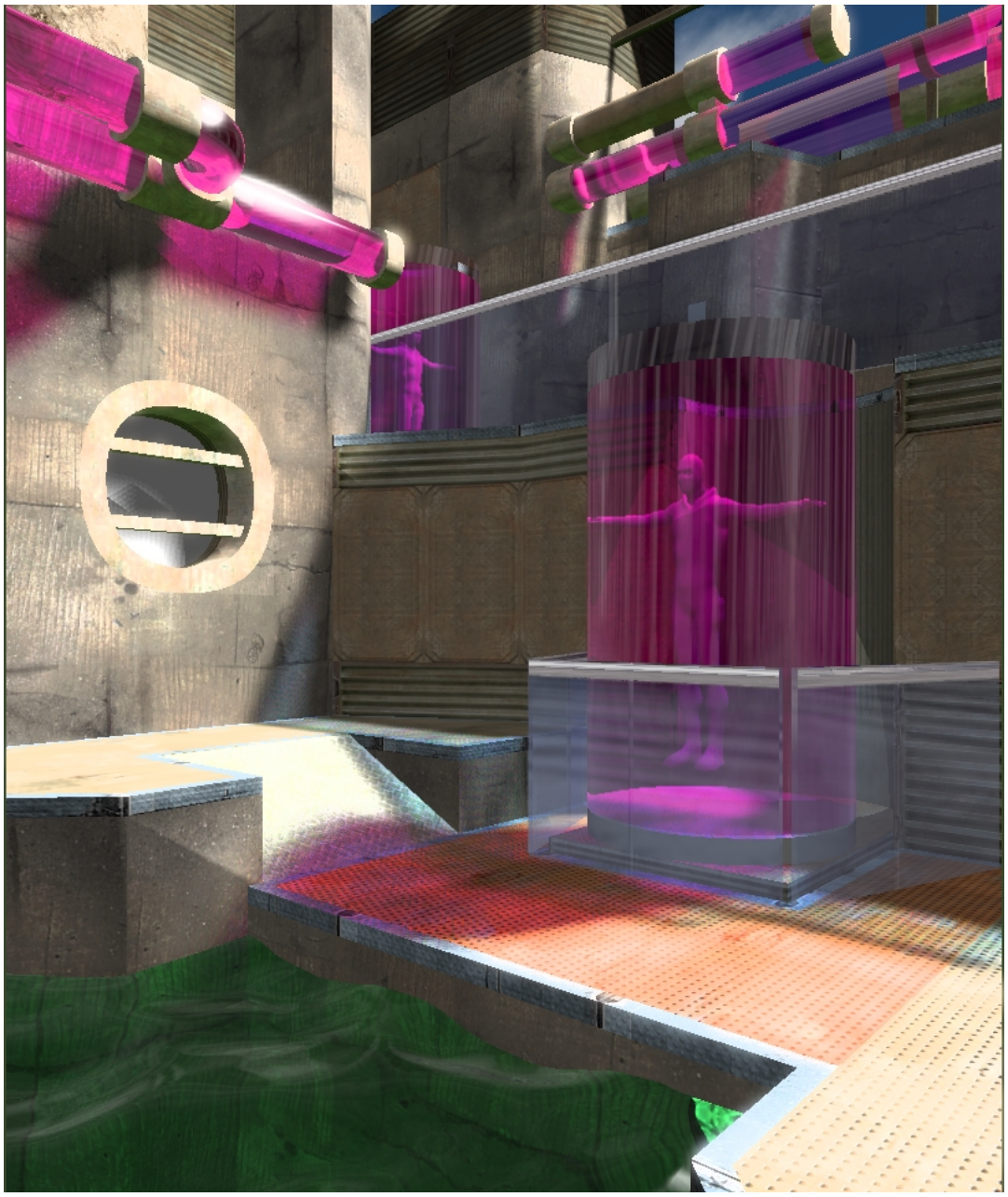
6.2 Možná rozšíření

Přestože aplikace splňuje vytyčené cíle, není ani zdaleka dokonalá. Zde navrhuji několik možností pokračování projektu:

První možnost je vytvořit podporu pro načítání modelů scény ve více různých formátech za běhu programu. Přestože tato úprava neovlivní samotný vykreslovací algoritmus, ale umožnila by testování algoritmu s různými modely a rozšířila by možnosti využití aplikace. V aktuální podobě lze bez úpravy kódu vykreslovat pouze ukázkovou scénu.

Dále by mohlo být vhodné aplikovat některá komplexnější rozšíření stínových map, za účelem zmírnění aliasu na okrajích stínů (konkrétně perspektivní aliasing). Například metodu Cascaded Shadow Maps [5]. Pro zvýšení reálnosti zobrazených stínů je možné aplikovat metodu Percentage-Closer Soft Shadows popsanou v publikaci [8]. Metoda zajišťuje, aby stíny blíže objektům, které je vrhají, byly ostré a se zvyšující vzdáleností více rozmazané.

