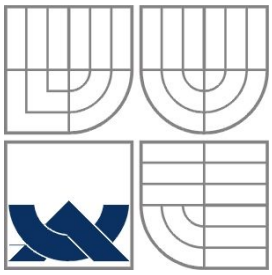
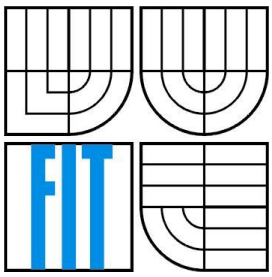


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

TECHNIKY VÝPOČTU STÍNŮ V REÁLNÉM ČASE

SHADOW COMPUTING TECHNIQUES IN REAL-TIME GRAPHICS

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE
AUTHOR

Bc. BORIS KUDLAČ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2009

Abstrakt

Cieľom tejto práce je vytvoriť súhrnný prehľad existujúcich techník počítačovej grafiky na výpočet tieňov v reálnom čase. Obsahuje teoretické základy jednotlivých techník, popis algoritmov, ako aj postup ich implementácie a použitie v praxi. Snaží sa kompletne rozobrať jednotlivé techniky, s ich vlastnosťami, problémami, rozšíreniami, výhodami a nevýhodami pre konkrétne situácie a popri tom odpovedať na otázku, či existuje jedna univerzálna technika vhodná pre všetky situácie. Nakoniec techniky porovnáva z hľadiska možností, obmedzení a v neposlednom rade hardwarovej náročnosti.

Kľúčové slová

tiene, techniky tieňov, počítačová grafika, tiene v reálnom čase, tieňové teleso, tieňové mapy, rovinné tiene

Abstract

The goal of this work is to create complete overview about existing techniques for real-time shadow computation in computer graphics. It describes theory of each technique and it's algorithm, as well as implementation details and practical usage. It completely informs about each technique features, problems, extensions, advantages and disadvantages for different situations and asks a question, if there is one all-solving shadow technique suitable for all purposes. In the end, the thesis compares all the techniques in their features, limitations and hardware usage.

Keywords

shadows, shadow techniques, computer graphics, real-time shadows, shadow volume, shadow maps, planar shadows

Citácia

Boris Kudlač: Techniky výpočtu tieňov v reálnom čase, diplomová práce, Brno, FIT VUT v Brně, 2009

Techniky výpočtu tieňov v reálnom čase

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Adama Herouta, Ph.D.

Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Boris Kudlač
26.5.2009

Poďakovanie

Chcel by som poďakovať vedúcemu diplomovej práce, za ochotu prejavenú pri konzultáciách a poskytnutie odbornej pomoci.

© Boris Kudlač, 2009.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Obsah....1
1 Úvod....3
1.1 Význam tieňov v počítačovej grafike....4
1.2 Typy tieňov....5
2 Rovinné tiene....10
2.1 Princíp algoritmu....10
2.2 Matica tieňa....12
2.3 Rovinné tiene na ďalších rovinách....12
2.4 Iný prístup k výpočtu rovinných tieňov....13
2.5 Problémy....15
2.6 Rozšírenia....19
3 Tieňové mapy....20
3.1 Princíp algoritmu....20
3.2 Tvorba tieňovej mapy....21
3.3 Zobrazenie tieňov v scéne....22
3.3.1 Prevod súradníc do priestoru svetla....23
3.3.2 Test hĺbkovej mapy....24
3.3.3 Vrhanie tieňov....25
3.4 Problémy....27
3.4.1 Aliasing na okrajoch tieňov....27
3.4.2 Aliasing vlastného zatienenia....29
3.5 Rozšírenia....32
3.5.1 Perspektívne tieňové mapy....32
3.5.2 Percentage-closer filtering (PCF)....34
3.5.3 Ďalšie rozšírenia tieňových máp....36
4 Tieňové telesá....39
4.1 Depth-pass algoritmus....41
4.2 Depth-fail algoritmus....43
4.3 Problémy....44
4.3.1 Problémy algoritmu depth-pass....44
4.3.2 Problémy algoritmu depth-fail....45
4.4 Rozšírenia....46
5 Techniky zobrazovania tieňov v praxi....47
5.1 Implementácia vybraných techník....47

5.1.1 Rovinné tiene.....47
5.1.2 Tieňové mapy.....51
5.1.3 Tieňové telesá.....59
5.2 Porovnanie vybraných techník.....66
5.2.1 Rovinné tiene.....66
5.2.2 Tieňové mapy.....68
5.2.3 Tieňové telesá.....69
6 Záver.....72
Literatúra.....73
Zoznam príloh.....76

1 Úvod

V oblasti počítačovej grafiky existuje mnoho postupov, ktorými môžeme docieliť efekt vrhania tieňov. Výpočet tieňov je s rôznymi metódami rôzne výpočetne náročný. Niekedy je výpočet tieňov súčasťou výpočtu osvetlenia. Existujú systémy, v ktorých sa osvetlenie a tiene počítajú pomocou zložitých postupov dodržujúcich fyzikálne zákony. Takéto systémy však nepracujú v reálnom čase, to znamená, že nedokážu vykresľovať obraz dostatočne rýchlo pre ľudské vnímanie (25-30 obrázkov za sekundu). V minulosti bolo možné presne riešiť zobrazovanie tieňov iba v takýchto, neinteraktívnych systémoch.

Dnes je však situácia iná. Ako mnoho iných problémov, niekedy riešiteľných iba v neinteraktívnych systémoch, sa aj problém výpočtu tieňov presunul do oblasti počítačovej grafiky pracujúcej v reálnom čase. Zásluhu na tom majú jednak neustále sa zlepšujúce a zrýchľujúce hardwarové vybavenie, a jednak ľudia, ktorí (taktiež neustále) prichádzajú s novými nápadmi a riešeniami. Zobrazenie tieňov objektov v scéne, vo virtuálnej krajine, kde sa môže užívateľ pohybovať v reálnom čase, je tak už dnes pomerne bežnou záležitosťou. Samozrejme, že zobrazenie tieňov nie je vždy úplne fyzikálne presné, ale postačuje nám na celistvejšie a reálnejšie vnímanie scény. Kvalita techník riešiacich problém tieňov sa líši. Táto práca sa snaží zachytiť techniky, ktoré dosahujú prijateľné výsledky zobrazenia tieňov v reálnom čase.

Prvá kapitola otvára problematiku tieňov v počítačovej grafike. Zaoberá sa významom tieňov na vnímanie scény, a to z hľadiska perspektívneho vnemu a reálnosti vnímania scény. Ďalej rozdeľuje tiene do kategórií podľa kvality, charakteru tieňov a dynamičnosti.

Nasledujú kapitoly, ktoré sa zaoberajú teoretickou časťou vybraných techník. Obsahujú princípy algoritmov, popis vlastností techník, jej problémy a možnosti rozšírenia.

Druhá kapitola sa venuje najjednoduchšej technike, rovinným tieňom. Popisuje matematický základ algoritmu, tvorbu tieňovej matice a dva rôzne prístupy k riešeniu rovinných tieňov.

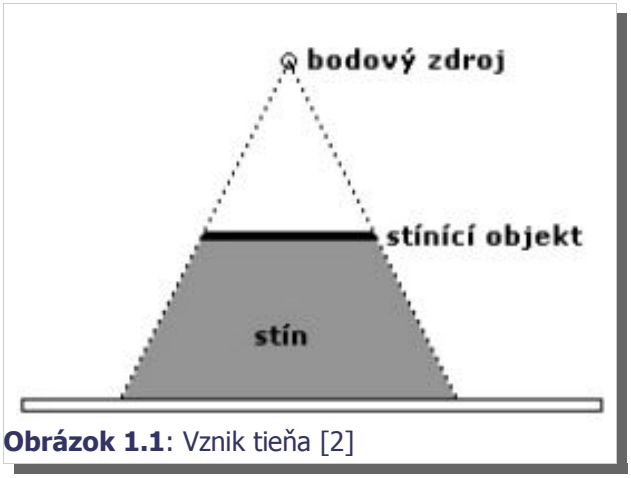
V tretej kapitole sa nachádza popis techniky tieňových máp. Opäť je popísaný princíp algoritmu, tvorba tieňovej mapy a spôsob výsledného zobrazenia tieňov. Ďalej táto kapitola rozoberá problémy techniky a navrhuje ich riešenie. Koniec kapitoly patrí rozšíreniam pôvodného algoritmu.

Technika tieňových telies je predmetom štvrtej kapitoly. Informuje o dvoch hlavných prístupoch výpočtu tieňov pomocou tejto techniky, ich problémoch a spôsobu rozšírenia.

Poslednou kapitolou je kapitola piata. Rozpráva o konkrétnej implementácii vybraných techník, porovnáva jednotlivé techniky z hľadiska kvality, možností, zložitosti implementácie a nakoniec testuje výpočetnú náročnosť.

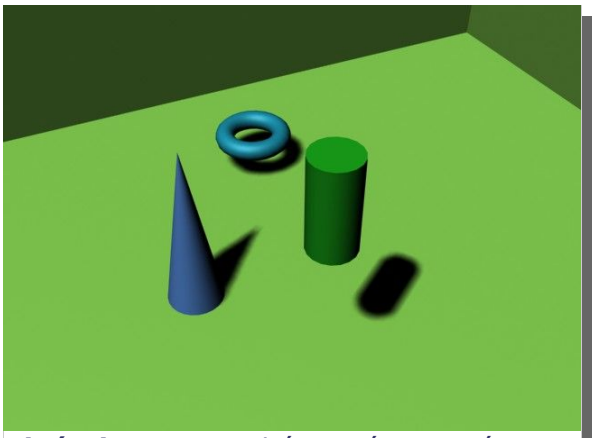
1.1 Význam tieňov v počítačovej grafike

Každý objekt, ktorý stojí v ceste svetelným lúčom vychádzajúcim zo svetelného zdroja, vrhá tieň. Takýto objekt sa nazýva **tieniaci objekt** (angl. *occluder* alebo *blocker*). Aby bol však vrhnutý tieň viditeľný potrebuje ešte jeden objekt, na ktorý tieň dopadne. Tento objekt sa nazýva **príjemca** (angl. r*eciever*).

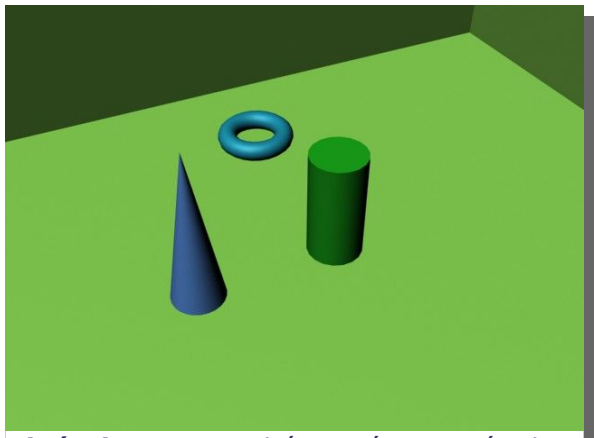


Obrázok 1.1: Vznik tieňa [2]

Tiene hrajú veľkú úlohu v priestorovom vnímaní scény. Dávajú nám vizuálnu informáciu o polohe objektov vzhľadom na iné objekty, o ich vzájomnej vzdialenosti, o ich tvare a v neposlednom rade nám napomáhajú určiť miesto svetelného zdroja. Bez týchto informácii je obtiažne pochopiť všetky súvislosti scény. Nedokážeme napr. správne odhadnúť, či určitý objekt leží na podložke alebo je nad ňou.



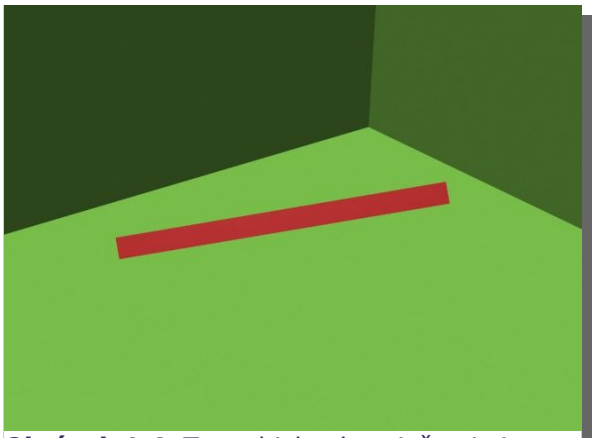
Obrázok 1.2: Perspektívne vnímanie scény s tieňmi



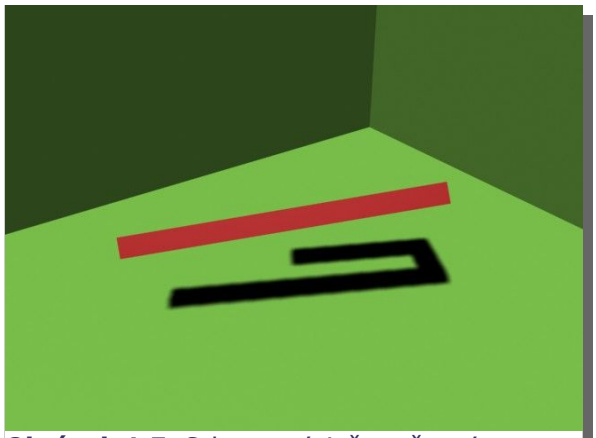
Obrázok 1.3: Perspektívne vnímanie scény bez tieňov

Vľavo scéna zobrazená bez tieňov, podľa ktorej by sme mohli predpokladať, že všetky objekty ležia v jednej rovine. Vpravo tá istá scéna zobrazená s tieňmi, ktoré nám poskytujú ďalšie informácie o polohe objektov.

Ako keby nám tiene dávali možnosť pozrieť sa na scénu z iného pohľadu, a to z pohľadu svetla. Dodávajú nám ďalšie informácie v podobe geometrických pomôcok, podľa ktorých si vieme domyslieť ďalšie súvislosti. Ľudský mozog si potom celú situáciu vyhodnotí oveľa presnejšie, môžeme napr. určiť tvar objektu, ktorý je inak priamym pohľadom nezreteľný alebo vôbec nezistiteľný.



Obrázok 1.4: Tvar objektu bez tieňu nie je zreteľný



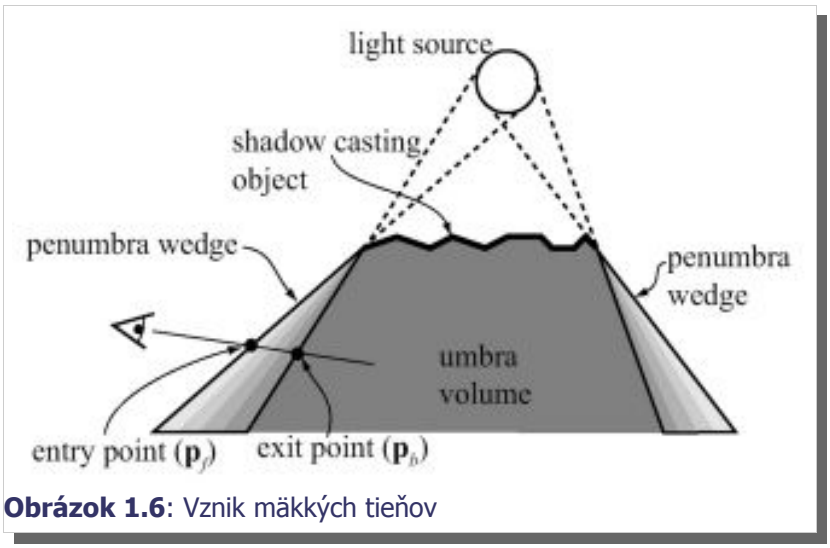
Obrázok 1.5: Schopnosť tieňu určovať tvar objektu

Pri určitom uhle pohľadu nemusí byť tvar objektu zreteľný. Objekt naľavo by sme mohli považovať za hranol. Zobrazenie tieňu nám však ukazuje skutočný tvar objektu.

Keďže tiene zohrávajú tak veľkú úlohu pri vnímaní celej scény, snažíme sa v počítačovej grafike o ich čo možno najpresnejšie zobrazenie. Čím ďalej, tým viac, sa počítačová grafika posúva k reálnemu stvárneniu scény, a k tomu jednoznačne prispieva aj kvalitné zobrazenie tieňov. Platí, že čím kvalitnejšie je ich zobrazenie, tým realistickejšie scénu vnímame, čo potvrdzujú aj vedecké experimenty. V reálnom svete predstavuje vrhanie tieňov fyzikálny jav spojený s fotónovým vnímaním svetla, čo je pre svet grafiky a počítačov ako takých výpočetne veľmi náročná záležitosť. Existujú síce riešenia, ktoré problém tieňov riešia na úrovni fotónov, no ich náročnosť sa nehodí pre interaktívne aplikácie. Ak chceme scénu zobrazovať v reálnom čase, neostáva nám (zatiaľ) nič iné, ako si problém zjednodušiť, zabstrahovať a preniesť ho na niečo výpočetne menej náročné. V riešeniach, ktoré poskytujú výsledky v reálnom čase sa teda nesnažíme o fyzikálne presné napodobenie reálnej situácie, ale väčšinou dosahujeme podobného výsledku zjednodušením na geometrické problémy, resp. na riešenie viditeľnosti. Existuje viacero algoritmov na zobrazenie tieňov v real-time grafike, každý so svojimi výhodami a nevýhodami, možnosťami použitia v určitých situáciach a samozrejme každý s inou výpočetnou zložitosťou. Ďalej sa líšia kvalitou zobrazenia tieňov, schopnosťou vrhať vlastné tiene alebo vytvárať mäkké tiene. Niektoré z týchto pojmov vysvetlíme v nasledujúcich odstavcoch.

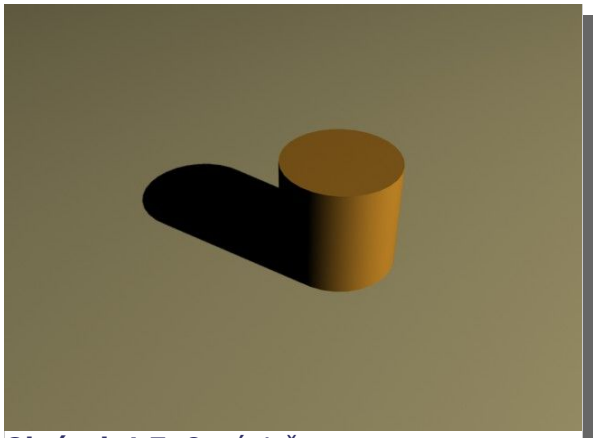
1.2 Typy tieňov

Aby bol náš slovník terminológie tieňov kompletný, musíme spomenúť ešte termíny *umbra* a *penumbra*. Umbra (latinsky *tieň*) je oblasť príjemcu plne zatienená tieniacim objektom. Intenzita tieňu je tu najväčšia, vyjadrené percentuálne ide o 100%. Penumbra je oblasť, kde tieň nie je úplne tmavý, čo nastáva na okrajoch tieňu, nazýva sa tiež **polotieň**. Intenzita polotieňu závisí od polohy svetelného zdroja a platí, že klesá so vzrastajúcou vzdialenosťou od plného tieňu. Oblasť, kam dopadá polotieň, je teda zatienená iba čiastočne. Polotieň je javom, ktorý vzniká iba pri svetelných zdrojoch, ktoré sú plošné. Ak hovoríme o plošných svetelných zdrojoch, umbre a penumbre, hovoríme o druhu tieňov, ktoré sa nazývajú **mäkké** (*soft shadows*).

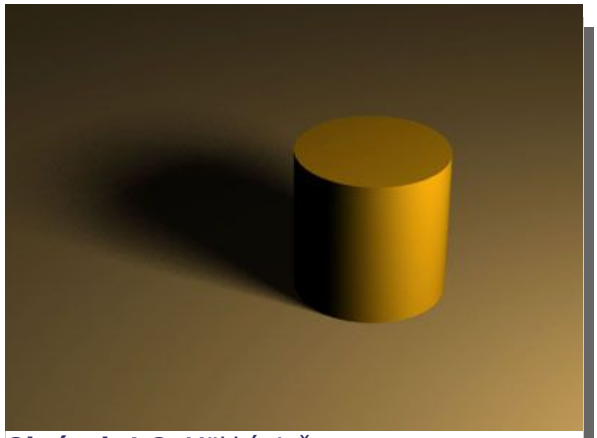


Obrázok 1.6: Vznik mäkkých tieňov

V reálnom svete sú síce všetky svetelné zdroje viac-menej plošné, ale v počítačovej grafike sa často kvôli zjednodušeniu používajú zdroje bodové. Plošné svetelné zdroje sa potom simulujú pomocou viacerých bodových zdrojov. Čím presnejšie chceme mäkké tiene zobraziť, tým viac bodových svetiel musíme použiť, čo samozrejme stojí výpočetný čas. Často sa musí tieň vrhať vo viacerých prechodoch (pre každé bodové svetlo jeden prechod), preto sa na stvárnenie mäkkých tieňov používajú aj iné, zjednodušujúce techniky. Využívajú sa pomôcky ako je napríklad filtrovanie okrajov tieňov, čím sa síce na oko dosiahne hladkých a mäkkých tieňov, v tomto prípade sa však jedná o nepravé mäkké tiene.



Obrázok 1.7: Ostrý tieň



Obrázok 1.8: Mäkký tieň

Vľavo mäkký tieň vrhnutý objektom, ktorý blokuje plošný svetelný zdroj. Vpravo podobná scéna s bodovým svetelným zdrojom a ostrým tieňom.

Ak je zdroj svetla bodový alebo smerový (svetelné lúče sú rovnobežné), tiene vytvorené týmito druhmi zdrojov nazývame **ostré** (*hard shadows*). Intenzita ostrého tieňu je vo všetkých jeho miestach rovnaká a zároveň maximálna. Nevytvára sa žiadny polotieň. Ostré tiene síce neposkytujú najrealistickejšie zobrazenie scény, vo väčšine prípadov však plnia svoju úlohu dobre. Je to dané aj tým, že ľudský mozog ich neodmieta ako niečo nereálne, pretože aj v reálnom svete môžu nastať situácie, kedy sa nám tiene objektov môžu zdať ako ostré. Príkladom je napr. ostrý tieň vytvorený po osvietení objektu LED diódou v inak neosvetlenej miestnosti. LED dióda zohráva v tomto prípade rolu bodového svetelného zdroja. Ostré tiene vznikajú aj pri osvetlení smerovými svetelnými zdrojmi. Za takýto zdroj môžeme považovať napr. slnko, pretože je od nás veľmi vzdialené a jeho lúče môžeme

považovať za takmer rovnobežné (ak uvažujeme o tieňoch vrhnutých napr. postavou človeka). Tiene, ktoré nás každodenne sprevádzajú za slnečného počasia na uliciach, majú teda veľmi blízko ostrým tieňom.

Obrázok 1.9: (Nepravé) mäkké tiene v hre Crysis (2007, Crytek)

Hlavnou výhodou ostrých tieňov je, že sú niekoľkonásobne časovo menej náročné na výpočet ako tiene mäkké. Preto sa v minulosti hojne využívali v interaktívnych aplikáciách a počítačových hrách a používajú sa aj v súčasnosti ako základné nastavenie zobrazovania tieňov. So vzrastajúcou výpočetnou silou moderných grafických kariet sa však začína čím ďalej, tým viac, presadzovať napodobnenie mäkkých tieňov. V novších herných tituloch je kvalitné stvárnenie tieňov veľmi dôležité pre dotvorenie realistickej grafickej stránky, preto sa začína prechádzať od jednoduchých algoritmov k zložitejším a teda i ku kvalitnejšiemu výpočtu tieňov. Často sa napr. používa filtrovanie okrajov tieňov, ktoré vytvára dojem mäkkého tieňa.

Ako sme už spomínali, existuje viacero techník na výpočet tieňov v reálnom čase. Táto práva sa bude zaoberať najčastejšie používanými technikami, ktorými sú techniky produkujúce ostré tiene.

Algoritmy na výpočet ostrých tieňov:

- Rovinné tiene (P*lanar Shadows*)
- Tieňové mapy (*Shadow Maps*)
- Tieňové telesá (*Shadow Volumes*)

Podľa interakcie s okolím môžeme tiene rozdeliť do dvoch kategórií:

- **vlastné tiene**
- **vrhané tiene**

Obrázok 1.10: Vlastné tiene (self shadows)

Interakciou s okolím myslíme správanie sa tieňov vzhľadom na iné objekty a vzhľadom na objekty, ktoré sú ich pôvodcami. Vrhaným tieňom (*cast shadow*) označujeme tieň, ktorý vrhá jeden objekt na druhý. Vlastný tieň (*self shadow*) je potom tieň, ktorý vrhá objekt sám na seba. Vlastné tiene sú podstatnou súčasťou tvorby tieňov, pretože nám opäť môžu napovedať dôležité informácie o tvare objektu a jeho orientácii v scéne. Každá plocha odvrátená od svetelného zdroja leží vo vlastnom tieni, to však nie je až také podstatné, pretože odvrátené plochy sú zatmavené už pri výpočte osvetlenia. Preto sa zameriavame na tiene vrhnuté jednou časťou telesa na druhú. Nie všetky techniky vytvárajú vlastné tiene, niektoré zvládajú iba vrhané tiene, to však nemusí byť na škodu a dá sa to v určitých situáciách využiť.

Podľa dynamičnosti môžeme ďalej tiene rozdeliť na:

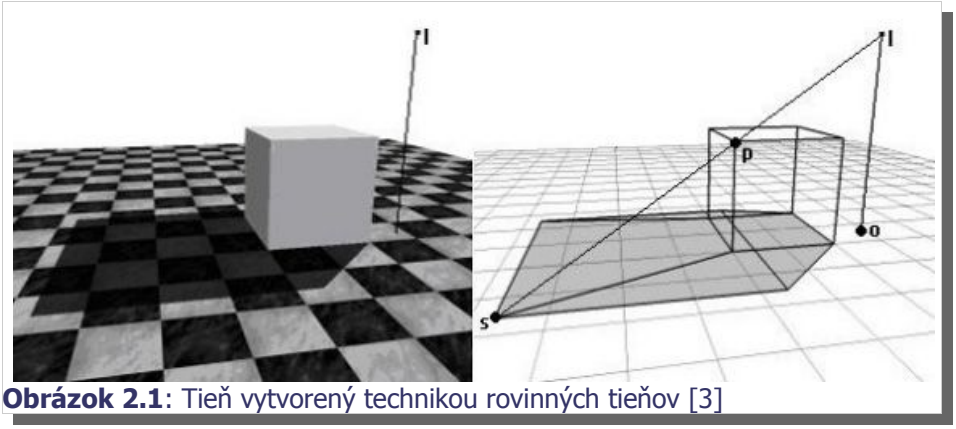
- **statické**
- **dynamické**

Statické tiene nemenia svoju polohu so zmenou polohy svetla. Nereagujú ani na pohyb objektu, ktorý je ich pôvodcom. Môžu byť pritom ale veľmi realisticky zobrazené. Do tejto kategórie patria tiene predpočítané a následne zavedené do scény v podobe textúry. Dokonca sa niekedy informácia o tieni môže zapisovať do geometrie objektov [5]. Ich hlavnou výhodou je, že sa nemusia vypočítavať počas behu aplikácie a pritom môžu dobre plniť svoju vizuálnu funkciu (budova môže mať statický tieň, pokiaľ sa pozícia slnka nemení). Tento druh tieňov je často vidieť v počítačových hrách, aj keď ho už čím ďalej, tým viac nahrádzajú tiene dynamické.

Tiene dynamické sú tieňmi interaktívnymi. Reagujú na zmenu polohy svetla aj objektov. Na rozdiel od tieňov statických sa ich zobrazenie vypočítava v každom snímku aplikácie, sú teda výpočetne náročnejšie. Čo sa týka kvality a fyzikálnej presnosti, niekedy nemusia poskytovať realistickejší vizuálny vnem ako tiene statické, ale doháňajú to svojou interaktivitou. Používajú sa všade tam, kde je prítomný pohyb, animácia. Využívajú sa v počítačových hrách na zobrazenie tieňov pohyblivých postáv a objektov. Všetky techniky, ktoré táto práca rozoberá, produkujú dynamické tiene.

2 Rovinné tiene

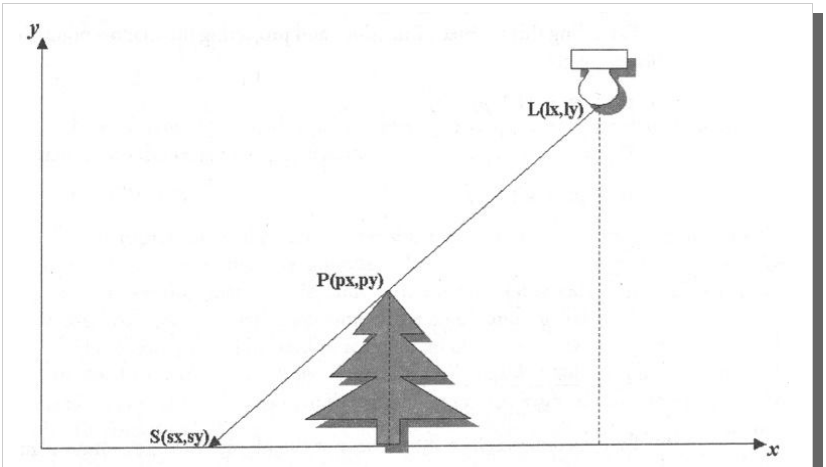
Rovinné tiene (označujú sa ako *planar shadows*, *ground-plane shadows* alebo *flat shadows*.) predstavujú najjednoduchšie riešenie problému vrhania tieňov. Často je potreba, aby objekt alebo postava vrhala tieň iba „pod seba", na povrch, na ktorom stojí, prípadne na priľahlú stenu atď. V prípade, že povrch nie je komplexný a môžeme ho považovať za rovinu, môžeme použiť techniku rovinných tieňov.



Obrázok 2.1: Tieň vytvorený technikou rovinných tieňov [3]

2.1 Princíp algoritmu

Základnou myšlienkou metódy je vykresliť „pod objekt" resp. na príslušnú rovinu plochú (dvojrozmernú) verziu objektu vhodne skosenú podľa uhlu dopadajúceho svetla. Je teda treba nájsť transformáciu, ktorá prevedie objekt v 3D priestore do 2D roviny, tak aby výsledný 2D objekt zodpovedal tieňu objektu pri danej pozícii svetla. V transformácii bude vystupovať pozícia objektu, pozícia svetla a rovina, na ktorú tieň vrháme. Transformáciu použijeme na všetky vrcholy objektu, čím ich premietneme na povrchovú plochu. Takto premietnuté body vytvoria transformované polygóny, ktoré môžeme použiť na vykreslenie tieňu objektu.



Obrázok 2.2: Svetlo v bode L vrhá tieň bodu P na rovinu dopadu do bodu S [1]

Obrázok ukazuje vrhanie tieňov na rovinu v 2D. Bod ***P*** na objekte vrhá tieň zo svetla ***L*** na určitý bod ***S*** roviny ***x***. Naším cieľom je nájsť súradnice bodu ***S***. Z podobnosti trojuholníkov dostávame vzťah:

$$\frac{sx - px}{sx - lx} = \frac{py}{ly}$$

Vzorec 2.1

z čoho:

$$sx = px + py * (\frac{sx - lx}{ly})$$

Vzorec 2.2

V tomto vzťahu však vystupuje ***sx*** na pravej strane. Jediný výskyt ***sx*** na pravej strane je vo výraze ***(sx - lx)***, čo predstavuje horizontálnu vzdialenosť bodu ***S*** od svetla ***L***. Za predpokladu, že dĺžka tieňu je v porovnaní so vzdialenosťou svetla od objektu veľmi malá, môžeme výraz ***(sx - lx)*** nahradiť výrazom ***(ox - lx)***, kde ***ox*** predstavuje ***x***-ovú súradnicu objektu, ide teda o horizontálnu vzdialenosť objektu od svetla. Po nahradení teda dostávame:

$$sx = px + py * (\frac{ox - lx}{ly})$$

Vzorec 2.3

alebo:

$$sx = px + k * py$$

kde

$$k = (\frac{ox - lx}{ly})$$

Toto zjednodušenie, resp. predpoklad, že svetlo je od objektu vo veľkej diaľke, predstavuje vlastne vnímanie svetla ako smerového a nie ako bodového zdroja. Ak tento výpočet presunieme do 3D priestoru a budeme vrhať tieň na rovinu ***xz***, budú vzťahy vypadať nasledovne:

$$sx = px + k1 * py$$
$$sy = 0$$
$$sz = pz + k2 * py$$

kde

$$k1 = (\frac{ox - lx}{ly}) \quad \text{a} \quad k2 = (\frac{oz - lz}{ly})$$

sú kompletne závislé iba na relatívnej pozícii objektu a svetla.

2.2 Matica tieňa

Na základe týchto vzťahov môžeme vytvoriť tzv. **maticu tieňa** (*shadow matrix*). Vieme, že všetky transformácie, ktoré sa počas vykresľovania dejú, sú realizované maticovými operáciami, presnejšie násobením jednotlivých transformačných matíc. Tieňová matica bude jednou z týchto transformačných matíc, pomocou ktorej vytvoríme tieň objektu na určitej rovine. Jedná sa o maticu 4x4 a bod tieňu vyjadríme ako:

$$s = M * p$$

kde

$$s = (sx, sy, sz, 1)$$

je bod tieňu premietnutý na príslušnú rovinu v súradnicovom systéme scény

$$p = (px, py, pz, 1)$$

je bod objektu v súradnicovom systéme scény a ***M*** je projekčná matica tieňu (*shadow matrix*).

$$M = \begin{bmatrix} 1 & k1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & k2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Pomocou tejto matice potom realizujeme transformáciu objektu, bod po bode, na tieň objektu. Všetky body premietnutého objektu sa budú nachádzať v jednej rovine. Tieň bude zodpovedať projekcii objektu na plochu z pohľadu svetelného zdroja. A to kompletne, teda s tieňovaním, textúrami a pod. Vykreslenie scény objektu s vrhnutým tieňom pozostáva potom zo samotného vykreslenia daného objektu, následne sa vykreslí jeho transformovaná verzia vo farbe tieňa. Tieňovú verziu objektu vykresľujeme pomocou jednoduchšieho konštantného tieňovania polygónov. Taktiež môžeme úplne ignorovať informácie o textúre objektu. Pre urýchlenie výpočtu sa dá použiť aj model s menším množstvom polygónov ako pôvodný objekt, obrysová informácia však musí súhlasiť s pôvodným objektom.

2.3 Rovinné tiene na ďalších rovinách

Na začiatku tejto kapitoly sme uvažovali iba o vrhaní tieňov na rovinu ***xz*** alebo inak tiež rovinu ***y = 0***. Nie je problém pozmeniť tento algoritmus tak, aby boli tiene vrhané na roviny ***yz (x = 0)*** alebo ***xy (z = 0)***. Vzťahy z kapitoly 2.1 by potom vypadali nasledovne:

$$sx = 0$$
$$sy = py + k1 * px$$
$$sz = pz + k2 * px$$

kde

$$k1=(\frac{oy-ly}{lx}) \quad \text{a} \quad k2=(\frac{oz-lz}{lx})$$

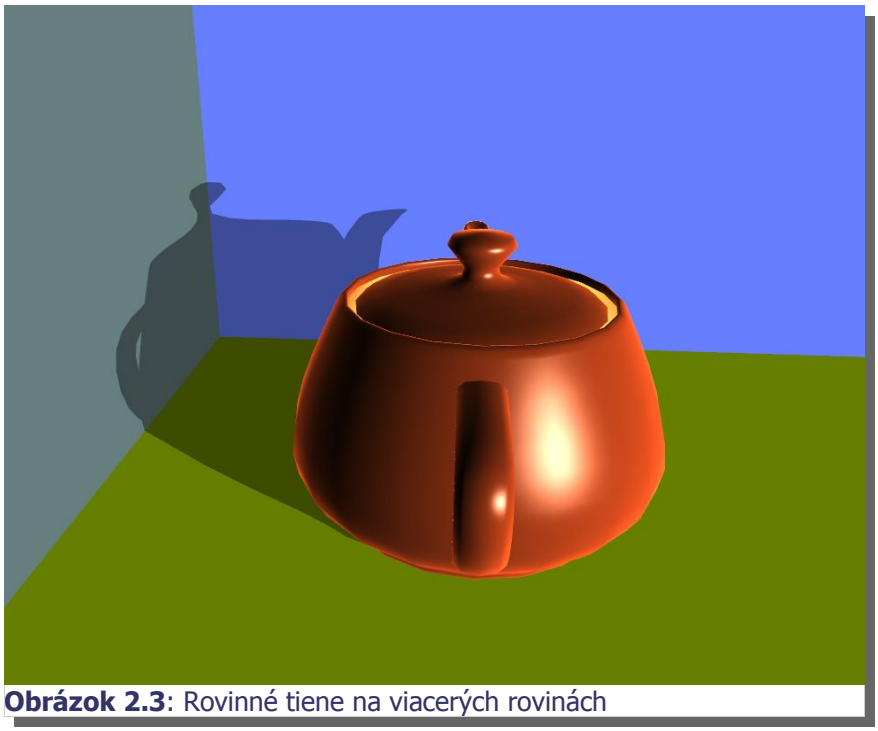
pre rovinu ***yz*** *(**x** = 0)*, a

$$\begin{aligned} sx&=px+k1*pz \\ sy&=py+k2*pz \\ sz&=0 \end{aligned}$$

kde

$$k1=(\frac{ox-lx}{lz}) \quad \text{a} \quad k2=(\frac{oy-ly}{lz})$$

pre rovinu ***xy*** *(**z** = 0)*.



Obrázok 2.3: Rovinné tiene na viacerých rovinách

2.4 Iný prístup k výpočtu rovinných tieňov

Predchádzajúci algoritmus bral do úvahy svetlo ako smerový zdroj, lúče svetla teda považoval za navzájom rovnobežné. Išlo hlavne o zjednodušenie procesu transformácie objektu do roviny tieňu tak, aby bola projekčná matica tieňu čo najjednoduchšia. Nedostatok, z toho vyplývajúci, je potom nerealistickosť produkovaných tieňov, ak je zdroj svetla blízko objektu. Tiene sa so vzdialenosťou svetla od objektu nijak nezväčšujú ani nezmenšujú, čo značne ovplyvňuje použiteľnosť tejto techniky.

Nasledujúci algoritmus bude počítať so svetlom ako s bodovým zdrojom. Už v roku 1988 ho publikoval James Blinn [4]. Východzia situácie je opäť rovnaká ako na obrázku 2.2. Svetlo *L* teda vrhá tieň bodu *P* na bod roviny *S* a naším cieľom je zistiť súradnice bodu *S*. Nebudeme však na ich výpočet využívať podobnosť trojuholníkov, ale parametrické vyjadrenie priamky, pričom priamkou myslíme lúč svetla prechádzajúci bodom *P*. Priamka prechádza taktiež bodom *S* a preto môžeme zapísať:

$$S = L + t(P - L)$$

Vzorec 2.4

kde

$$S = (sx, sy = 0, sz)$$

$$L = (lx, ly, lz)$$

$$P = (px, py, pz)$$

a *t* je parameter.

Keďže vieme, že $sy = 0$, dostaneme z rovnice $sy = ly + t(py - ly)$, že

$$t = \frac{ly}{(ly - py)}$$

Súradnice bodu *S* teraz už ľahko vypočítame dosadením parametra *t* do vzťahu (2.4).

Výsledná transformácia bude mať opäť podobu:

$$S = M * P$$

alebo

$$\begin{pmatrix} sx \\ 0 \\ sz \\ 1 \end{pmatrix} = \begin{bmatrix} l_y & -l_x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -l_x & l_y & 0 \\ 0 & -1 & 0 & l_y \end{bmatrix} \cdot \begin{pmatrix} px \\ py \\ pz \\ 1 \end{pmatrix}$$

Tým sme dostali transformačnú maticu *M* pre rovinu ***xz (y = 0)***

$$M=\begin{bmatrix} l_y & -l_x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -l_x & l_y & 0 \\ 0 & -1 & 0 & l_y \end{bmatrix}$$

Ak chceme vypočítať transformačnú maticu pre ľubovoľnú rovinu zadanú rovnicou roviny

$$\vec{n}.\vec{r}+d=0$$

kde

$$\vec{n}=(a,b,c)$$

je normálový vektor roviny a

$$\vec{r}=(rx,ry,rz)$$

je ľubovoľný bod ležiaci na rovine, musíme za *r* dosadiť náš bod *S*, teda bod tieňu a následne vypočítať nový parameter *t*. Rovnica potom bude vypadať nasledovne:

$$a.sx+b.sy+c.sz+d=0$$

a po dosadení:

$$a(lx+t(px-lx))+b(ly+t(py-ly))+c(lz+t(pz-lz))+d=0$$

z čoho potom dostaneme parameter *t*:

$$t=\frac{\vec{n}\vec{l}+d}{\vec{n}(\vec{l}-\vec{p})}$$

Parameter potom opäť dosadíme do vzťahu (2.4) a nakoniec dostávame transformačnú maticu

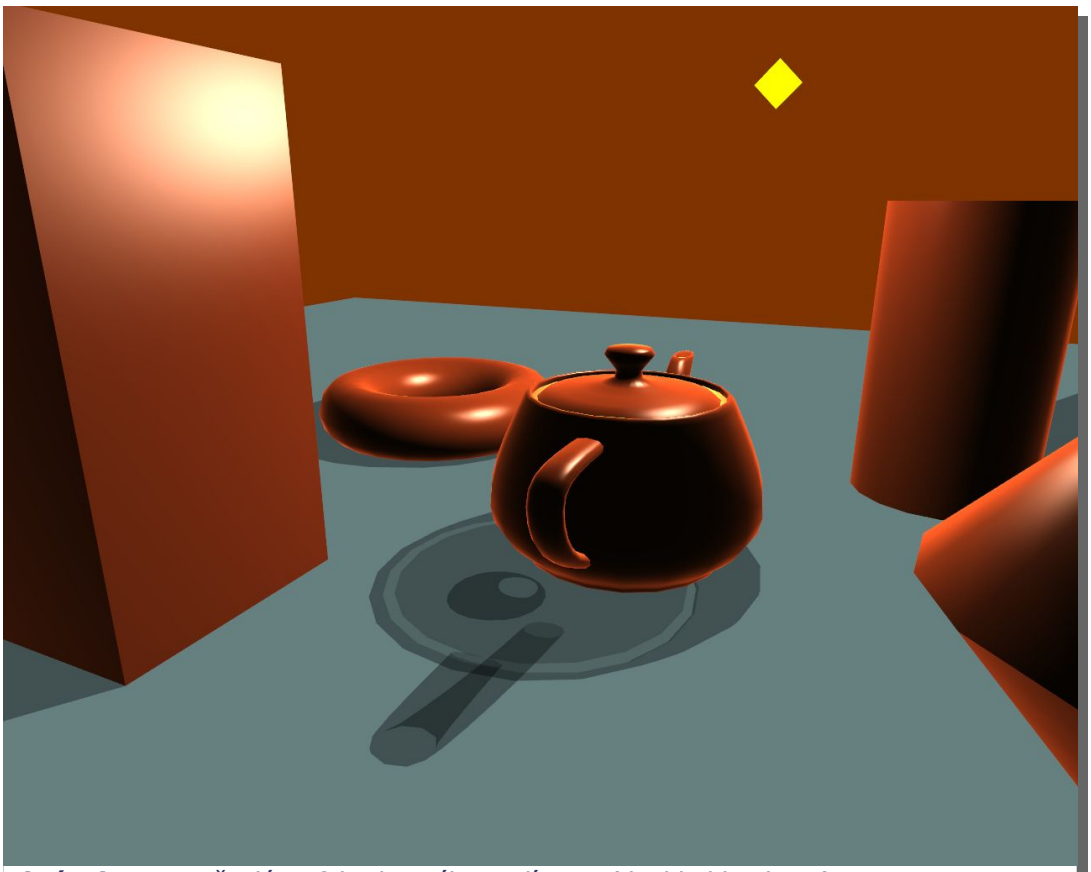
$$M=\begin{bmatrix} \vec{n}\vec{l}+d-l_x n_x & -l_x n_y & -l_x n_z & -l_x d \\ -l_y n_x & \vec{n}\vec{l}+d-l_y n_y & -l_y n_z & -l_y d \\ -l_z n_x & -l_z n_y & \vec{n}\vec{l}+d-l_z n_z & -l_z d \\ -n_x & -n_y & -n_z & \vec{n}\vec{l} \end{bmatrix}$$

Transformačná matica tieňa

Aplikáciou tejto matice môžeme objekt premietnuť do roviny a vykresliť ho ako tieň. Výsledok však nebude úplne podľa očakávania. Nastáva tu viacero neželaných sprievodných javov, žiaden z nich však nie je neriešiteľný. Problémy, ktoré pri tejto metóde môžu nastať si popíšeme v nasledujúcom odstavci.