Využití nějaké sofistikovanější metody pro zobrazování průhledných materiálů by přineslo znatelné vylepšení kvality zobrazení složitějších scén. Například algoritmus Depth Peeling. Také je možné vylepšit rekonstrukční filtr pro zjemnění odstínových odlišností barev stínů.



Obrázek 6.2: Herní scéna složena z více jak jednoho miliónu trojúhelníků. Zobrazena v rozlišení $1920x1080$ s CSSM v rozlišení 2048^2. Zdroj: [12].

Kapitola 7

Závěr

Cílem této práce bylo nastudovat algoritmus výpočtu stínů skrze průsvitné materiály a naimplementovat tento algoritmus ve vzorové aplikaci. Aplikaci jsem navrhl a implementoval. Aplikace zobrazuje scénu, kde jsou stíny vytvořeny pomocí algoritmu Colored Stochastic Schadow Maps. Scénou je možné libovolně procházet a měnit směr zdroje světla. Demonstrační aplikace je detailněji popsaná v kapitole 5.

Je možné měnit některé parametry, které jsou důležité, jak pro rychlost tvorby stínů, tak pro jejich vizuální kvalitu. Změnou těchto parametrů můžeme pozorovat, jak je důležitá jejich volba a není snadné určit ideální poměr mezi kvalitou a rychlostí vykreslování. Dále jsem experimentoval s více druhy rekonstrukčních filtrů a způsobů generování náhodných čísel pro sběr stochastických vzorků při tvorbě CSSM. Došel jsem k závěru, že jsou velice podstatné pro rychlost zobrazování a pro celkový vzhled zobrazené scény. Tento poměr je často relativní vzhledem k účelu výsledné aplikace.

Při implementaci aplikace jsem narazil na několik problémů a nedostatků základního algoritmu v praxi. Některé problémy se mi podařilo vyřešit a některé bohužel ne. To především kvůli náročnosti související s překročením rámce této práce. K dosažení velice kvalitních vizuálních výsledků by bylo zapotřebí implementovat další metody. Řešení těchto problémů jsem navrhl jako pokračování projektu v kapitole o možných rozšířeních 6.2.

Hlavním přínosem mé bakalářské práce je snaha inspirovat čtenáře k experimentování se stochastickými metodami, které mají podle mého názoru velký potenciál. Podstatné je také zjištění, že i taková v jistých směrech neoptimalizovaná implementace, jakou jsem představil, dokáže pro jednoduchou scénu zobrazovat stíny skrze průsvitné materiály v reálném čase. Při neustálém zvyšování výkonu grafických karet je velmi pravděpodobné, že se stochastické metody budou hojně využívat při tvorbě různých grafických knihoven, protože nabízejí mnoho podstatných výhod.

Literatura

[1] *Common Techniques to Improve Shadow Depth Maps (Windows)* [online]. 2012-11-22 [cit. 2013-03-14].

[2] *Tutorials for OpenGL 3.3 and later: Shadow mapping* [online]. 2013-03-09 [cit. 2013-03-14]. Dostupné na: <http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/>.

[3] Bavoil, L. et al. *Order independent transparency with dual depth peeling.* 2008.

[4] Bridson, R. Fast Poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 sketches.* New York, NY, USA: ACM, 2007. SIGGRAPH '07.

[5] Dimitrov, R. *Cascaded Shadow Maps.* 2007. Dostupné na: <http://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf>.

[6] Enderton, E. et al. Stochastic Transparency. In *I3D '10: Proceedings of the 2010 symposium on Interactive 3D graphics and games.* New York, NY, USA: [b.n.], 2010. S. 157–164. ISBN 978-1-60558-938-1.

[7] Everitt, C. *Interactive Order-Independent Transparency.* 2001.

[8] Fernando, R. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches.* New York, NY, USA: ACM, 2005. SIGGRAPH '05. Dostupné na: <http://doi.acm.org/10.1145/1187112.1187153>.

[9] Isidoro, J. R. Shadow Mapping: GPU-based Tips and Techniques. *GDC:06.* 2006. Dostupné na: <http://developer.amd.com/wordpress/media/2012/10/Isidoro-ShadowMapping.pdf>.

[10] Kilgard, M. J. *Shadow Mapping with Today's OpenGL Hardware.* 2001.

[11] Lokovic, T. et al. Deep shadow maps. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. S. 385–392. SIGGRAPH '00. Dostupné na: <http://dx.doi.org/10.1145/344779.344958>. ISBN 1-58113-208-5.

[12] McGuire, M. et al. Colored Stochastic Shadow Maps. February 2011. Dostupné na: <http://research.nvidia.com/publication/colored-stochastic-shadow-maps>.

[13] Phong, B. T. Illumination for computer generated pictures. *Commun. ACM.* 1975. S. 311–317.

[14] Tulleken, H. Poisson Disk Sampling. *Dev.Mag Issue 21.* 2008.

[15] Williams, L. Casting Curved Shadows on Curved Surfaces. In *In Computer Graphics (SIGGRAPH '78 Proceedings).* 1978. S. 270–274.