2.5 Problémy

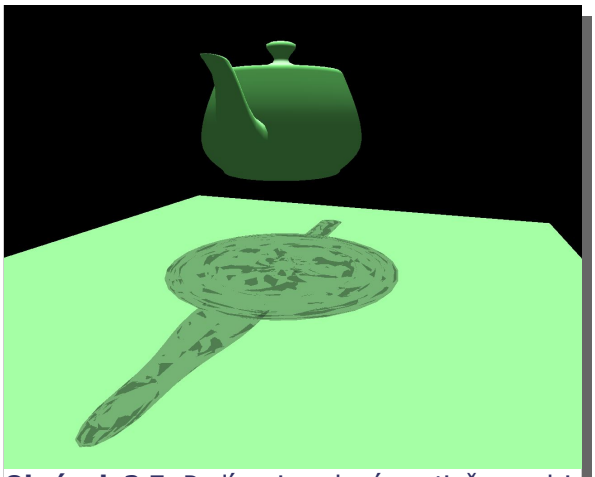
Ak sa celý objekt transformáciou premietne do jednej roviny, všetky jeho polygóny ležia v tejto rovine. Viacero polygónov sa prirodzene bude prekrývať, a keďže tieň vykresľujeme pomocou priesvitných šedých polygónov, na týchto miestach bude tieň tmavší. Tento jav sa nazýva dvojité prelínanie (*doble blending*).



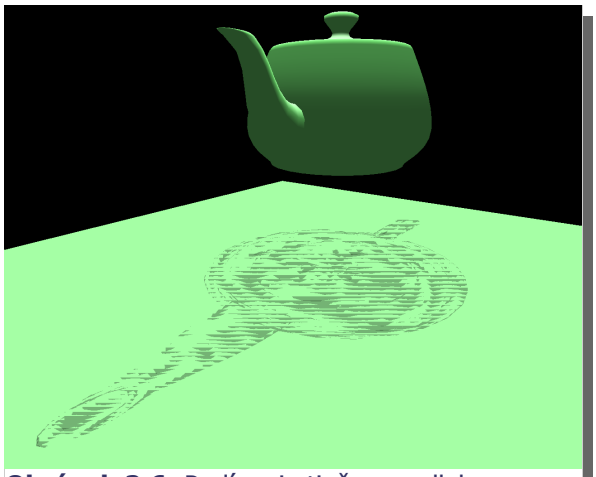
Obrázok 2.4: Nežiadúci efekt dvojitého prelínania (double blendingu)

Tieň sa vykresľuje s vypnutým hĺbkovým testovaním (*depth test*), jednotlivé tieňové polygóny sa vykreslia „na seba" podľa poradia vykresľovania. Na miestach, kde sa polygóny tieňu prekrývajú dochádza k zosilneniu farby tieňu. To je samozrejme nežiadúci efekt a nie je pri tejto metóde jediný. Riešením, ktoré prvé zíde na um, by mohlo byť zapnutie hĺbkového testovania. Ak problém spôsobuje vypnuté hĺbkové testovanie, tak ho odstránime jeho zapnutím (mohlo by sa zdať). Prekrývajúce sa časti sa pomocou hĺbkovej pamäti (*depth-buffer*, *z-buffer*) odstránia z vykresľovacieho procesu, pretože sa vykresľujú iba nezakryté časti polygónov. Bohužiaľ, s týmto riešením prichádzajú ďalšie problémy.

Druhým neduhom tejto techniky je prelínanie polygónov tieňa, ktoré spôsobuje nepresnosť hĺbkovej pamäti. Jednotlivé tieňové polygóny, ktoré sú navzájom tak blízko, že majú v hĺbkovej pamäti rovnaké hodnoty, spôsobujú vo vykresľovacom procese nejednoznačnosť. Problémom je, ktorý polygón má byť na obrazovku vykreslený skôr. V každom snímku sa teda môže poradie vykreslenia tieňových polygónov vyhodnotiť inak, čo spôsobuje spomínaný nežiadúci artefakt s preblikávaním. Okrem toho môže dôjsť k prelínaniu tieňa s podlahou.



Obrázok 2.5: Prelínanie polygónov tieňu medzi sebou



Obrázok 2.6: Prelínanie tieňu s podlahou

Spomínané problémy tejto techniky rovinných tieňov môžeme vyriešiť pomocou jednoduchých korekcií. Presnosť hĺbkovej pamäti nie je nekonečná a závisí od vzdialenosti. Musíme teda zabezpečiť aby do vhodnej vzdialenosti artefakt s prelínaním polygónov nevznikal, a to posunutím každého ďalšieho vykresľovaného tieňového polygónu o určitú (pokiaľ možno čo najmenšiu) hodnotu v hĺbkovej pamäti ďalej. Táto hodnota bude závisieť od vzdialenosti tieňu od pozorovateľa. Keďže presnosť pamäte hĺbky logaritmicky klesá so vzdialenosťou, bude hodnota posuvu tieňových polygónov tým väčšia, čím ďalej bude tieň vzdialený od pozorovateľa. To isté platí o prelínaní polygónov tieňu a podlahy. Stačí podlahu posunúť o malú hodnotu nižšie ako tieň a k problémom nebude dochádzať.

Ako vidieť s riešením jedného, resp. dvoch problémov sme zaviedli problém nový. Ak sa chceme úplne vyhnúť problémom s hĺbkovou pamäťou, bude najlepšie ho na riešenie artefaktov nepoužiť. Existuje ďalší spôsob ako si poradiť s prelínajúcimi sa tieňovými polygónmi, a tým je použitie pamäti šablóny (*stencil buffer*). Stencil buffer sa používa hlavne na šablónovanie, orezávanie a všeobecne na obmedzovanie priestoru pre vykresľovanie určitých oblastí scény. Presne sa teda hodí na naše riešenie. Prvým krokom bude vymazanie pamäti šablóny na hodnotu 0. Podlaha, alebo ľubovoľná rovina, na ktorú bude objekt vrhať tieň, sa vykreslí s hodnotou šablóny 1. Tieň sa bude vykresľovať iba na týchto miestach scény. To zároveň zaistí, že tieň bude krajmi roviny (podlahy) orezaný a nemôže sa stať, že sa vykreslí aj mimo nej (ako tomu bolo doteraz). Na tých miestach podlahy, kde sa vykreslí tieň (prejde *stencil testom*) zvýšime hodnotu šablóny na 2. Keďže sa tieň vykresľuje iba na miestach s hodnotou šablóny 1 a zatienené miesta majú hodnotu 2, nemôže sa stať, že sa tieňové polygóny budú prelínať.

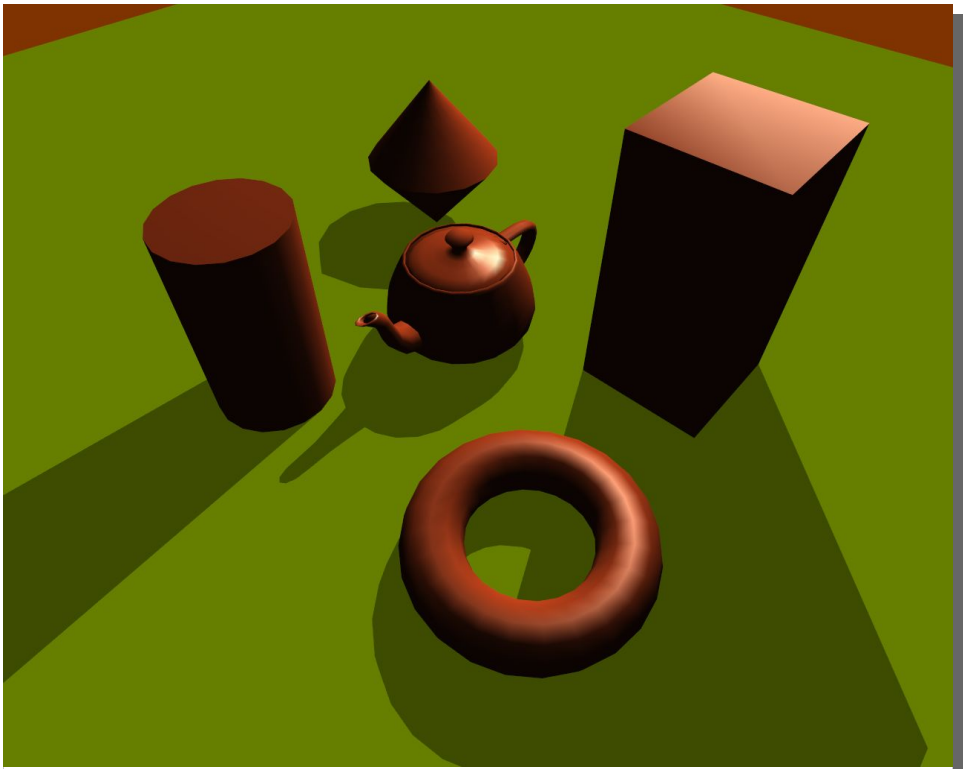
Algoritmus 2.1:

1. `Inicializuj pamäť šablóny na 0`
2. `Vykresli tieňovú rovinu s hodnotou šablóny 1`
3. `Vykresli tieňový polygón na miesta scény s hodnotou šablóny 1 a zároveň inkrementuj hodnotu šablóny na 2`
4. `Vykonaj krok 3 pre všetky tieňové polygóny`

Stále však môže dochádzať k prelínaniu sa polygónov tieňu s podlahou. Riešením je posun tieňu o malú hodnotu v smere osi z.

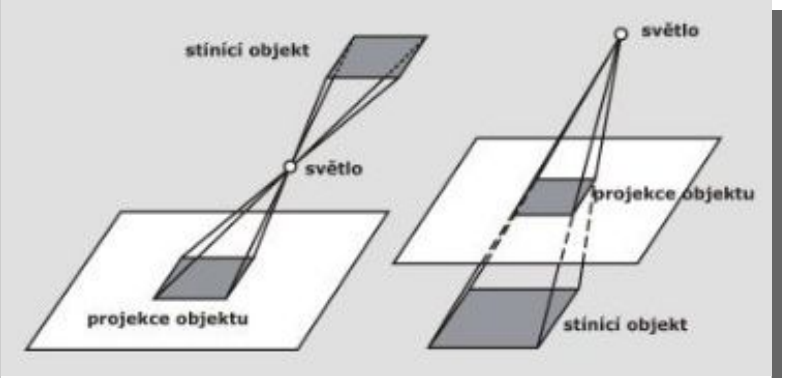
Obrázok 2.7: Korekcia so stencil bufferom

Dosiahli sme správne vykreslenie tieňu pod objektom. Nedostatkom tejto techniky je však fakt, že tieto tiene ležia iba v rovinách. Ak by sa napr. v zatienenej časti podlahy nachádzal iný objekt, nebol by tento objekt zatienený. Rovinné tiene totiž neinteragujú s inými objektmi. Nedokážu sa správne „namapovať" na polygóny objektu, pretože existujú iba v rovine.



Obrázok 2.8: Nezatienený objekt v zatienenej oblasti

Jedným z ďalších problémov tejto techniky je vrhanie **falošných tieňov** a **anti-tieňov**. Jadro problému spočíva v tom, že transformácia objektu na rovinu a následné vykreslenie tieňu sa vykoná za každých podmienok, a to aj v prípade, keby sa nemalo. V skutočnosti vrhá objekt tieň iba ak sa nachádza medzi svetelným zdrojom a plochou, na ktorú má vrhať tieň. V prípade tejto metódy sa však tieň na rovine vykreslí aj v prípade, že je objekt za svetelným zdrojom, teda svetlo sa nachádza medzi objektom a rovinou. V tomto prípade vznikajú tzv. anti-tiene. Druhým prípadom je, ak sa objekt nachádza pod rovinou, teda rovina sa nachádza medzi svetlom a objektom. Aj v tomto prípade, sa objekt transformuje na rovinu a vykreslí sa jeho tieň, ktorý nazývame nepravý alebo falošný.



Obrázok 2.9: Vznik antitieňov a nepravých tieňov [2]

Riešenie problému s nepravými tieňmi navrhli Michael Herf a Paul Heckbert [6]. Pozmenili stávajúcu transformáciu z 3D do 2D, ktorá objekt premietne do roviny, na transformáciu z 3D do 3D a to tak, že táto transformácia zobrazí telesá spôsobujúce nepravé tiene mimo pohľadového priestoru. Telesá mimo pohľadového priestoru potom nebudú vykreslené a teda ani nedôjde k vykresleniu ich nepravých tieňov.

2.6 Rozšírenia

Zatiaľ však vieme vrhať tiene objektov iba na jednu rovinu. Objekt, ktorý by sa nachádzal napr. v rohu miestnosti, by svoj tieň vrhal iba na podlahu. Rovinné tiene nemajú sami o sebe schopnosť ohybu na iné objekty, pretože ležia iba v jednej rovine, ktorá je stanovená transformačnou maticou tieňu. Ak chceme, aby sa tieň patrične „rozložil" do viacerých rovín (napr. do rohu miestnosti), musíme tieň vykresliť na každú rovinu zvlášť. To znamená, že pre každú rovinu sa musí uskutočniť transformácia objektu do tejto roviny a vykreslenie jeho tieňových polygónov. Aby teda objekt vrhal tieň správne aj v rohu miestnosti, uskutočníme transformáciu celkom 3-krát, raz pre podlahu a 2-krát pre priľahlé steny. Výsledok ukazuje obrázok 2.3.

Dosiahneme tak správneho rozloženia tieňu na viacerých pretínajúcich sa rovinách, stojí to však trojnásobok výpočetného času.

3 Tieňové mapy

Algoritmus výpočtu tieňových máp - *shadow mapping* - navrhol a úspešne realizoval Lance Williams už v roku 1978, vo svojom článku [7]. Jedná sa o jednoduchý, ale efektívny, a veľmi použiteľný spôsob zobrazovania tieňov v reálnom čase, ktorý sa výborne hodí najmä na komplexné objekty.

Obrázok 3.1: Tieňové mapy v počítačovej hre Call of Juarez Bound In Blood (2009, Ubisoft)

3.1 Princíp algoritmu

Hlavnou súčasťou algoritmu je využitie **z-buffera** (*depth buffer*), teda hĺbkovej pamäti obrazu. Hĺbková pamäť umožňuje zaznamenávať a uchovávať informácie o hĺbke objektov v scéne. Pomáha tým pri určovaní viditeľnosti jednotlivých objektov a udáva poradie v akom budú objekty vo výslednej vizualizácii vykreslené. Je reprezentovaný dvojrozmerným poľom o veľkosti ***x,y***, kde ***x,y*** sú rozmery obrazu. Obsahuje teda jednu hodnotu pre každý fragment (pixel) obrazu. S každým vykresleným objektom sa na aktuálny pixel uloží do hĺbkovej pamäti jeho vzdialenosť od kamery – hĺbka v obraze. Ak má byť na rovnaký pixel vykreslených viacero objektov (ide teda o prekrytie objektov), hodnoty hĺbky

týchto pixelov sa najprv porovnajú a vyberie sa hodnota bližšie k pozorovateľovi. Vybraná hodnota je potom zapísaná do hĺbkovej pamäti, prepisujúc hodnotu starú. Toto umožňuje, aby objekty bližšie k pozorovateľovi prekrývali objekty za nimi.

Využitie z-buffera v algoritme mapovania tieňov spočíva v porovnaní hodnôt hĺbky obrazu z pohľadu pozorovateľa a z pohľadu svetelného zdroja. Povedané v skratke, algoritmus zisťuje, či je daný pixel viditeľný z bodu svetelného zdroja, a to porovnaním hodnôt hĺbky obrazu (uložených v z-bufferi) z pohľadu pozorovateľa a obrazu vytvoreného pohľadom z bodu svetelného zdroja (uložený v textúre).

Algoritmus 3.1

1. Zobraz scénu z pohľadu svetelného zdroja. Hodnoty z-buffera ulož do hĺbkovej mapy.
2. Zobraz scénu z pohľadu kamery pomocou z-buffera
3. Pre všetky pixely [u, v] s hĺbkou w zobrazenej scény sprav nasledujúce:

- preveď súradnice pixelu [u, v, w] do sústavy súradníc zdroja svetla a získaj tak jeho nové súradnice [x, y, z]
- do premennej Z_A si ulož hodnoty hĺbkovej mapy pixelu na súradniciach [x, y]
- do premennej Z_B si ulož hodnoty z
- ak je $Z_A < Z_B$, je pixel na súradniciach [u, v] zatienený, inak je osvetlený

Ukážeme si ďalej ako jednotlivé kroky algoritmu konkrétne realizovať a ako sa týmto spôsobom dá dosiahnuť vykresľovanie ľubovoľne komplexných tieňov.

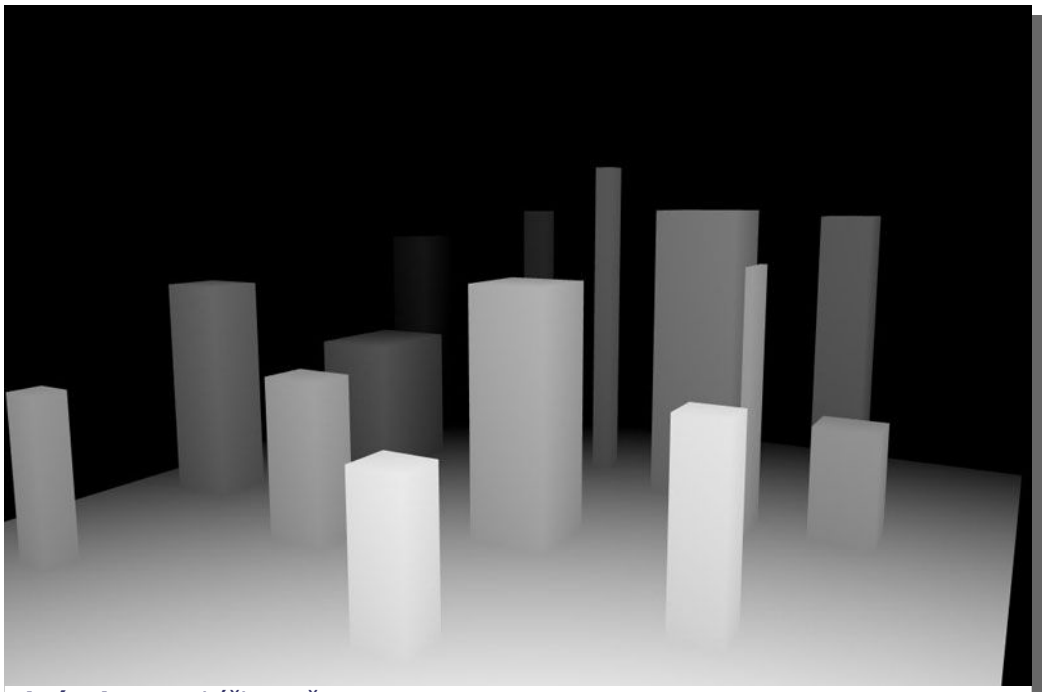
3.2 Tvorba tieňovej mapy

Prvým krokom algoritmu je vytvorenie tzv. **tieňovej mapy** (*shadow map*), resp. **mapy hĺbky** (*depth map*). Tieňovú mapu tvoria hodnoty hĺbkovej pamäti scény vykreslenej z pohľadu svetla. Je teda potrebné umiestniť kameru do bodu svetelného zdroja, nastaviť príslušnú projekciu podľa charakteru svetla (perspektívna projekcia pre bodové svetlo, ortogonálna pre smerové), a z tohto pohľadu vykresliť žiadaný objekt (alebo viacero objektov). Výsledok potom môžeme uložiť do špeciálnej textúry – hĺbkovej mapy, alebo (ak to hardware umožňuje) vykresľovať priamo do textúry. Keďže nám ide iba o hodnoty hĺbky pixelov obrazu, pri vykresľovaní môžeme vynechať výpočet osvetlenia, informácie o textúre, či zápis do pamäti farby, čím ušetríme nemalú časť vykresľovacieho času. Tým sa zároveň stáva produkcia tieňovej mapy menej výpočetne náročná ako zvyčajné vykreslenie objektu.

Dôležitá je veľkosť tieňovej mapy, teda rozlíšenie obrazu, ktorý vykresľujeme z pohľadu svetla. Vo veľkej miere ovplyvňuje nielen výslednú kvalitu tieňov, ale aj celkový čas výpočtu algoritmu, teda výpočetnú náročnosť.

Všetky objekty, ktoré majú vrhať tieň, sa musia nachádzať vnútri pohľadového telesa svetla, tj. pohľadového telesa kamery umiestnenej do bodu svetelného zdroja. Tvar a veľkosť tohto pohľadového telesa ovplyvňuje viacero faktorov. Ide hlavne o typ projekcie a umiestnenie orezávacích rovín (near plane, far plane). V prípade perspektívnej projekcie je dôležitým faktorom field-of-view, teda rozsah uhlu pohľadu. Ak sa objekt nachádza mimo pohľadového telesa svetla, nebude objekt zahrnutý do tieňovej mapy a jeho tieň nemôže

byť vytvorený. Neexistujú záznamy o hĺbke pixelov, ktoré objekt tvoria, a teda nemáme informáciu, s ktorou by sme mohli porovnávať. Tak isto, ak sa v hĺbkovej mape nachádza iba časť objektu, dokážeme vytvoriť tieň iba pre časť obsiahnutú v hĺbkovej mape. Veľkosť a tvar pohľadového telesa svetla však priamo ovplyvňuje kvalitu zobrazovaných tieňov a preto je dôležité nastaviť pohľadové teleso čo najoptimálnejšie. Veľkú rolu pritom zohrávajú blízka a vzdialená orezávacia rovina, ktoré vo veľkej miere ovplyvňujú presnosť z-buffera v jednotlivých častiach scény. Viacej o tejto problematike v časti 3.4.



Obrázok 3.2: Ukážka tieňovej mapy

Pokiaľ ide o to, ako často musíme vytváranie hĺbkovej mapy realizovať, tak je to práve vtedy, ak sa hodnoty hĺbky objektu v scéne vzhľadom na svetelný zdroj môžu meniť. Jednou z takéhoto typu situácií je zmena polohy bodu svetelného zdroja. Druhou situáciou je potom zmena polohy samotného objektu vrhajúceho tieň. V ostatných prípadoch, napr. pri prostej zmene pohľadu kamery, môžeme využiť už raz vyprodukovanú hĺbkovú mapu, čím môžeme ďalej ušetriť čas celkového vykresľovania. Nutno ešte podotknúť, že v prípade viacerých svetelných zdrojov je potreba zhotoviť hĺbkovú mapu pre každý svetelný zdroj zvlášť.

Pozn.: V tejto práci budeme termín „tieňová mapa" často zamieňať za pojem „hĺbková mapa", ak je potreba zdôrazniť, že mapa obsahuje informácie o hĺbke objektov, myslíme tým ale jednu a tú istú vec. Niekedy sa používa aj termín „tieňová mapa hĺbky", ktorý je taktiež významovo totožný s predchádzajúcimi pojmami.

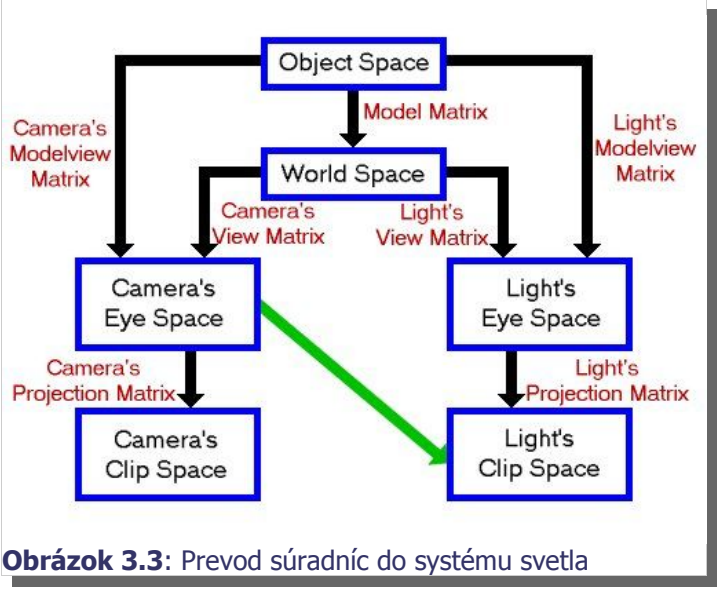
3.3 Zobrazenie tieňov v scéne

Druhý krok algoritmu výpočtu tieňových máp je obvyklé zobrazenie scény z pohľadu kamery. Pri tomto zobrazení už uplatňujeme všetky žiadané funkcie zobrazovacieho systému ako výpočet osvetlenia, textúrovanie a pod. Zároveň aplikujeme na scénu tieňovú mapu, čo je posledným krokom algoritmu. Aplikácia tieňovej mapy však nie je úplne triviálnou záležitosťou, pozostáva ďalej z viacerých krokov, ktoré si teraz rozpíšeme.

3.3.1 Prevod súradníc do priestoru svetla

Z 3. kroku algoritmu 3.1 môžeme vidieť, že pôjde o porovnávanie hodnôt hĺbky obrazu videného z pohľadu svetelného zdroja a obrazu videného kamerou. Na základe tohto porovnania sa potom určí, ktoré miesta v scéne budú zatienené, a ktoré osvetlené. Aby sme však mohli takéto dva obrazy, resp. ich hĺbkovú pamäť porovnávať, musia byť hodnoty hĺbky získané z ekvivalentných miest (bodov, fragmentov, pixelov) v scéne. Ak zoberieme súradnice pixelu v obraze z pohľadu svetla (hĺbková mapa), daný pixel určite nebude zodpovedať pixelu na obraze vytvorenom kamerou. Preto je treba tieto súradnice zjednotiť do spoločného súradnicového systému tak, aby bod v scéne zobrazenej kamerou zodpovedal tomu istému bodu videnému z pohľadu svetla. Dosiahneme to prevedením súradníc obrazu kamery do súradnicového systému svetelného zdroja.

Prevod súradníc z jedného súradnicového systému do druhého sa zvyčajne realizuje maticovým násobením. Ešte pred tým, ako začneme násobiť, mali by sme si vyjasniť terminológiu ohľadne použitých súradnicových systémov. Body v obraze vytvorenom kamerou sa nachádzajú v systéme, ktorý sa nazýva **svetový súradnicový systém** (*world coordinate system*), budeme ich teda volať **svetové súradnice**. Body v obraze vytvorenom z pohľadu zo svetelného zdroja sa nachádzajú v súradnicovom systéme svetla, alebo v **priestore svetla** (*light space*), a zvyčajne sa nazývajú **súradnice v priestore svetla** (*light space coordinates*). Aby sme zo svetových súradníc dostali súradnice v priestore svetla, potrebujeme ich transformovať rovnakými maticami, ktoré sme použili pri nastavení pohľadového telesa svetla a jeho projekcie (či už sa jedná o perspektívu alebo ortogonálnu projekciu). Jedna z matíc sa nazýva **pohľadová matica svetla** (*light view matrix*) a druhá **projekčná matica svetla** (*light projection matrix*). Pohľadovú maticu svetla získame uložením transformačnej matice po nastavení pohľadového telesa svetelného zdroja (v OpenGL po zavolaní funckie `gluLookAt` alebo `glFrustum`). Projekčnú maticu svetla potom získame uložením transformačnej matice po nastavení projekcie pohľadu svetla (v OpenGL po zavolaní funckie `gluPerspective`).



Obrázok 3.3: Prevod súradníc do systému svetla

Vynásobením svetových súradníc týmito dvoma maticami v poradí projekčná matica násobená pohľadovou, získame homogénne súradnice, ktoré sa po perspektívnej korekcii stávajú normalizovanými súradnicami, čo znamená že ich hodnoty ***x***,***y*** a ***z*** spadajú do intervalu -1 až 1. Keďže informácie o hĺbke bodov v scéne zobrazene z pohľadu svetelného

zdroja sú uložené vo forme hĺbkovej mapy, a s tou sa obvykle pracuje ako s textúrou, viacej by nám vyhovovalo, keby transformované súradnice do priestoru svetla spadali do intervalu 0 až 1. Dôvodom je systém procesu texturovania, ktorý používa pre namapovanie textúry na objekt súradnice práve z intervalu 0 až 1. Previesť súradnice z oboru hodnôt -1 až 1 do intervalu 0 až 1 však nie je žiadny problém. Postačí na to jednoduchá transformácia maticou v tvare:

$$\begin{bmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ktorú nazývame **maticou posunu** (*bias matrix*). Matica posunu zastáva v poradí transformácie svetových súradníc na súradnice v priestore svetla prvé miesto, celá transformácia potom vyzerá nasledovne:

$$\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & & & \frac{1}{2} \\ & \frac{1}{2} & & \frac{1}{2} \\ & & \frac{1}{2} & \frac{1}{2} \\ & & & 1 \end{bmatrix} \begin{bmatrix} Light \\ Frustum \\ (projection) \\ Matrix \end{bmatrix} \begin{bmatrix} Light \\ View \\ (look\ at) \\ Matrix \end{bmatrix} \begin{bmatrix} Modeling \\ Matrix \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{bmatrix}$$

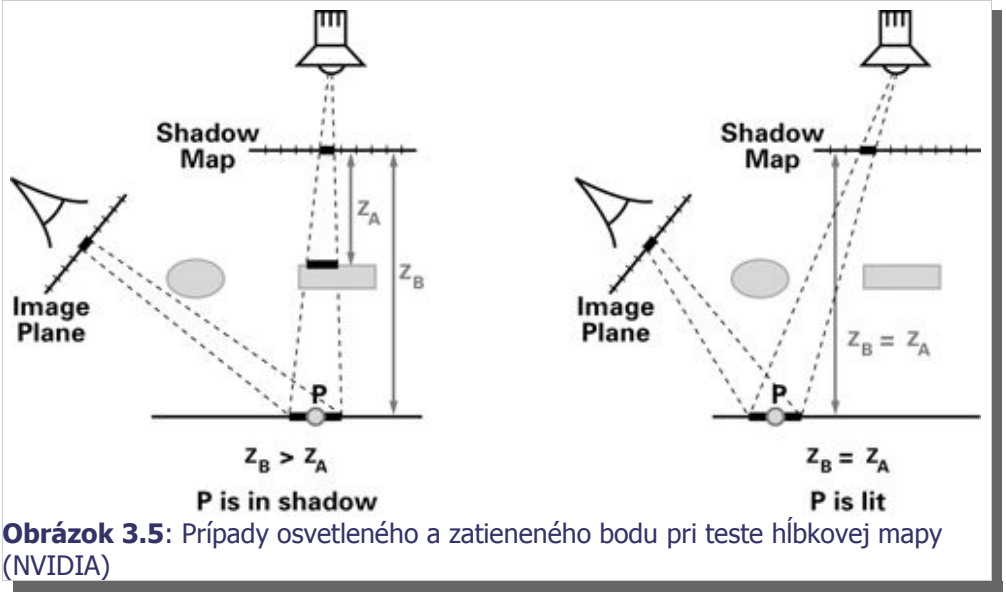
Obrázok 3.4: Transformácia prevodu súradníc do sústavy svetla

Matica posunu sa vynásobí projekčnou maticou svetla a výsledok sa nakoniec násobí pohľadovou maticou svetla.

3.3.2 Test hĺbkovej mapy

Teraz, keď máme zjednotené súradnice do jedného súradnicového systému (súradnicový systém svetla), môžeme začať porovnávať jednotlivé hodnoty hĺbky obrazu produkovaného kamerou s hodnotami v tieňovej (hĺbkovej) mape. V praxi sa to realizuje tak, že sa tieňová mapa, v podobe textúry, namapuje pomocou vyššie spomínaných transformovaných súradníc na scénu vykreslenú z pohľadu kamery. Pri mapovaní tieňovej mapy sa využívajú tie isté princípy ako pri projekčnom mapovaní textúry. Metóda projekčného mapovania je spôsob, pri ktorom sa na scénu dokáže aplikovať obrázok textúry ako keby bol premietaný z projektora. V našom prípade to znamená, že tieňová mapa sa namapuje na scénu ako keby bola premietaná z bodu svetelného zdroja.

Body v scéne sú teda presne „pokryté" zodpovedajúcimi bodmi v tieňovej mape. Môžeme teda priamo porovnávať hodnoty hĺbky týchto bodov, čo znamená, že vlastne porovnávame hodnoty vzdialeností od bodu svetelného zdroja jednotlivých objektov, resp. bodov, ktoré ich tvoria. Na jednej strane máme vzdialenosť od svetelného zdroja v tieňovej mape, teda scény videnej z pohľadu svetla. Na strane druhej máme vzdialenosť od svetelného zdroja v scéne zobrazovanej aktuálnou kamerou. Ak má bod v tieňovej (hĺbkovej) mape nižšiu vzdialenosť od svetelného zdroja ako v aktuálne zobrazovanej scéne, znamená to, že medzi ním a bodom svetelného zdroja sa musí nachádzať objekt, ktorý „stojí v ceste" lúču svetla dopadajúceho na daný bod scény, pretože vzdialenosť tohto objektu od svetla je nižšia. Ako je to uvedené v algoritme 3.1 $Z_A < Z_B$, inak povedané, daný bod je tým pádom zatienený.



Obrázok 3.5: Prípady osvetleného a zatieneného bodu pri teste hĺbkovej mapy (NVIDIA)

Proces porovnávania hĺbky bodov scény s bodmi v hĺbkovej mape sa nazýva **test hĺbkovej mapy** (*depth map test*). Ak je po prevedení testu bod v scéne zatienený, hovoríme, že test zlyhal. Ak naopak test uspeje, znamená to, že daný bod je osvetlený. Postupným testovaním celého rozsahu hĺbkovej mapy sa v scéne vytvoria tiene objektov, ktoré boli v hĺbkovej mape zaznamenané. Pre miesta v scéne, ktoré nie sú pokryté hĺbkovou mapou nemá zmysel test prevádzať, tieto miesta sú teda zobrazené s plným osvetlením.

3.3.3 Vrhanie tieňov

Praktická realizácia testu hĺbkovej mapy môže mať viacero podôb. Vďaka veľkému rozšíreniu použitia tejto techniky zobrazovania tieňov v real-time grafike sa stretávame na dnešných grafických čipoch s veľmi dobrou hardwarovou podporou. Niektoré kroky algoritmu sú pre nás teda značne zjednodušené a dejú sa automaticky.

Ak uvažujeme vykresľovací proces s využitím programovateľného zobrazovacieho reťazca (*shaderu*), test má podobu jednoduchého získania hodnoty z textúry (hĺbkovej mapy), pretože sa pri tomto úkone deje porovnávanie hodnôt hĺbky automaticky. Procedúra, ktorá v prípade normálnej obrazovej textúry vracia farbu textúry na daných súradniciach, vracia v prípade tieňovej mapy výsledok testu hĺbkovej mapy. Konkrétne to znamená, že ak leží bod scény v zatienenej oblasti, vráti 0, ak naopak leží v osvetlenej oblasti, vráti 1. Tento fakt sa dá jednoducho využiť napr. na spriemerovanie výsledku získaného z určitého okolia záujmového bodu, ktoré si neskôr popíšeme v časti 3.5.2.

Aj bez využitia programovateľného zobrazovacieho reťazca máme možnosť si test hĺbkovej mapy zjednodušiť. V prostredí OpenGL je to napríklad rozšírením `GL_ARB_shadow`. Ale zatiaľčo s využitím shaderu môže byť celá procedúra zobrazenia scény s tieňmi dvojpriechodová záležitosť (prvým priechodom je zobrazenie scény z pohľadu svetelného zdroja, druhý priechod predstavuje samotné vykreslenie scény s realizovaním testu hĺbkovej mapy), s použitím fixnej funkčnosti zobrazovacieho reťazca pozostáva realizácia algoritmu z troch priechodov.

Algoritmus 3.2

1. `Zobrazenie scény z pohľadu svetelného zdroja`
2. `Vykreslenie celej scény v tieni.`
3. `Zapnutie testovania hĺbkovej mapy a vykreslenie osvetlenej scény.`

Prvý krok algoritmu 3.2 by už mal byť zrejmý a bol dostatočne popísaný v predchádzajúcej časti. Druhým krokom je vykreslenie celej scény v tieni, to znamená vykreslenie scény s osvetlovacím modelom, ktorý chceme použiť pre zatienené miesta v scéne. Obvykle to býva osvetlovací model pozostávajúci iba z osvetlenia okolia (*ambient lighting*) a bez odrazovej (*specular*) zložky svetla, pretože v zatienených oblastiach nemajú odlesky na predmetoch zmysel. Občas sa ešte k tomuto modelu pridáva určitá difúzna zložka svetla, zaisťujúca, aby tiene nevyzerali príliš plocho a aby aj na zatienených miestach scény bolo viditeľné zakrivenie povrchov. Presnejšie vyjadrenie tohto osvetlovacieho modelu poskytuje [8]:

> *"Požadovaná farba osvetleného, otexturovaného tieňového fragmentu sa vypočíta podla vzťahu:*
>
> `(ambient+diffuseShade*diffuse)*decal+specular*shade`
>
> *kde* `diffuseShade=dimming+(1.0-dimming)*shade` *a* `shade` *je výsledok testu hĺbkovej mapy v rozsahu [0,1].* `dimming` *vyjadruje rozptýlené svetlo v scéne."*

Obrázok 3.6: Vplyv difúznej zložky na realistickosť tieňov (Naigovzin, [14])

Tretím a posledným krokom je uplatnenie testu hĺbkovej mapy. Pri zapnutom testovaní sa vykresľuje iba na miesta scény, ktoré týmto testom prešli, to znamená fragmenty, ktoré nie sú v zatienenej oblasti. V tomto kroku sa používa obvyklý osvetlovací model so všetkými zložkami svetla. Fragmenty obrazu, ktoré nesplnia test hĺbkovej mapy teda ostávajú v stave v akom boli po prevedení kroku 2 algoritmu 3.2, teda zatienené.

Na obrázku 3.7 vidíme scénu po prevedení všetkých krokov algoritmu tieňových máp. Objekty vrhajú do scény tiene správne zakrivené podľa tvaru objektov, na ktoré dopadajú.

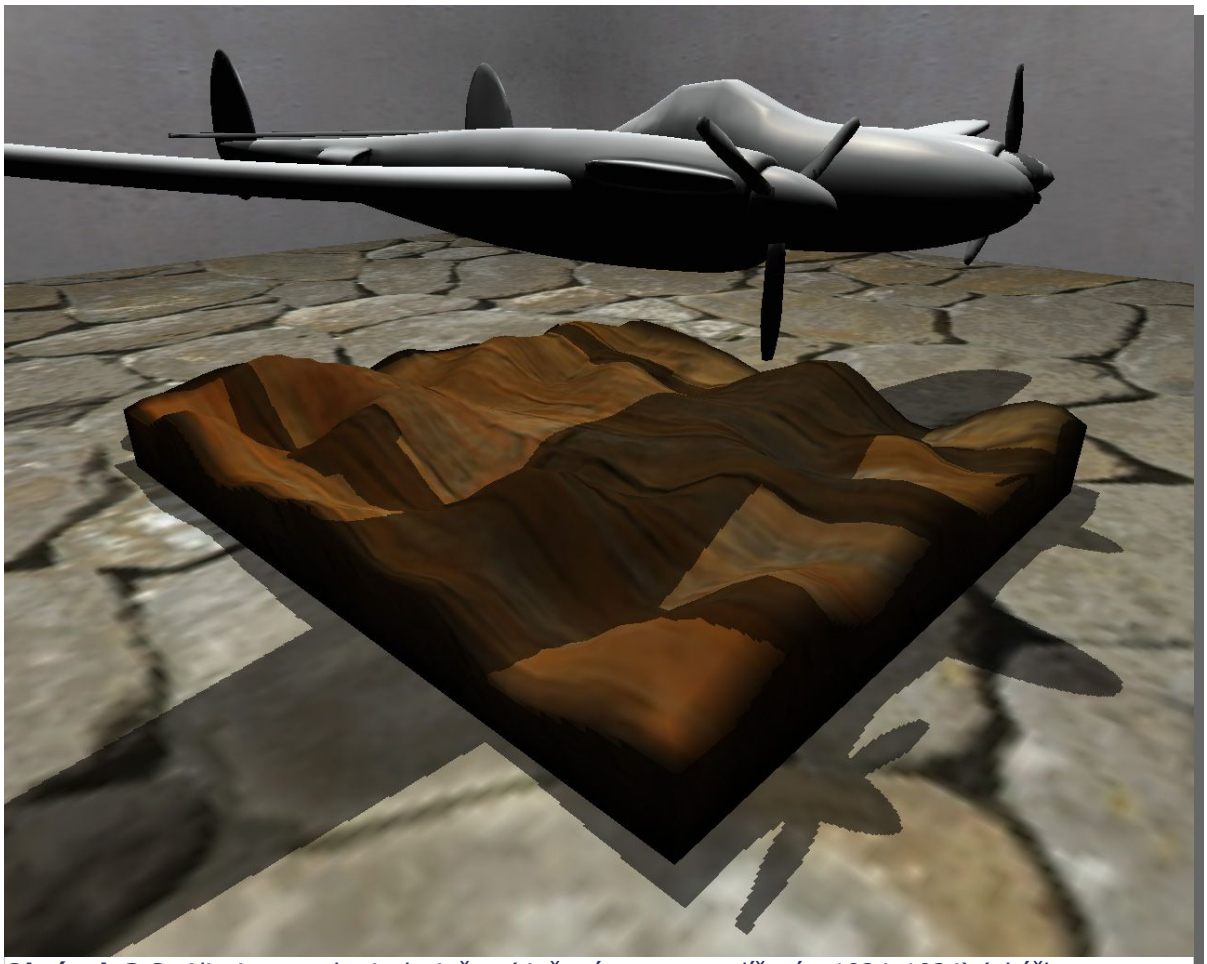
Zároveň táto technika umožňuje vytvárať aj tiene vlastné, teda objekty zatieňujú aj samé seba. Na rozdiel od väčšiny iných algoritmov na zobrazovanie tieňov v real-time grafike, nie sú tiene produkované algoritmom tieňových máp výsledkom žiadnych polygonálnych operácií. Algoritmus totiž pracuje v tzv. **obrazovom priestore** (*image space*). Nepracuje sa s geometriou objektu, ale s obrazom vytvoreným z pohľadu svetelného zdroja. Vďaka tomu nie je náročnosť tohto algoritmu zďaleka tak závislá na geometrickej zložitosti objektov (počet polygónov) ako iné metódy. Preto je vhodná práve pre scény s veľkým množstvom polygónov. Zvyšovanie počtu polygónov v scéne sa prejavuje iba na predĺžení vykresľovacieho času pri tvorbe tieňovej (hĺbkovej) mapy, rovnako ako pri každom inom vykresľovaní. Samotný proces vytvárania tieňov to však neovplyvňuje, pretože tieňová mapa je už iba súbor hodnôt hĺbky obrazu. Nezáleží teda, či v tomto obraze budú objekty so 100 polygónmi alebo s 100 000 polygónmi.

Obrázok 3.7: Technika tieňových máp využitá v hre Far Cry 2 (2008, Ubisoft)

3.4 Problémy

3.4.1 Aliasing na okrajoch tieňov

Skutočnosť, že algoritmus tieňových máp pracuje v obrazovom priestore neprináša iba výhody. Prináša to so sebou zároveň aj najväčšiu nevýhodu tejto techniky, ktorou je aliasing na okrajoch takto vzniknutých tieňov.



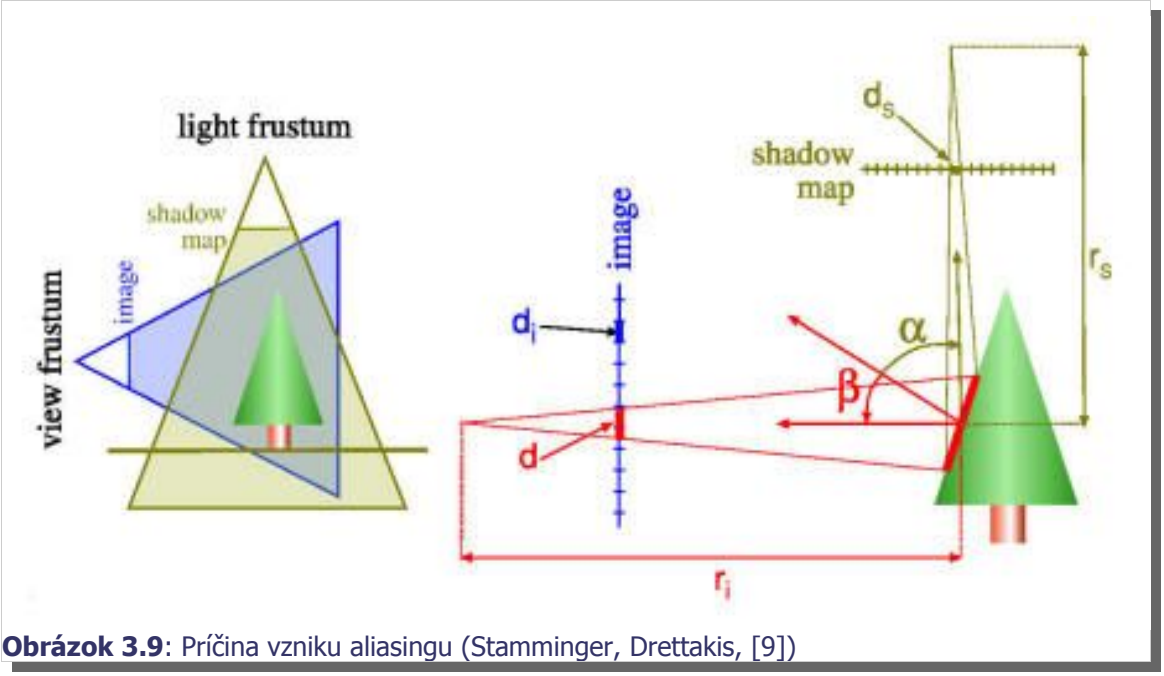
Obrázok 3.8: Aliasing na okrajoch tieňov (tieňová mapa s rozlíšením 1024x1024) (ukážka z demonstračnej aplikácie)

Na prvý pohľad je jasné, že aliasing má veľký vplyv na kvalitu zobrazovaných tieňov. Prejavuje sa nekontinuálnymi, hranatými okrajmi tieňov, a má svoj pôvod v diskrétnom vzorkovaní, ktoré sa uplatňuje pri ukladaní informácií o hĺbke fragmentov obrazu do hĺbkovej mapy. Hĺbková mapa má určité rozlíšenie (veľkosť), ktoré je vo väčšine prípadov menšie ako rozlíšenie výsledného obrazu. Pojme teda informácie o konečnom počte fragmentov, preto hovoríme, že sa informácia o hĺbke vzorkuje diskrétne. Pri realizácii testu hĺbkovej mapy sa porovná fragment ku fragmentu, avšak veľkosť fragmentu v scéne nemusí zodpovedať veľkosti fragmentu v hĺbkovej mape (ak by boli oba fragmenty rovnako veľké jednalo by sa o ideálny prípad a aliasing by nevznikal). Práve to, že väčší fragment v hĺbkovej mape zodpovedá menšiemu fragmentu v scéne spôsobuje, že sa väčší fragment namapuje (pokryje) viacero menších fragmentov v scéne. Dobrý formálny popis tohto problému poskytuje [9]:

> ***„Každý pixel veľkosti ds * ds v hĺbkovej mape sa mapuje na oblasť pixelov veľkosti (približne) d vo výslednom obraze:***

$$d = ds \frac{r_s}{r_i} \frac{\cos(\alpha)}{\cos(\beta)}$$

Vzorec 3.1:



Obrázok 3.9: Príčina vzniku aliasingu (Stamminger, Drettakis, [9])

*Podvzorkovanie nastáva, ak je d väčšie ako veľkosť pixelu výsledného obrazu d_i. Toto nastáva ak sa ds * (rs / ri) stane príliš veľkým, čo sa prejavuje ak sa pozorovateľ priblíži blízko okraju tieňa natoľko, že sú viditeľné jednotlivé pixely tieňovej mapy. Budeme to nazývať* ***perspektívny aliasing.***

Ak sa podiel stane cos(a) / cos(B) príliš veľkým, nastáva ***projekčný aliasing.*** *Toto sa zvyčajne stáva, keď sú lúče svetla takmer paralelné s povrchom, takže sa tieň natiahne pozdĺž povrchu.“*

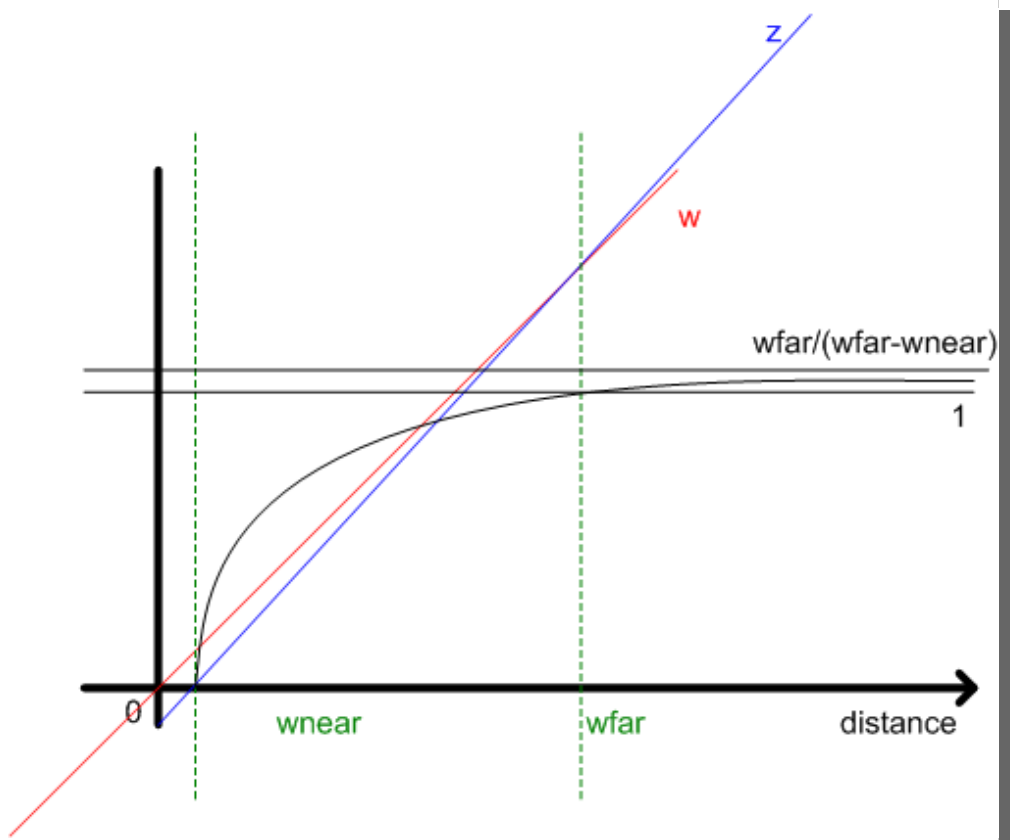
Viac o perspektívnom aliasingu a ako ho minimalizovať si ukážeme v časti 3.5.1 o perspektívnych tieňových mapách.

Problém aliasingu sa dá jednoducho vyriešiť (alebo aspoň zmierniť) zvyšovaním rozlíšenia tieňovej mapy. To so sebou ale prináša spomalenie celého algoritmu a zvýšenú pamäťovú náročnosť. Je teda dobré využiť všetkých dostupných riešení na potlačenie aliasingu aj bez zvyšovania rozlíšenia tieňovej mapy. Ak aj tak nie je problém vyriešený, môžeme rozlíšenie zvyšovať až na úroveň, ktorá je pre našu konkrétnu potrebu únosná a dostatočne kvalitná. Záleží pritom hlavne na type aplikácie - ak sa jedná o herný engine, určite budeme chcieť, aby sme spotrebovali pokiaľ možno čo najmenej prostriedkov a výpočetného času, ale napr. v prehliadači modelov si môžeme dovoliť aj veľmi vysoké rozlíšenie tieňových máp.

3.4.2 Aliasing vlastného zatienenia

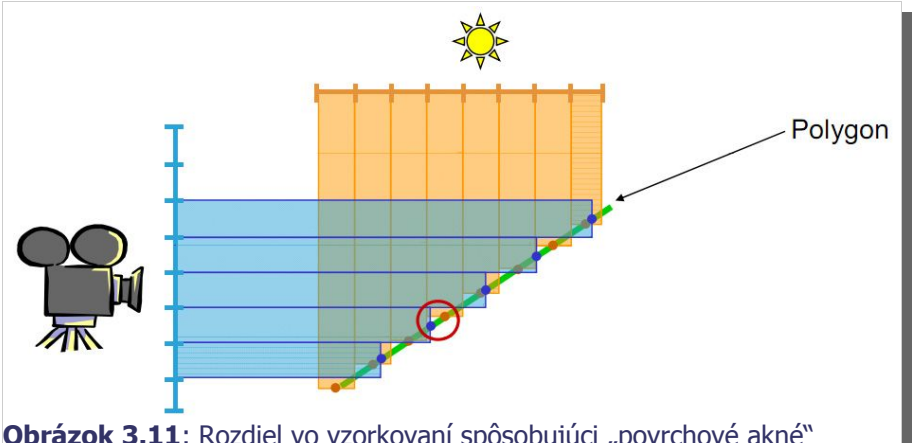
Ďalším problémom techniky tieňových máp je obmedzená presnosť hĺbkovej pamäti. Hĺbka fragmentu obrazu, tak ako ostatné súradnice, sa po perspektívnej korekcii mapuje do intervalu -1 až 1. Toto mapovanie však nie je uniformné, hodnoty v prvej polovici intervalu sú vzorkované hustejšie ako hodnoty v druhej polovici. Spôsobuje to zvýšenú presnosť hĺbkovej pamäti pre objekty pri bližšej orezávacej rovine a výrazne nižšiu presnosť od polovice vzdialenosti blízkej a vzdialenej orezávacej roviny. Pre zachovanie čo najlepšej presnosti teda

posúvame blízku orezávaciu rovinu, čo najďalej od oka pozorovateľa a čo najbližšie k pozorovanému objektu. Tak ako to platí pre bežné zobrazovanie scény, platí to aj pre tvorbu tieňovej mapy, kde je presnosť hĺbkovej pamäti dôležitým faktorom pri predchádzaní problémom.

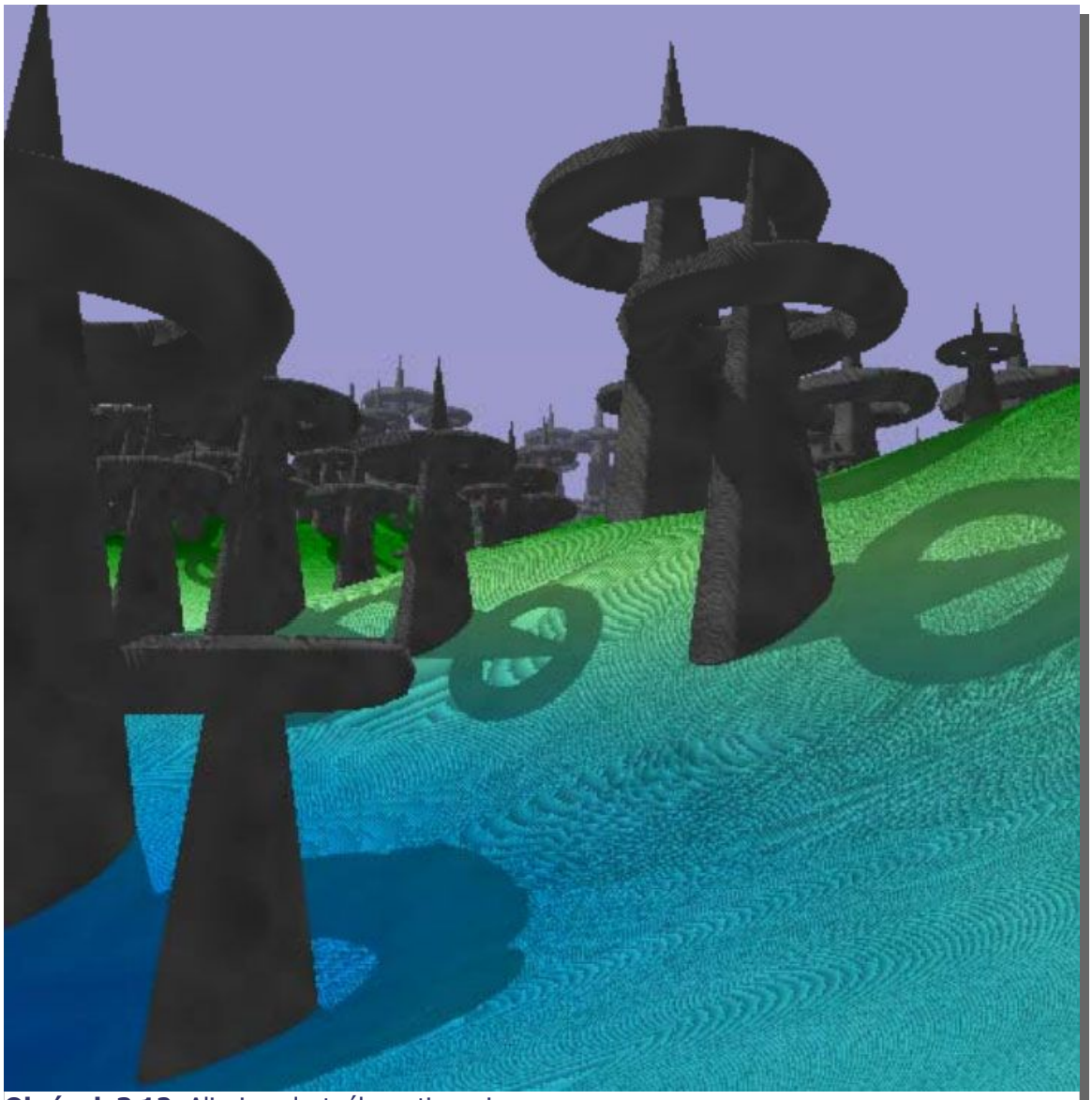


Obrázok 3.10: Distribúcia z-buffera v závislosti na polohách orezávacích rovín (gpwiki.org)

Nepresnosť pamäti hĺbky totiž spôsobuje ďalší typ aliasingu – **aliasing vlastného zatienenia** (*self shadow aliasing*, „povrchové akné" - *surface acne*). Pri porovnávaní hodnôt hĺbky fragmentov scény s bodmi v tieňovej mape sa vďaka nepresnosti hĺbkovej pamäti môže stať, že neporovnávame dva totožné body, ale body tesne vedľa seba, posunuté o určitú malú hodnotu hĺbky z. Tým pádom môže nastať situácia, že niektoré body budú zatieňovať samy seba (odtiaľ názov aliasing vlastného zatienenia), aj keď by mali byť osvetlené.



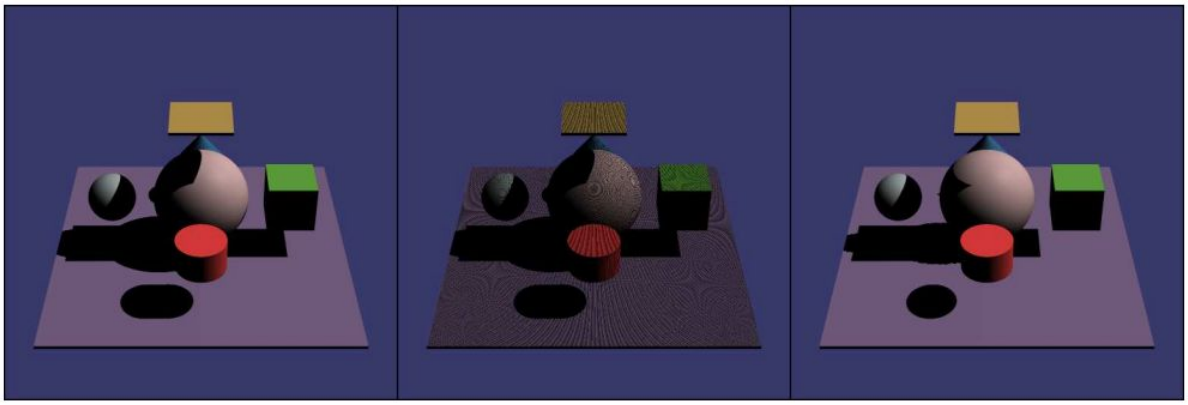
Obrázok 3.11: Rozdiel vo vzorkovaní spôsobujúci „povrchové akné"



Obrázok 3.12: Aliasing vlastného zatienenia

Aj tento typ aliasingu sa však dá vo veľkej miere obmedziť, riešenie však nie je až také priamočiare. Existujú najmenej dva postupy, pričom obidva sa môžu kombinovať. Jedným z nich je použitie maskovania predných plôch polygónov (*front face culling*) pri vykresľovaní scény do tieňovej mapy. Týmto sa scéna vykreslí s použitím odvrátených plôch polygónov, ktoré majú vždy o niečo menšiu hĺbku ako predné plochy. Pri vykresľovaní scény z pohľadu kamery sa potom použije opačné maskovanie odvrátených plôch polygónov (*back face culling*). Pri porovnávaní hĺbok, teda pri teste hĺbkovej mapy, bude potom predná plocha každého polygónu o určitú malú hodnotu **z** bližšie ako záznam o hĺbke odvrátenej plochy toho istého polygónu. Tým pádom polygón nezatieni sám seba.

V niektorých prípadoch však ani rozdiel v hĺbke medzi prednou a zadnou plochou polygónu nestačí na to, aby sa alias potlačil. Druhou možnosťou je potom použiť manuálne posunutie polygónu v smere **z** funkciou typu `glPolygonOffset` v OpenGL. Nájsť ale správnu hodnotu, o ktorú sa polygóny pri tvorbe tieňovej mapy majú posunúť nemusí byť triviálne.



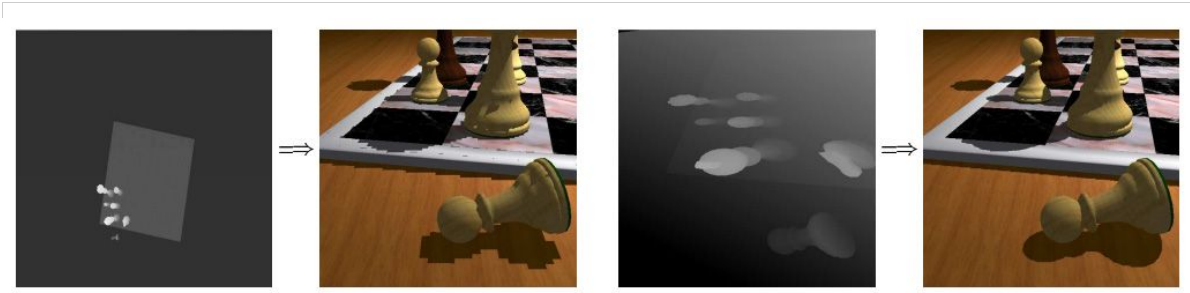
Obrázok 3.13: Správne nastavený posun (vľavo), príliš malý posun (stred), príliš veľký posun (vpravo) (Naigovzin, [14])

3.5 Rozšírenia

3.5.1 Perspektívne tieňové mapy

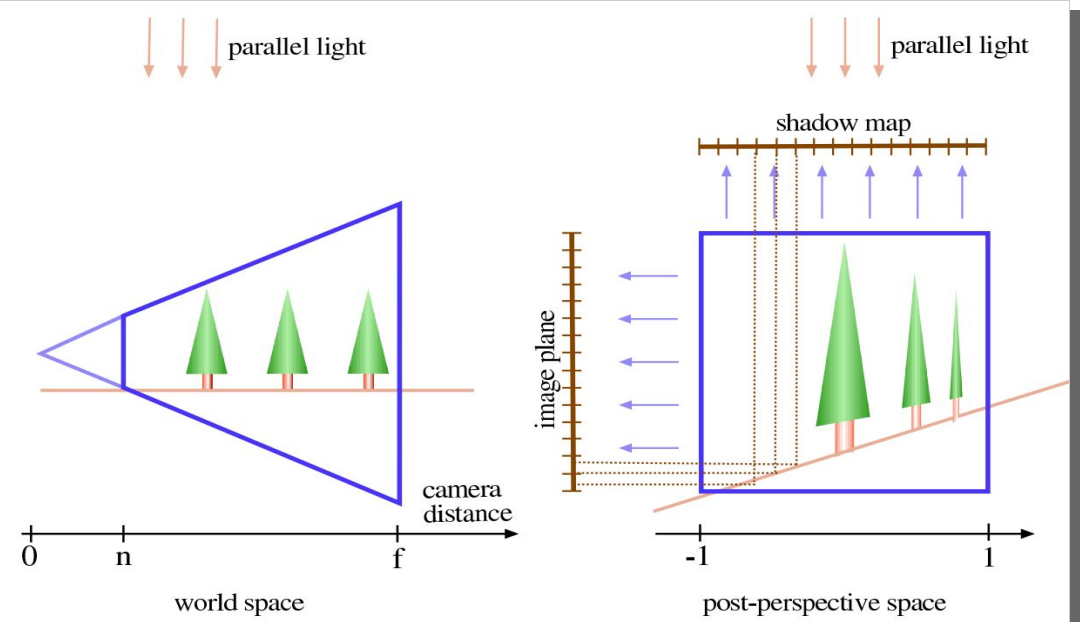
Nápad vytvárať tieňové mapy až po perspektívnej korekcii kamery (*Perspective shadow maps*, ďalej PSM), teda v normalizovanom priestore, je jedným z rozšírení pôvodného algoritmu tieňových máp, ktoré rieši problémy s aliasingom na okraji tieňov. Algoritmus bol prezentovaný na konferencii SIGGRAPH v roku 2002 pánmi Stammingerom a Drettakisom [9].

Princíp pôvodného algoritmu ostáva nezmenený, jedinou úpravou prešlo vytváranie tieňovej mapy. Kým v pôvodnom algoritme bola tieňová mapa vytváraná pohľadom zo svetelného zdroja s perspektívnou projekciou z bodu svetla, pri PSM sa toto dopĺňa ešte o jednu transformáciu, ktorou je perspektívna projekcia kamery. Pre tieňovú mapu to znamená, že objekty bližšie ku kamere budú v tieňovej mape zväčšené a objekty ďalej od kamery zmenšené. Tým pádom, sa zaistí, že miesta scény bližšie ku kamere, teda miesta najcitlivejšie na odhalenie vizuálnych nepresností užívateľom, dosiahnu väčšiu hustotu vzorkovania hĺbky obrazu a teda vyššiu kvalitu samotných tieňov.



Obrázok 3.14: Štandartné tieňové mapy (hore), perspektívne tieňové mapy (dolu), (Stamminger, Drettakis, [9])

Vyjadrené formálne ide elimináciiu perspektívneho aliasingu znižovaním hodnoty výrazu *ds * (rs / ri)* vo vzorci 3.1. Ideálny prípad pre využitie PSM nastáva ak je smer paralelného svetla kolmý na smer pohľadu pozorovateľa (kamery). V tomto prípade majú všetky body v tieňovej mape rovnakú veľkosť ako body výsledného obrazu a perspektívny aliasing je teda úplne potlačený.

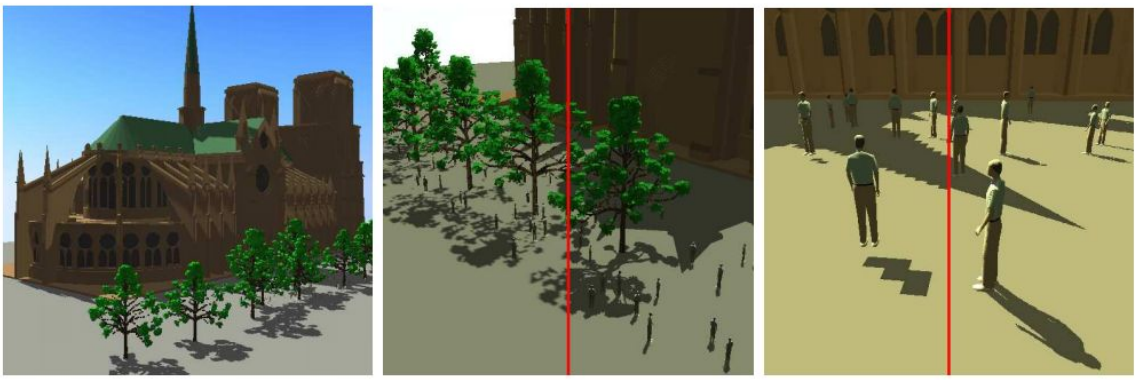


Obrázok 3.15: Transformovanie scény do normalizovaného priestoru (Stamminger, Drettakis, [9])

S čím je treba pri použití PSM počítať je, že transformovanie scény do normalizovaného priestoru môže ovplyvniť charakter svetiel v scéne. Všetky smerové svetlá, ktoré nie sú paralelné s rovinou obrazu sa menia na bodové a všetky bodové svetlá kolmé na rovinu obrazu sa stávajú naopak smerovými. S tým súvisí vhodnosť použitia tohto algoritmu. Najlepšie výsledky dosahuje pri smerových svetlách kolmých na rovinu obrazu a bodových svetlách ležiacich v rovine paralelnej s rovinou kamery (typickým príkladom je svetlo na helme baníka). Naopak najhoršie výsledky preukazuje v prípade smerového svetla paralelného s rovinou obrazu, kedy dostávame v normalizovanom priestore bodové svetlo umiestnené v nekonečne. Najhorším výsledkom pritom myslíme rovnaký výsledok ako pri použití štandardnej, uniformnej tieňovej mapy.

Ďalším problémom je zahrnutie všetkých potencionálnych objektov vrhajúcich tieň do scény, ktorý sa dá vyriešiť pomocou virtuálnych kamier, čo už ale presahuje hranice detailnosti tejto práce. Viacej o tejto problematike v [9] a [10].

Na záver treba povedať, že aj keď je toto rozšírenie jedno z najjednoduchších, poskytuje výborné výsledky, bohužiaľ iba v určitých prípadoch.



Obrázok 3.16: Rozdiel v použití štandartných a perspektívnych tieňových máp (Stamminger, Drettakis, [9])

3.5.2 Percentage-closer filtering (PCF)

Problémom s podvzorkovaním a následným hranatým okrajom tieňov sa niekedy pri technike tieňových máp nedá predísť. Vysoké rozlíšenie tieňovej mapy nemusí byť pri niektorých typoch aplikácií prípustným riešením. Existuje však technika, ktorá dokáže tento typ aliasingu zmierniť až úplne eliminovať. Narozdiel od techniky perspektívnych tieňových máp (alebo ďalších [11], [12], [13]) sa nejedná o riešenie, ktoré by sa snažilo vzniku aliasingu predchádzať, ale rieši jeho odstránenie až po jeho vzniku. Jedná sa o techniku Percentage-Closer Filtering (PCF).

Obrázok 3.17: Hladké okraje tieňov s použitím PCF (NVIDIA)

Ide o techniku, ktorá využíva programovateľného zobrazovacieho reťazca (shader) grafického čipu, konkrétne shader na spracovanie fragmentov. Jej výsledkom sú jemné, kontinuálne okraje tieňov, ktoré môžu napodobovať efekt mäkkých tieňov. Samozrejme, že sa jedná o nepravé mäkké tiene, ale výsledok môže byť v určitých prípadoch vizuálne postačujúci a okrem toho je výpočet takýchto nepravých tieňov niekoľkonásobne rýchlejší ako výpočet fyzikálne správnych, pravých mäkkých tieňov.

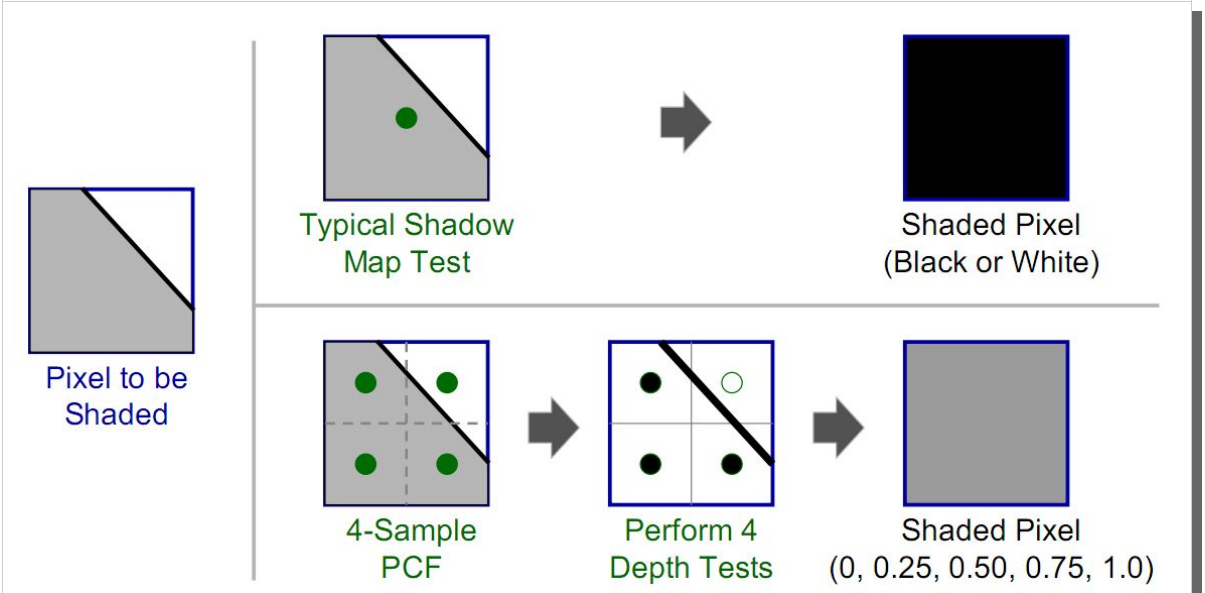
Hlavnou myšlienkou tejto techniky je priemerovací filter. Tieňová mapa sa však nemôže filtrovať ako obyčajná textúra, pretože neobsahuje informácie o farbe fragmentov, ale o ich hĺbke. Priemerovanie hodnôt hĺbky nedáva v tomto prípade veľký zmysel, pretože by to aliasing na okraji tieňov vôbec nevyriešilo. Ak by sme spriemerovali hodnoty hĺbky povedzme dvoch vedľa ležiacich bodov, dostali by sme bod s hĺbkou medzi týmito bodmi. Čo ale potrebujeme je zjemniť prechody medzi miestami, ktoré sú plne zatienené (nachádzajú sa v umbre) a miestami nezatienenými. Ide teda o akési napodobenie polotieňa, teda penumbry. Namiesto toho, aby sme filtrovali (priemerovali) hodnoty hĺbky v tieňovej mape, budeme filtrovať výsledky testu hĺbkovej mapy (depth map test). Zoberieme teda výsledky testu hĺbkovej mapy z určitého okolia daného fragmentu, výsledky spriemerujeme a výsledná hodnota bude určovať percentuálne zatienenie daného fragmentu. Napr. pri 4

vzorkách použitých na filtrovanie bude môcť fragment nadobúdať hodnoty 0%, 25%, 50%, 75% a 100%.

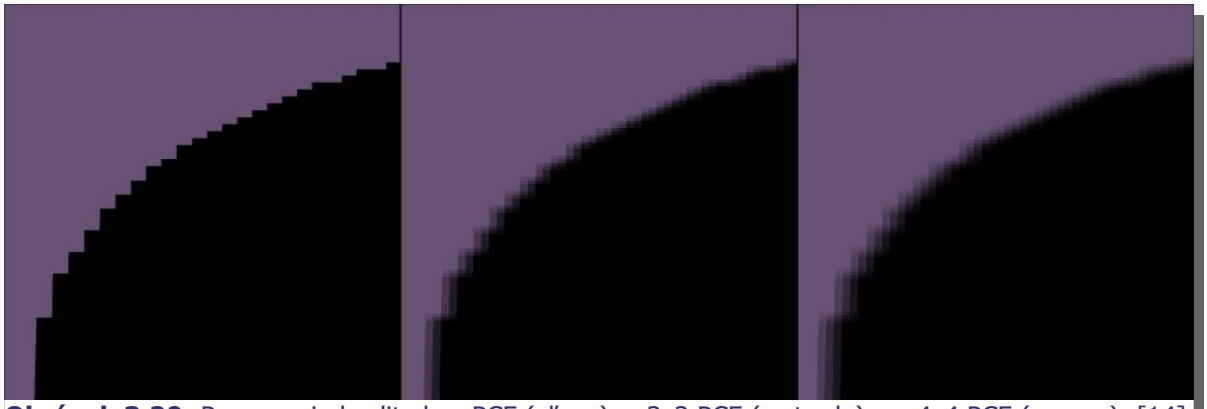
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	1
0	0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Obrázok 3.18: Jadro filtra PCF. 0 – neúspešný test hĺbkovej mapy, 1 – úspešný test

Pre lepšiu kvalitu okrajov tieňov je možné použiť oblasti 4x4, 8x8, 16x16, alebo ľubovoľné okolie, ktoré nám poskytne vyhovujúcu kvalitu. Je treba dodať, že so zväčšujúcim sa počtom vzoriek, ktoré sa použijú na vypočítanie farby tieňu, vzrastá výpočetná náročnosť.



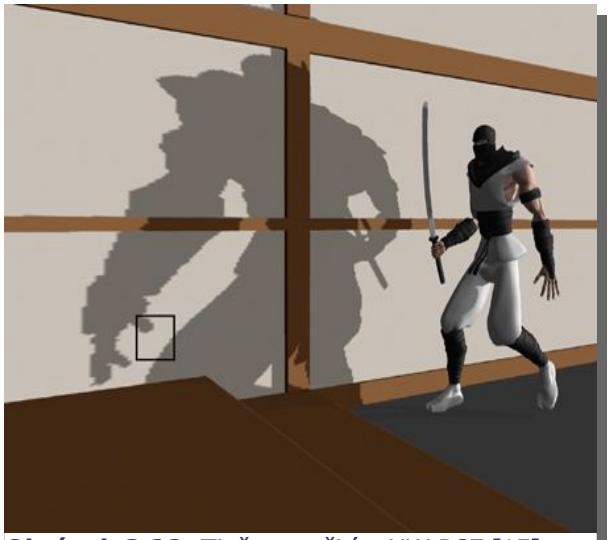
Obrázok 3.19: Ukážka pracovania PCF



Obrázok 3.20: Porovnanie kvality bez PCF (vľavo), s 3x3 PCF (v strede) a s 4x4 PCF (vpravo) [14]

Obrázok 3.21: PCF s použitím lineárneho filtrovania v HW [14]

V grafických kartách od firmy NVIDIA je dokonca algoritmus PCF hardwarovo zabudovaný už od rady grafických čipov GeForce3. Jedná sa o 2x2 PCF použité automaticky pri zapnutí bilineárneho filtrovania textúry (musí byť použité rozšírenie pre hĺbkovú textúru). Pri každom teste hĺbkovej mapy sa tak porovnáva štvorica pixelov a používa sa podielová časť textúrovacích súradníc na bilineárnu interpoláciu intenzity tieňa. Toto hardwarové PCF sa samozrejme dá naďalej kombinovať s PCF počítaným vo fragmen shaderi.



Obrázok 3.22: Tieň s použitím HW PCF [15]

Obrázok 3.23: HW PCF + 4x4 PCF v GPU [15]

Pôvodný PCF algoritmus počítal s mikropolygónmi ohraničujúcimi určitý počet fragmentov. Na dnešnom modernom hardware grafických čipov s programovateľnými reťazcami je však vhodnejšie využiť pre tento algoritmus fragmentové operácie.

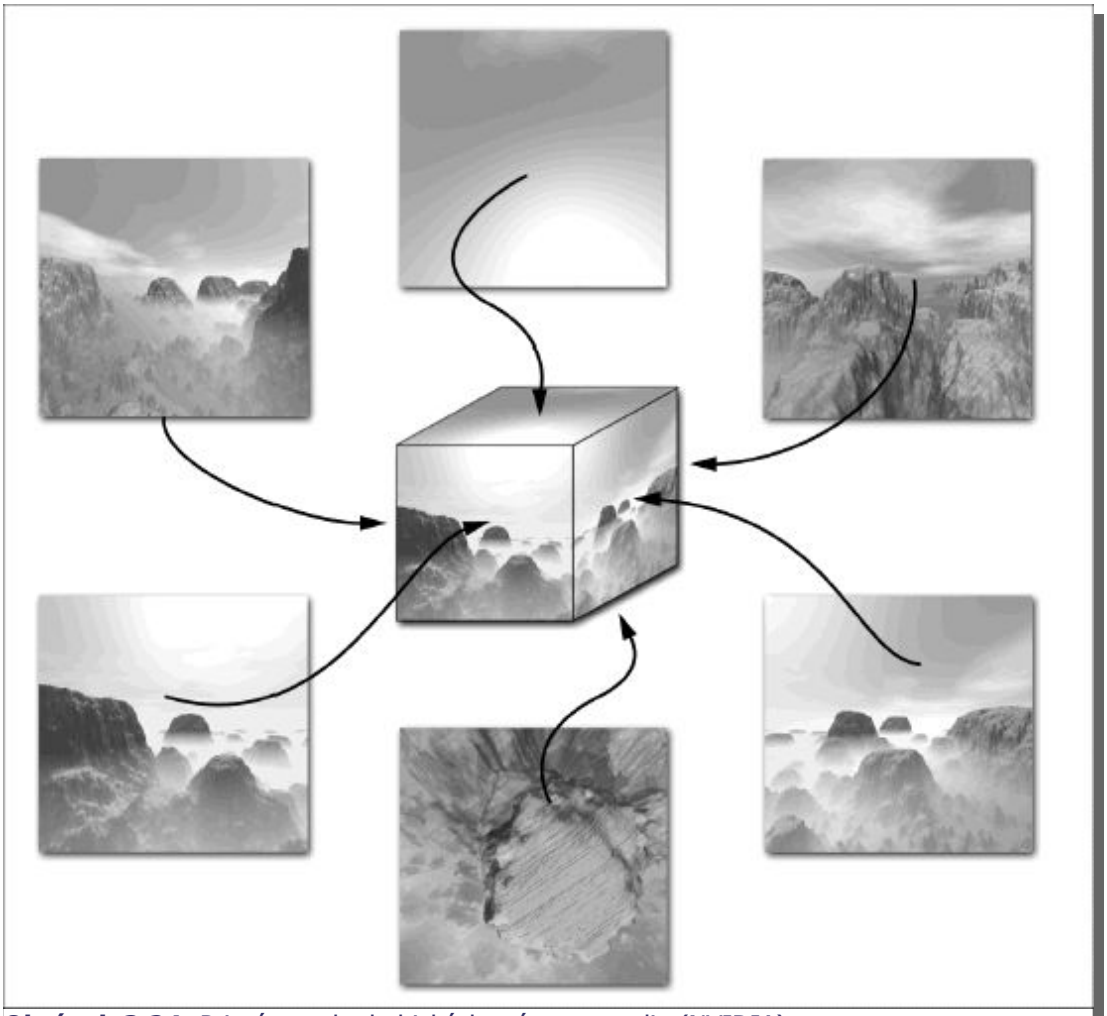
Ďalšie rozšírenie tohto postupu na vytvorenie realistickejších mäkkých tieňov poskytuje [16].

3.5.3 Ďalšie rozšírenia tieňových máp

Existuje ešte mnoho rozšírení algoritmu tieňových máp, a mnohé stále pribúdajú. Nie je možné do tejto práce zahrnúť všetky, preto si aspoň vymenujeme tie najdôležitejšie s odkazmi na literatúru pre prípadných záujemcov o ďalšie rozšírenie znalostí z tejto oblasti.

Tieňové mapy pre všesmerové bodové svetlá.

Doteraz sme sa zaoberali vytváraním tieňových máp z pohľadu kužeľových svetiel a smerových svetiel. Všesmerové bodové svetlá by však spôsobovali ťažkosti pri vytváraní tieňovej mapy, pretože by sme do nej nevedeli zaznamenať priestor v okolí 360 stupňov okolo svetla. Na tieto prípady sa používajú tzv. kubické mapy prostredia (*enviromental cube maps*), ktoré nie sú ničím iným ako 6-timi samostatnými mapami zachytávajúcimi prostredie v každom smere okolo svetla s uhlovým rozpätím pohľadového telesa 90 stupňov.



Obrázok 3.24: Princíp tvorby kubických máp prostredia (NVIDIA)

Výpočetná náročnosť celého procesu tvorby tieňov sa však takto výrazne zvýši, pretože potrebujeme 6+1 vykresľovacích prechodov (6-krát vykresľujeme bez výpočtu osvetlenia z pohľadu svetla, 1-krát vykresľujeme finálnu scénu s kompletným osvetlením). Preto sa táto myšlienka ďalej rozvíjala s cieľom vylepšiť ju a zmenšiť počet prechodov. Vznikla tak technika *dual-paraboloid shadow maps* [17, 18], teda dvoch hemisférických tieňových máp, ktorá znižuje počet prechodov na 2+1. Celé prostredie sa zaznamená pomocou dvoch, proti sebe postavených tieňových máp, z ktorých každá pokryje jednu hemisféru (180 stupňov).

Obrázok 3.25: Hemisférické tieňové mapy [18]

Adaptívne tieňové mapy

Rozšírenie algoritmu tieňových máp o vlastnosť adaptácie, teda prispôsobenie rozlíšenia tieňovej mapy. Zvýšenie rozlíšenia sa uplatňuje na okrajoch tieňov objektov v tieňovej mape, ktoré najviac prispievajú k zlepšeniu vizuálnej kvality. Vizuálne dôležité miesta sa hierarchicky delia podľa požadovanej kvality a stanovených pamäťových nárokov. Nevýhodou je potreba spätného čítania obrazového buffera z GPU. Viacej v [19]

Paralelne rozdelené tieňové mapy

Paralelne rozdelené tieňové mapy (*Parallel-split shadow maps*, PSSM) [12] delia pohľadové telesa na časti pomocou rovín paralelných k rovine pohľadu a generujú pre každú časť vlastnú tieňovú mapu. Delenie je realizované rýchlym a robustným systémom, založeným na aliasingu tieňovej mapy, čoho výsledkom je lepšie rozloženie vzorkovania v rámci celého rozsahu hĺbky. Použitím geometrickej aproximácie na každú rozdelenú časť tieňovej mapy zvlášť sa zvýši využitie rozlíšenia tieňovej mapy. Podobným spôsobom pracuje aj metóda kaskádových tieňových máp (*Cascade shadow maps*) [11].

Lichobežníkové tieňové mapy

Technika lichobežníkových tieňových máp (*Trapezodial shadow maps*) [13] aproximuje pohľadové teleso pozorovateľa videné z pohľadu svetla lichobežníkom, ktorým potom transformuje tieňovú mapu. Táto lichobežníková aproximácia spôsobuje lepšie rozloženie vzoriek hĺbky v tieňovej mape, čím poskytuje zvýšenú kvalitu tieňov pre objekty bližšie k pozorovateľovi. Princíp tejto metódy je veľmi podobný technike perspektívnych tieňových máp (3.5.1).

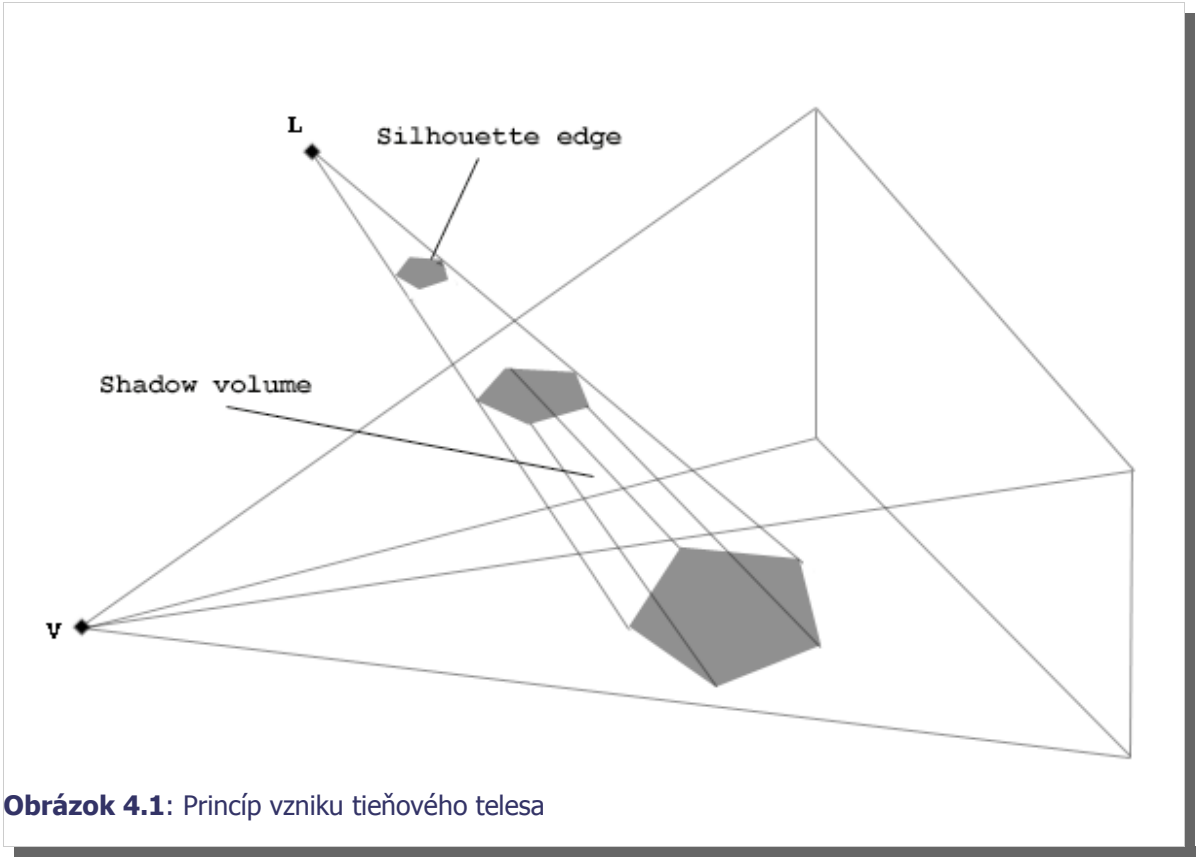
Projekčné tieňové mapovanie

Projekčné tieňové mapovanie nie je v pravom slova zmysle rozšírením algoritmu tieňových máp, skôr by sme ho mohli nazvať zjednodušením. Pracuje veľmi podobne ako pôvodný algoritmus, ale prvý krát v ňom nevystupuje tieňová mapa ako záznamový priestor pre hĺbku scény, ale iba ako čierno-biela textúra. Označujeme ju síce stále ako „tieňovú mapu" ale už nemáme na mysli „tieňovú mapu hĺbky". S informáciou o hĺbke objektov sa v tejto metóde

vôbec nepracuje. Postup metódy je nasledovný: objekt vrhajúci tieň sa vykreslí z pohľadu svetla a uloží do čierno-bielej textúry, na príjemcu tieňa sa táto textúra namapuje metódou projekčného mapovania (popísané v časti 3.3), nanesená textúra sa premieša s povrchom príjemcu, čím vzniká tieň. Výhodou tejto techniky je jednoduchosť a výpočetná nenáročnosť (keďže sa úplne vynecháva test hĺbkovej mapy), nevýhodou zase neschopnosť vrhať vlastné tiene.

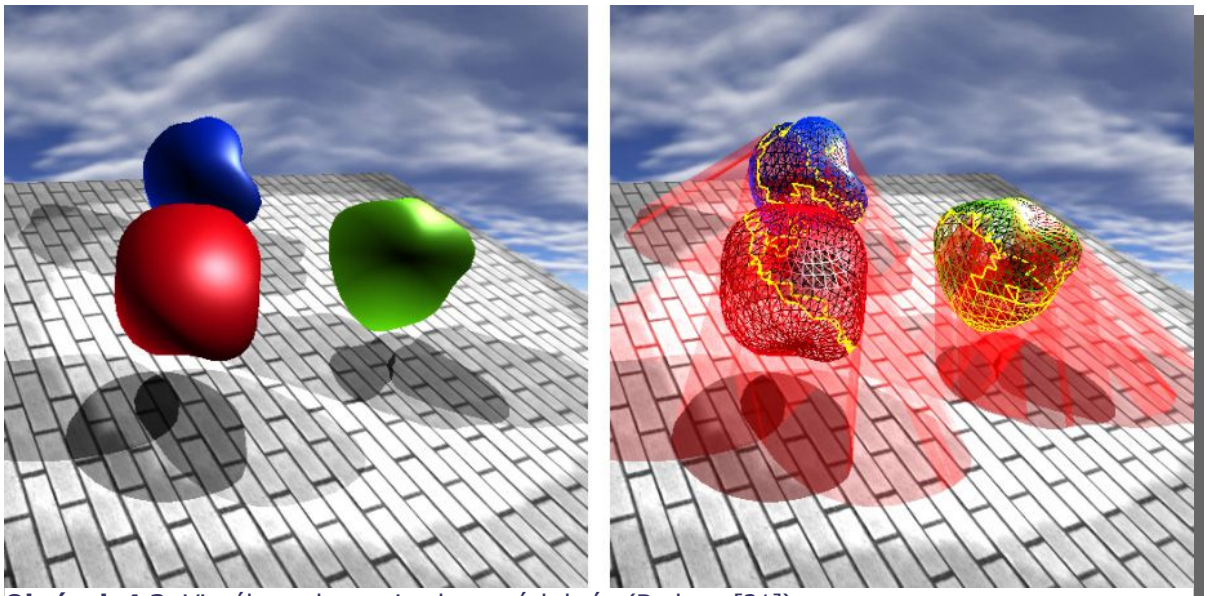
4 Tieňové telesá

Algoritmus tieňových telies (taktiež tieňových objemov, *shadow volumes*, *stencil shadows*) je jedným z najpoužívanejších techník počítačovej grafiky na výpočet tieňov v reálnom čase (spolu s algoritmom tieňových máp). Ako úplne prvý ho predstavil [20]. Jedná sa o algoritmus pracujúci v objektovom priestore (object space), to znamená, že na výpočet tieňov objektu používa informáciu o geometrii objektu. Základom metódy je tvorba tieňových telies (shadow volumes), ktoré vznikajú interakciou lúčov svetla s tieniacimi objektmi. Vznik tieňového telesa objektu je vidieť na obrázku 4.1.



Obrázok 4.1: Princíp vzniku tieňového telesa

Tieňové telesá potom v scéne vymedzujú priestor, kam sa cez tieniaci objekt svetelné lúče nedostanú, teda miesto, ktoré objekt zatieňuje. Algoritmus pracuje v základnej podobe s bodovými svetelnými zdrojmi a tvorí teda ostré tiene. Pri naivnom postupe hrubou silou sa môže tieňové teleso vytvárať pre každý jeho polygón. To má ale zásadný vplyv na rýchlosť výpočtu tieňov, a tak je vhodné (a aj častejšie používané) nájsť najprv siluetu tieniaceho objektu – obrys, ktorí tvorí hranicu tieňa, a vytvárať tieňové teleso iba pomocou nej. Hľadáme tzv. *silhouette edges*, to znamená hrany objektu, ktoré tvoria siluetu tieniaceho objektu pohľadom zo svetla. Každá obrysová hrana určuje jednu stenu tieňového telesa. Výpočet tieňového telesa pomocou týchto hrán je potom niekoľkonásobne rýchlejší.



Obrázok 4.2: Vizuálne zobrazenie obrysových hrán (Brabec, [21])

Aby sme dokázali z tieňových telies vytvoriť samotné tiene objektov je potreba nejakým spôsobom určiť význam jednotlivých stien tieňového telesa pri tvorbe tieňa. Ak poznáme všetky tieňové telesá v scéne, riešenie tieňov je možné previesť na riešenie viditeľnosti objektov v scéne voči týmto telesám. Existujú dva základné postupy na výpočet tieňov z tieňových telies. Jedným z nich je **depth-pass** algoritmus a druhým **depth-fail** algoritmus. Obidva sú si v základe dosť podobné, obidva využívajú pamäť šablóny (*stencill buffer*), no ich použiteľnosť v praxi je rozdielna.

4.1 Depth-pass algoritmus

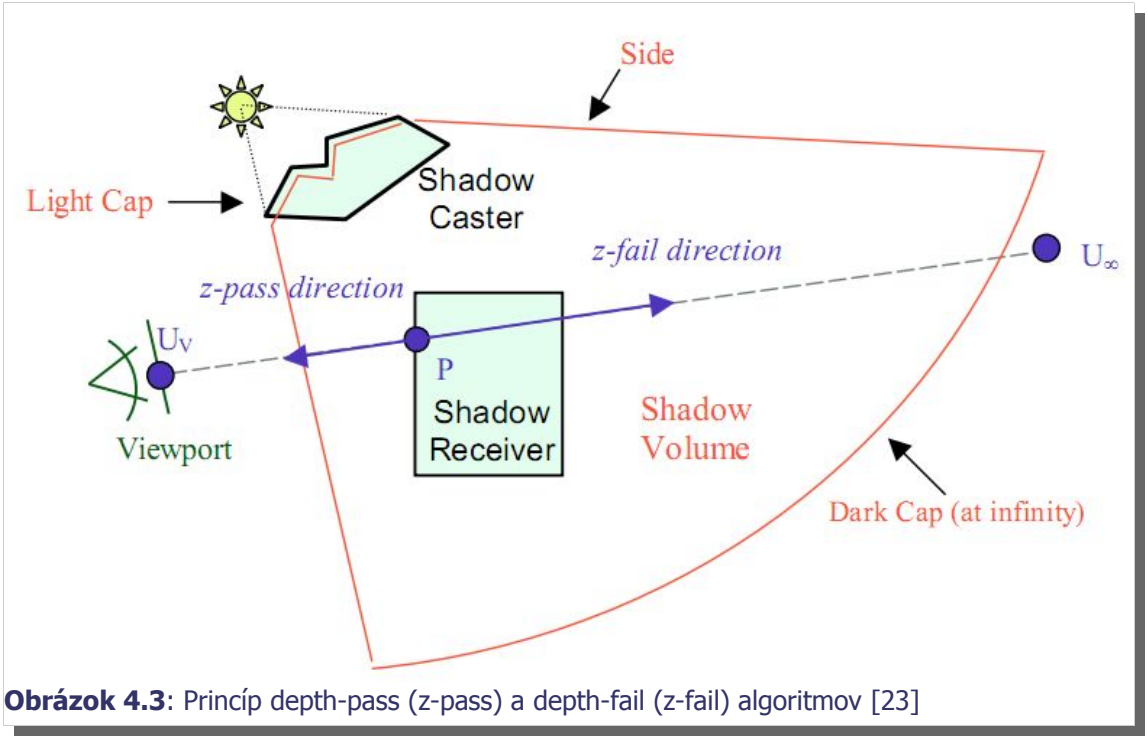
Algoritmus depth-pass (z-pass, označuje úspech testu pamäti hĺbky) [22] bol z hľadiska historického vývoja prvým hardwarovo akcelerovaným algoritmom na získanie tieňov z tieňových telies pomocou pamäti šablóny (*stencill buffer*). Aj keď o algoritme tieňových telesách hovoríme, že pracuje v objektovom priestore, jeho posledná časť, teda práve depth-pass algoritmus, pracuje už v priestore obrazovom.

Princíp tohto algoritmu je založený na vysielaní testovacích lúčov z bodu pozorovateľa smerom ku každému povrchu v scéne. Pritom sa zaznamenáva, ak testovací lúč preťal tieňové teleso, konkrétne sa počíta, koľkokrát testovací lúč do tieňového telesa vnikne a koľkokrát ho opustí.

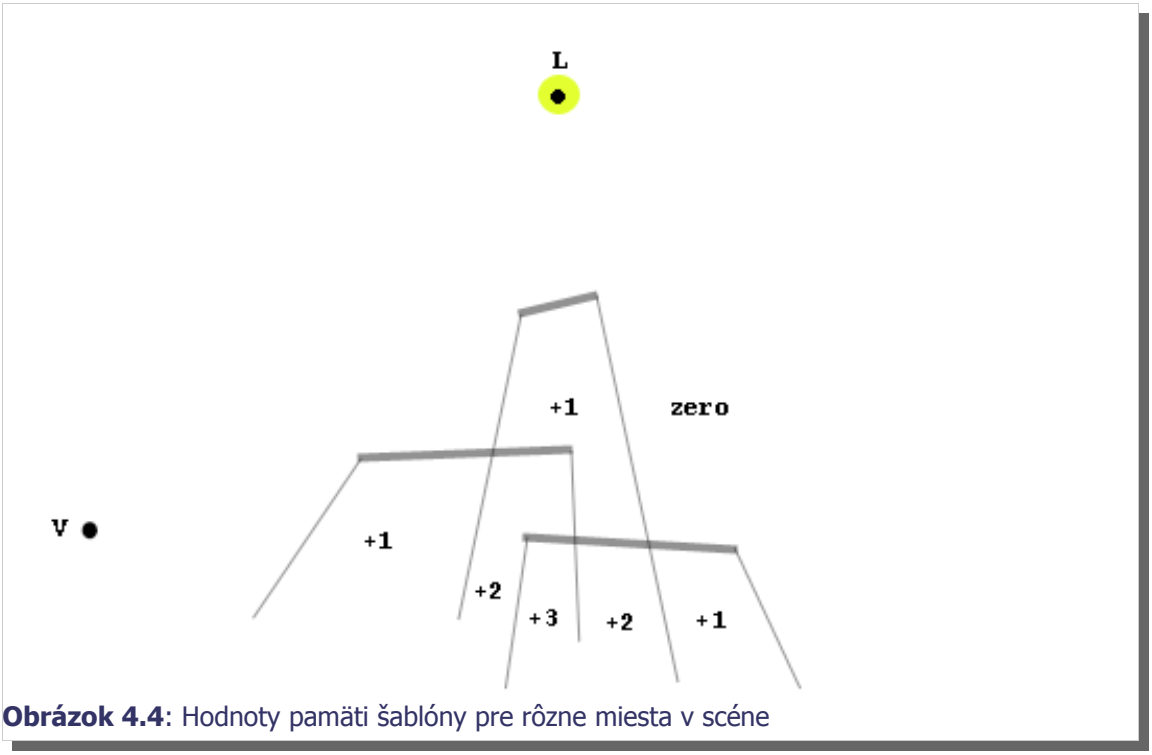
Na konci sa vypočíta rozdiel medzi počtom vniknutí a opustení tieňového telesa. Ak je tento rozdiel nenulový, znamená to, že testovací lúč neopustil všetky tieňové telesá, a teda povrch, na ktorý dopadol musí ležať vnútri tieňového telesa, teda v tieni. Pre lepšie pochopenie sa to dá popísať tak, že steny tieňového telesa, ktoré sú otočené ku kamere, zakrývajú všetko, čo leží za nimi, do tieňa, zatiaľčo steny od kamery odvrátené tento účinok rušia. Aby objekt, alebo nejaká jeho časť ležali v tieni, musí platiť, že sa nachádzajú za stenami otočenými ku kamere a zároveň pred odvrátenými stenami tieňových telies.

Na detekciu situácii, kedy testovací lúč vniká do tieňového telesa a kedy z neho vychádza, sa používa test pamäti hĺbky (depth test). Prvým krokom algoritmu depth-pass je obvyklé vyriešenie viditeľnosti v scéne (pomocou pamäti hĺbky) bez tieňových telies a uloženie, resp. zachovanie tejto informácie v pamäti hĺbky. Potom priradíme každému bodu

v scéne čítač, ktorého počiatočnú hodnotu inicializujeme na 0. Následne sa uplatní hĺbkový test polygónov (stien) tieňového telesa s ostatnými objektmi v scéne. Ak bude tento test pre daný bod v scéne úspešný (depth-pass), to znamená, že sa polygón tieňového telesa nachádza pred všetkými objektmi v scéne, potom v prípade, že sa jedná o plochu otočenú smerom ku kamere, čítač inkrementujeme, a naopak v prípade plochy odvrátenej od kamery, čítač dekrementujeme. Pokiaľ po spracovaní všetkých plôch tieňových telies v scéne bude hodnota čítaču nenulová, nachádza sa daný bod v tieni. V praxi sa na realizáciu čítaču pre každý bod v scéne používa buffer šablóny (stencil buffer).



Obrázok 4.3: Princíp depth-pass (z-pass) a depth-fail (z-fail) algoritmov [23]

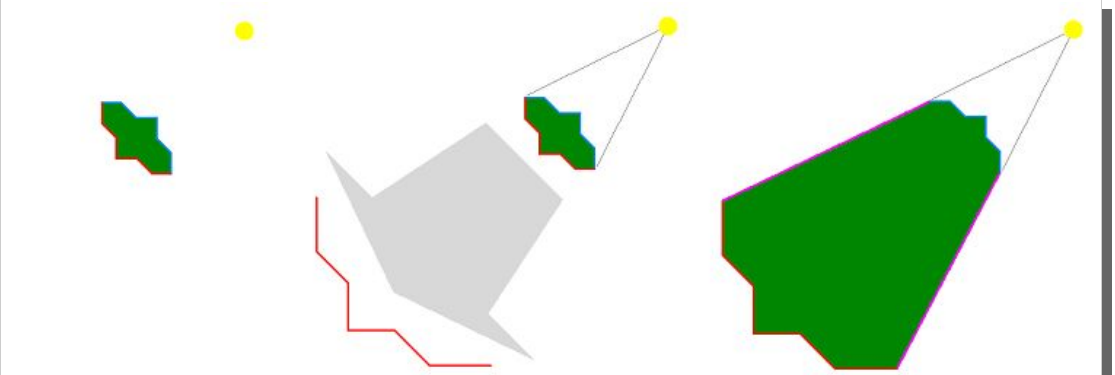


Obrázok 4.4: Hodnoty pamäti šablóny pre rôzne miesta v scéne

4.2 Depth-fail algoritmus

Druhý z používaných algoritmov na určenie tieňov pomocou tieňových telies – depth-fail (z-fail) [24, 25] algoritmus vychádza z toho prvého – depth-pass algoritmu. Jeho myšlienkou je, že obracia zmysel depth-pass algoritmu, to znamená, že sa hodnota čítača mení nie vtedy, ak test pamäte hĺbky uspeje, ale keď zlyhá. Ten zlyháva vtedy, ak je stena tieňového telesa za testovaným bodom v scéne. Týmto spôsobom smeruje testovací lúč z nekonečnej vzdialenosti k zobrazovanému povrchu do testovaného bodu, čo je opačný smer ako pri depth-pass algoritme.

Depth-fail algoritmus, narozdiel od predchádzajúceho depth-pass algoritmu, netrpí problémami s prednou orezávaciou rovinou kamery a je naviac nezávislý na počiatočnej polohe kamery. V tom sú jeho najväčšie výhody, pretože je tak v praxi oveľa použiteľnejší. Prináša však aj nové požiadavky, a to požiadavky na uzavretie tieňového telesa. K bočným stenám tieňového telesa sa tak musia pridať **predný uzáver** (*front cap*, *light cap*) a **zadný uzáver** (*back cap*, *dark cap*). Je taktiež potreba dávať pozor na polohu zadnej orezávacej roviny pohľadového telesa kamery, ktorá by mohla uzavretosť tieňového telesa narušiť.



Obrázok 4.5: Znázornenie predného (modrý) a zadného uzáveru (červený) (Microsoft DirectX SDK)

Predný uzáver je tvorený plochami tieniaceho objektu privrátenými ku svetlu (preto light cap), zadný uzáver je potom tvorený projekciou od svetla odvrátených plôch do vzdialenosti, o ktorú sme predĺžili bočné steny tieňového telesa.

Obrázok 4.6: Tieňové telesá v hre Doom3 (ID, 2004)

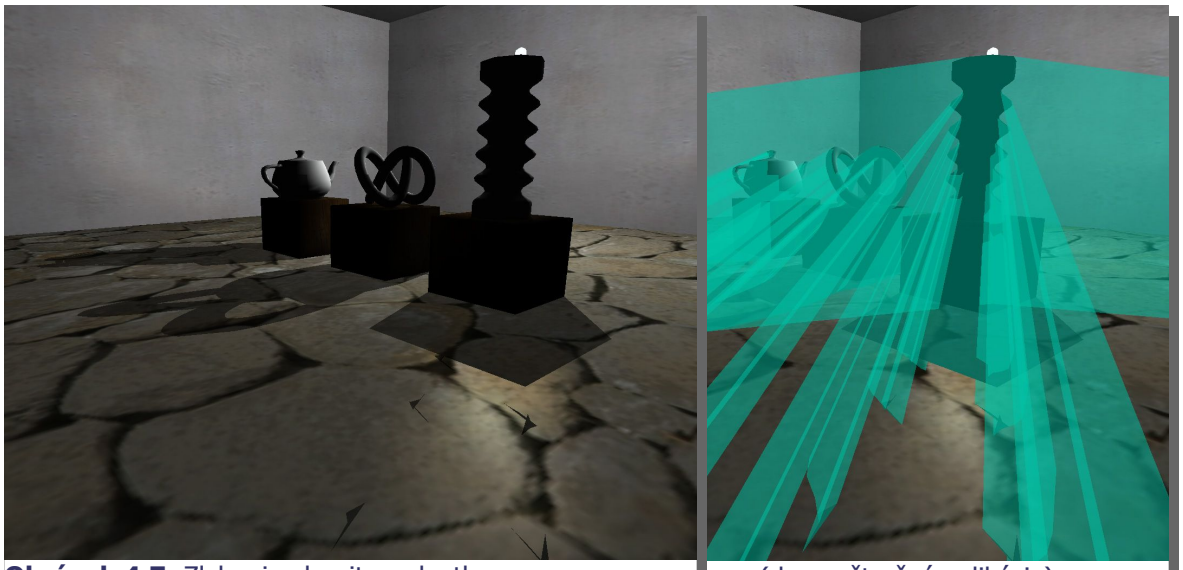
4.3 Problémy

V súvislosti s tým, že algoritmus tieňových telies je algoritmom pracujúcim v objektovom priestore, je treba spomenúť jednu jeho dôležitú vlastnosť. S narastajúcou geometrickou zložitosťou objektu (teda s narastajúcim počtom polygónov, z ktorých sa daný objekt skladá), rastie taktiež zložitosť celého algoritmu. Počet polygónov tieniaceho objektu priamo ovplyvňuje počet stien (polygónov) tieňového telesa. Pri komplexnejších modeloch to preto môže znamenať výrazný pokles počtu snímkov za sekundu (*framerate*, *frames per second*, FPS) oproti modelom s menšou komplexnosťou.

Potreba geometrickej informácie o objekte na vykreslenie jeho tieňa je taktiež dôvodom prečo sa algoritmus nehodí na použitie s objektmi, ktoré sa zobrazujú s určitou priehľadnou zložkou (*sprites*, *billboards*) alebo s objektmi s neuzavretou polygonálnou sieťou (*non-closed meshes*).

4.3.1 Problémy algoritmu depth-pass

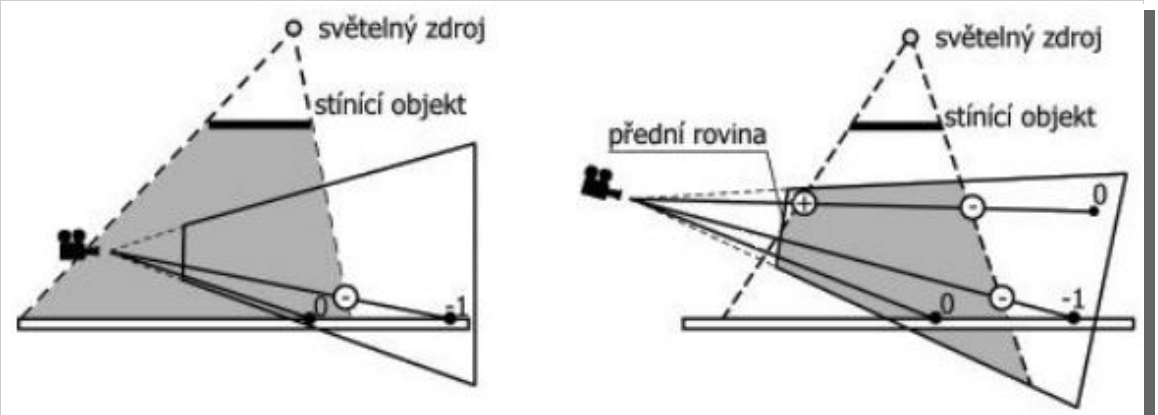
Algoritmus depth-pass má jednu dôležitú nevýhodu. Ak sa kamera nachádza vnútri niektorého z tieňových telies, algoritmus nepracuje správne a zobrazenie tieňov objektov sa naruší. Prečo je tomu tak? Dochádza k prípadu, keď sa nezaznamená vniknutie do tieňového telesa, a body ležiace vnútri tieňového telesa budú mať nulovú hodnotu čítaču. Nulová hodnota znamená, že bod je nezatienený, ostáva teda osvetlený aj napriek faktu, že zjavne leží vnútri tieňového telesa.



Obrázok 4.7: Zlyhanie algoritmu depth-pass (demonštračná aplikácia)

Keďže je to v praxi celkom bežný prípad, algoritmus depth-pass má obmedzené použitie v praxi (väčšinou sa kombinuje s algoritmom depth-fail).

A nejde iba o umiestnenie kamery v tieňovom telese. Problém tkvie v prednej a zadnej orezávacej rovine kamery. Môže totiž nastať situácia, že niektoré steny tieňového telesa budú orezané prednou alebo zadnou rovinou pohľadového telesa kamery. To spôsobí nesprávny výpočet hodnoty čítaču a následné narušené alebo úplne nesprávne určenie tieňov.



Obrázok 4.9: Dôvod zlyhania algoritmu depth-pass [2]

4.3.2 Problémy algoritmu depth-fail

Jeden z problémov algoritmu depth-fail spočíva v nutnosti zaistiť uzavretosť tieňového telesa. Ak dôjde k porušeniu uzavretosti tieňového telesa, nedôjde k správnemu vymedzeniu oblasti tieňa a tým pádom môžu mať tiene objektov trhliny alebo sa nezobrazovať vôbec.

Ďalším menším problémom tohto algoritmu je problém so zadnou orezávaciou rovinou pohľadového telesa. Tá nielenže môže narušiť uzavretosť tieňového telesa, ale taktiež spôsobuje nesprávne zobrazenie tieňov, ak pozorovateľovi vo výhľade na ňu nebráni žiadna prekážka. Nejedná sa ale o závažný problém, dá sa jednoducho odstrániť posunutím zadnej orezávacej roviny do nekonečna [26].

Obrázok 4.10: Zlyhanie algoritmu depth-fail (demonštračná aplikácia)

Okrem týchto dvoch neduhov nemá algoritmus vážnejšie problémy a nemusí riešiť žiadne špeciálne prípady pre správne zobrazovanie tieňov objektov. Pokiaľ teda zaistíme uzavretosť tieňového telesa a posunutie zadnej orezávacej roviny pohľadového telesa do nekonečna, dostávame robustnú metódu na zobrazovanie tieňov pomocou tieňových telies.

4.4 Rozšírenia

Už samotný algoritmus depth-fail je rozšírením pôvodného algoritmu tieňových telies [20, 22], avšak pretože sa stal bežne zaužívaným, neuvažuje sa o ňom ďalej už ako o rozšírení, ale skôr ako o súčasti novodobého algoritmu tieňových telies.

Čo už ale za rozšírenie považovať môžeme je práca pánov Everitta a Kilgarda [26], ktorý vo svojom článku doporučujú nielen posun zadnej orezávacej roviny pohľadového telesa do nekonečna, ale ukazujú, že nie vždy je voľba algoritmu depth-fail potrebná. Jednoduchým geometrickým testom sa dá zistiť, či je pohľadové teleso vnútri tieňového telesa, a tým pádom použiť depth-fail, iba v prípade potreby. Takéto obmedzenie použitia depth-fail algoritmu má svoje dôvody, ktoré sa týkajú hlavne výkonnosti.

Práca [27] zohľadňuje rozlišovanie situácie vhodnej pre depth-pass a depth-fail a vytvára tak svoj vlastný, hybridný algoritmus. Ďalšie rozšírenie, ktoré táto práca uplatňuje, sa týka vymedzenia regiónu, v ktorom má zmysel počítať tieňové telesá pre bodové svetlá. Bodové svetlo osvetľuje scénu iba do určitej, konečnej vzdialenosti, a preto je vhodné určiť hranice, za ktorými je už zbytočné vykresľovať polygóny tieňových telies, pretože tam už svetlo do scény nezasahuje. Tento priestor sa teda vymedzí určitým ohraničujúcim telesom, v prípade [27] guľou (*bounding sphere*), ktorá orezáva presahujúce tieňové telesá. To so sebou môže v niektorých prípadoch prinášať značné výkonnostné úľavy.

Ďalšie rozšírenia postavené na predchádzajúcich optimalizáciách prináša [23]. Ide napr. o použitie jednoduchšej kocky ako ohraničujúceho telesa (*bounding box*, resp. *bounding cube*) pre určenie hraníc pôsobnosti bodového svetla alebo vymedzenie hraníc hĺbkového testu na základe najbližšej a najviac vzdialenej hodnoty hĺbky vo vymedzenom regióne (*depth bounds test*). Vymedzeným regiónom sa myslí viditeľný, osvetlený priestor v scéne. Ten je tvorený priesečníkom pohľadového telesa, hraničného telesa svetla a hraničným telesom tieňového objemu. Hĺbkový test sa teda uplatňuje iba na oblasti, ktorá je ovplyvňovaná svetlom, čo znižuje počet inkrementácií a dekrementácií v pamäti šablóny a prejavuje sa zlepšením výkonnosti.

Vylepšením algoritmu tieňových telies sa zaoberá aj [21], konkrétne poskytuje nové prístupy týkajúce sa detekcie siluety a zníženia počtu vykresľovacích priechodov ukladaním priebežnej informácie do tieňovej masky.

5 Techniky výpočtu tieňov v praxi

V nasledujúcej kapitole sa budeme zaoberať použitím vybraných techník v praxi. Pozrieme sa na to, aké nástrahy a úskalia nás môžu čakať pri ich implementácii do real-time zobrazovacieho systému, ktorým bude pre túto prácu vytvorený, demonštračný prehliadač 3D scén. Nakoniec porovnáme vybrané techniky medzi sebou, a to z hľadiska výkonnosti, vlastností a vhodnosti použitia. Táto kapitola bude hľadať odpoveď na otázku, či existuje jedna nadradená technika, prekonávajúca ostatné vo všetkých podstatných aspektoch, vhodná na použitie do každého zobrazovacieho systému pracujúceho v reálnom čase alebo musíme vždy voliť kompromis a kombináciu známych techník.

5.1 Implementácia vybraných techník

Pre implementáciu vybraných techník som si zvolil grafickú knižnicu OpenGL a jazyk C++. Pomocou nich som vytvoril demonštračnú aplikáciu, ktorá je schopná načítať 3D scénu zo súboru a zobraziť ju s použitím jednej z vybraných techník na vykreslenie tieňov v reálnom čase. Medzi technikami je potom možné počas behu aplikácie ľubovoľne prepínať. V scéne je možné sa pohybovať a taktiež pohybovať so svetelným zdrojom.

5.1.1 Rovinné tiene

Implementácia techniky rovinných tieňov nie je, tak ako technika sama, ničím zložitým. Budeme používať Blinnov algoritmus [4], pretože je všeobecnejší a dosahuje najlepšie výsledky.

Najprv je potreba nájsť rovnicu roviny, na ktorej sa tiene budú zobrazovať. Rovina je určená 3 bodmi, takže aby sme našli rovnicu roviny v tvare

$$ax+by+cz+d=0$$

potrebujeme 3 body ležiace v nej. V praxi to znamená, že použijeme 3 vrcholy plochy, ktorá predstavuje v scéne tieňovú rovinu. V poradí smeru hodinových ručičiek (čo sa týka orientácie polygónov danej roviny) dosadíme tieto 3 vrcholy do funkcie `m3dGetPlaneEquation`, čím dostávame štvorzložkový vektor obsahujúci parametre rovnice roviny ***a***,***b***,***c*** a ***d***. Použitá funkcia vypočíta normálový vektor roviny ***n=[a,b,c]*** vektorovým súčinom dvoch vektorov získaných z troch bodov roviny a potom dopočíta posledný parameter ***d*** skalárnym súčinom normálového vektoru s ľubovoľným bodom roviny.

Ďalším krokom implementácie je vytvorenie matice tieňa (*shadow matrix*) v tvare:

$$M=\begin{bmatrix} \vec{n}\vec{l}+d-l_x n_x & -l_x n_y & -l_x n_z & -l_x d \\ -l_y n_x & \vec{n}\vec{l}+d-l_y n_y & -l_y n_z & -l_y d \\ -l_z n_x & -l_z n_y & \vec{n}\vec{l}+d-l_z n_z & -l_z d \\ -n_x & -n_y & -n_z & \vec{n}\vec{l} \end{bmatrix}$$

Vytvorenie matice tieňa sa deje vo funkcii `buildPlanarShadowMatrix()`, ktorá naplní maticu pomocou rovnice roviny a pozície svetla (popis zložiek matice v časti 2).

Ak máme hotovú maticu tieňa, môžeme ňou transformovať objekt, ktorý má vrhať tieň. Transformácia prebieha vynásobením stávajúcej modelovej matice maticou tieňa. V OpenGL teda použijeme funkciu `glMultMatrix()`. Takto transformovaný objekt môžeme potom vykresliť bez výpočtu osvetlenia alebo textúrovacej informácie. Vykresľujeme zvyčajne v šedej až čiernej farbe so zapnutým premiešavaním obrazu (`GL_BLEND`) s nastavením premiešavacej funkcie `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`, a aby sme predišli problémom s prelínaním sa polygónov, tak aj s vypnutým testovaním pamäti hĺbky. Vyššie uvedené sa deje vo funkcii `DrawPlanarShadow()`.

Zatiaľ popísaným postupom by sme sa ale nevyhli problému s double blendingom. Preto ešte musíme vymedziť v pamäti šablóny miesta, na ktorých sa má tieň zobrazovať a zaistiť, aby sa tieň nekreslil na miesta, ktoré už zatienené sú. Aby sme zaručili, že tieň sa bude zobrazovať iba na ploche, ktorá predstavuje v scéne rovinu tieňa, a nie mimo jej okrajov, vykreslíme najprv túto plochu do pamäti šablóny bez zápisu do pamäti farby alebo pamäti hĺbky. Miesta, ktoré plocha v scéne pokrýva označíme v šablóne určitou unikátnou hodnotou, takže každá plocha predstavujúca tieňovú rovinu bude mať svoje vlastné označenie, resp. identifikačné číslo. V demonštračnej aplikácii je použitý zoznam rovín, kde sa nachádzajú všetky objekty (plochy) označené ako roviny tieňa (majú v názve „ShadowPlane"). Ako jednoznačný identifikátor roviny v pamäti šablóny je potom použitý index (poradie) v tomto zozname. Týmto spôsobom budeme schopní vykresľovať tieň jedného objektu na viacerých tieňových rovinách.

Sme pripravený vykresliť tieň objektu. Zapneme zápis do pamäti farby a hĺbkový test. Nasleduje vykreslenie tieňa funkciou `DrawPlanarShadow()`, a to iba na miestach vymedzených identifikátorom roviny. Po zápise do tejto oblasti v scéne sa hodnota v pamäti šablóny dekrementuje (pri úspechu hĺbkového testu), čo zabráni opätovnému prekresleniu už raz zatienených oblastí a odstráni nežiadúci efekt double blendingu.

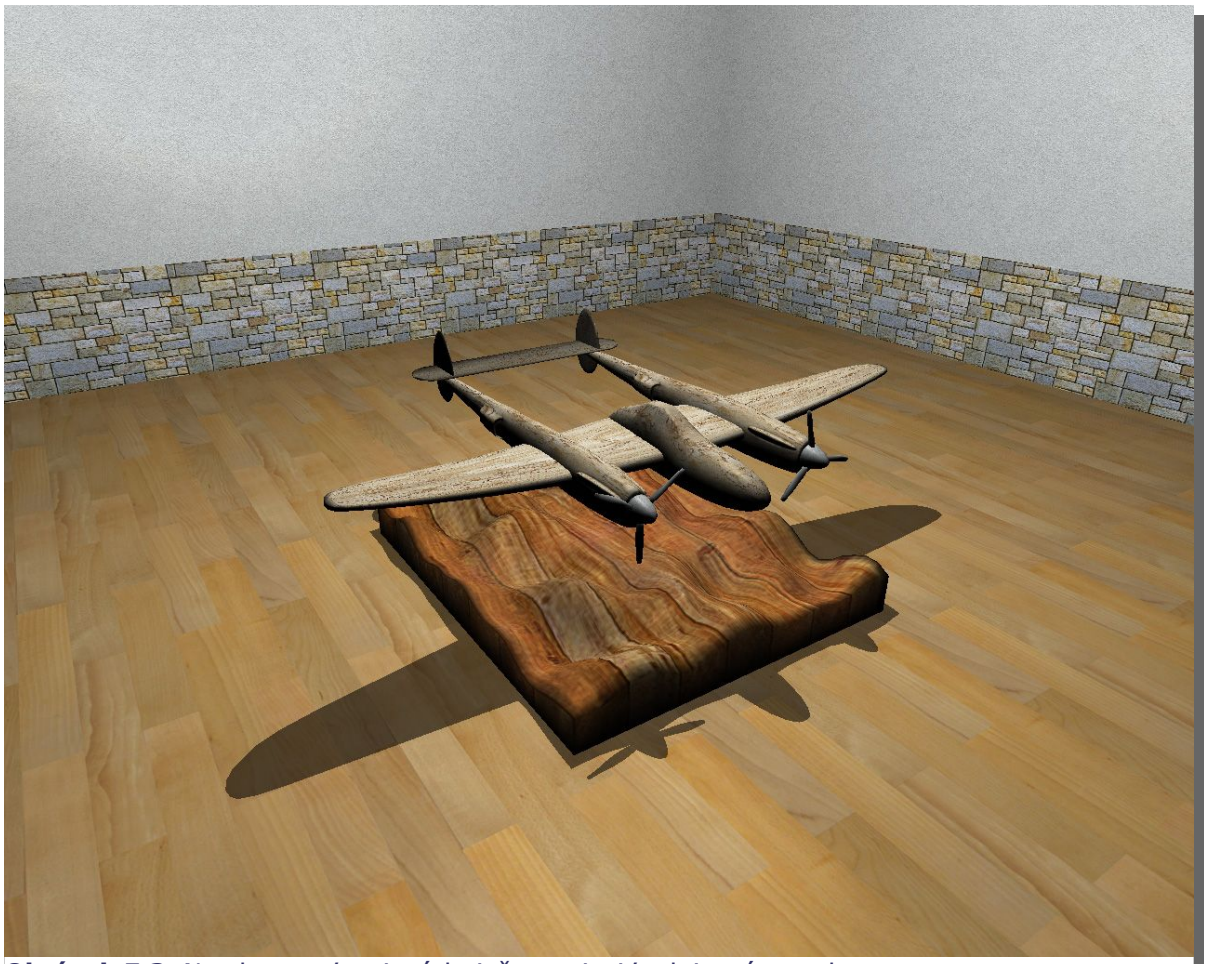
Tiene sú tým pádom už správne ošetrené a vykreslené a môžeme vykresliť vlastné objekty scény.

```
for (int i = 0; i < ShadowPlanes_list.size(); i++)
{
      glColor3f(0.4f, 0.5f, 0.5f);
      // draw shadow plane
      ShadowPlanes_list[i]->Draw();
      buildPlanarShadowMatrix(planarShadowMatrix,
      light->position, ShadowPlanes_list[i]);
      // DRAW SHADOW PLANE INTO STENCIL
      glColorMask(0.0f, 0.0f, 0.0f, 0.0f);
      glDepthMask(GL_FALSE);
      glEnable(GL_STENCIL_TEST);
      // write an INDEX of the shadow plane to the stencil
         buffer everywhere we are about to draw
```

```
        glStencilFunc(GL_ALWAYS, i+1, 0xFFFFFFFF);
        glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE);
        ShadowPlanes_list[i]->Draw();

        // DRAW PLANAR SHADOW where the stencil buffer has
           the INDEX
        glColorMask(1.0f, 1.0f, 1.0f, 1.0f);
        glDepthMask(GL_TRUE);
        glStencilFunc(GL_EQUAL, i+1, 0xFFFFFFFF);
        // decrease stencil where shadow has been drawn on
           zpass
        glStencilOp(GL_KEEP, GL_KEEP, GL_DECR);
        DrawPlanarShadow(planarShadowMatrix);
        glDisable(GL_STENCIL_TEST);
}

// DRAW SHADOW CASTERS ONLY
glColor3f(MatDiffuse[0], MatDiffuse[1], MatDiffuse[2]);
glDisable(GL_CULL_FACE);
DrawPlanarShadowCasters();
glEnable(GL_CULL_FACE);
```

Obrázok 5.1: Polygonálne jednoduchá scéna v demonštračnej aplikácii

Obrázok 5.2: Polygonálne zložitá scéna v demonštračnej aplikácii



Obrázok 5.3: Neschopnosť rovinných tieňov zatieniť zakrivené povrchy

5.1.2 Tieňové mapy

Inicializačná fáza

Inicializačná fáza algoritmu tieňových máp pozostáva z prípravy hĺbkovej mapy. Na využitie hardwarovej podpory algoritmu a použitie hĺbkovej mapy budeme musieť inicializovať špeciálne rozšírenia `GL_ARB_DEPTH_TEXTURE` a `GL_ARB_SHADOW`. Prvé umožňuje optimalizovať uloženie hodnôt v hĺbkovej mape a operácie s ňou. Druhé sa stará o porovnávanie hodnôt hĺbky scény s tieňovou mapou.

Postupujeme ako pri použití obvyklej textúry, v niektorých veciach sa však inicializácia hĺbkovej mapy od klasickej textúry mierne líši. Pri špecifikácii typu hodnôt v tieňovej mape požijeme vo funkcii `glTexImage2D` na mieste tretieho parametra konštantu `GL_DEPTH_COMPONENT`, čím dávame grafickej karte informáciu, že v tejto textúre budú hodnoty hĺbky. Je vhodné nastaviť okraj textúry s nulovou hodnotou. Pri mapovaní textúry totiž používame opakovanie posledného pixelu okraju (*clamp to border*) čím zabránime „unikaniu" tieňa (*shadow leaking*). Špeciálnymi nastaveniami sú parametre `GL_TEXTURE_COMPARE_MODE_ARB`, `GL_TEXTURE_COMPARE_FUNC_ARB` a `GL_DEPTH_TEXTURE_MODE_ARB`, ktoré sa nastavujú funkciou `glTexParameter`. Prvý z nich zapína porovnávanie hĺbkovej pamäte s hĺbkovou mapou. Druhý určuje funkciu tohto porovnávania, teda podmienku, pri ktorej test hĺbkovej mapy uspeje. Nastavujeme ho tak, že

test uspeje v prípade ak je porovnávaná hodnota zo scény *r* menšia alebo rovná ako hodnota v hĺbkovej mape. Vtedy je daný pixel osvetlený, ak test neuspeje, teda *r* je väčšie ako hodnota v tieňovej mape, je zatienený. *r* pritom predstavuje jednu z textúrovacích súradníc *s*, *t*, *r*, *q*. Súradnice *(s/q, t/q)* určujú pozíciu fragmentu v hĺbkovej mape a *r/q* predstavuje *z*-tovú súradnicu v priestore okna (*window space*) relatívnu ku svetlu. *r* teda reprezentuje vzdialenosť od svetla.

Posledný parameter `GL_DEPTH_TEXTURE_MODE_ARB` určuje typ výsledku testu hĺbkovej mapy. Výsledok môže byť typu `LUMINANCE`, `INTENSITY` a `ALPHA` (podrobnosti v manuále: http://www.opengl.org/registry/specs/ARB/shadow.txt).

Nastavením parametrov `GL_TEXTURE_MIN_FILTER`, `GL_TEXTURE_MAG_FILTER` na `GL_LINEAR` vynútime porovnávanie s viacerými hodnotami z hĺbkovej mapy a získame tak automatické hardwarové 2x2 PCF, naviac s lineárnou interpoláciou výsledku.

```
// Create the shadow map texture
glGenTextures(1, &shadowMapTexture);
glBindTexture(GL_TEXTURE_2D, shadowMapTexture);
glTexImage2D( GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, shadowMapSize,
            shadowMapSize, 0, GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE,
            NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, white);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
            GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
            GL_CLAMP_TO_BORDER);
// Enable shadow comparison
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE_ARB,
            GL_COMPARE_R_TO_TEXTURE);
// not in shadow if r<=texture
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC_ARB,
            GL_LEQUAL);


// INTENSITY result
glTexParameteri(GL_TEXTURE_2D, GL_DEPTH_TEXTURE_MODE_ARB,
GL_INTENSITY);
```

Fáza implementácie algoritmu

Implementácia algoritmu tieňových máp začína vykreslením scény z pohľadu svetla. Na to budeme potrebovať posunúť kameru do pozície svetla, nasmerovať ju požadovaným smerom a nastaviť správnu projekciu. Nastavenie pozície kamery a smeru pohľadu svetla

má na starosti funkcia `gluLookAt`, projekciu potom nastavíme podľa charakteru použitého svetla. Buď použijeme perspektívnu projekciu pre bodový svetelný zdroj (funkciou `gluPerspective`), alebo ortogonálnu projekciu pre smerový svetelný zdroj (`glOrtho`). Po vykonaní príslušných funkcií sa v modelview matici nachádza transformácia, ktorá zodpovedá zmene pohľadu, resp. projekcie. Tieto matice si potrebujeme uložiť, pretože ich budeme neskôr potrebovať pri mapovaní tieňovej mapy na scénu. Spravíme to nasledujúcim spôsobom: po zavolaní `gluLookAt` uložíme maticu pohľadovej transformácie funkciou `glGetFloat` s parametrom `GL_MODELVIEW_MATRIX`, do modelview matice načítame identitu, zavoláme `gluPerspective` a opäť uložíme maticu, tentokrát sa bude jednať o maticu projekčnej transformácie.

Správne nastavenie projekcie je veľmi dôležité vzhľadom na kvalitu tieňovej mapy a teda i výsledného tieňa. Musíme napr. zaistiť dostatočný uhlový rozsah pohľadu (***field-of-view***) tak, aby scéna vykreslená z pohľadu svetla obsahovala všetky objekty, od ktorých požadujeme, aby vrhali tieň. So šírkou pohľadu sa to ale nemôže preháňať, pretože veľký uhol znamená zníženie presnosti hĺbkovej pamäti a tým pádom môže spôsobiť nedostatočne hustú distribúciu vzoriek hĺbky. Následkom čoho by vznikol nežiadúci aliasing z podvzorkovania. Obvyklé pozorovacie uhly pre pohľad zo svetla sú od 45-90 stupňov. Nad 90 stupňov je už vplyv zníženia presnosti pamäti hĺbky na kvalitu tieňa dosť značný. Pozícia prednej a zadnej orezávacej roviny môže hrať taktiež dôležitú úlohu distribúcii hodnôt v hĺbkovej mape. Preto sa doporučuje posunúť prednú orezávaciu rovinu, čo najďalej od oka pozorovateľa a čo najbližšie k prvému objektu, ktorý je pôvodcom tieňa. Zadná orezávacia rovinu už potom nehrá až takú významnú rolu.

Po správnom nastavení pohľadového telesa svetla ešte obmedzíme oblasť vykresľovania na veľkosť tieňovej mapy pomocou `(glViewPort(0,0, shadowMapWidth, shadowMapHeight))`.

Potom už môžeme pristúpiť k vykresleniu scény z pohľadu svetla. Vypneme výpočet osvetlenia, textúrovanie a zápis do pamäti farby, necháme zapnutý iba zápis do pamäti hĺbky. Aby sme predišli problému so self-shadow aliasingom, budeme pamäti hĺbky vykresľovať zadné, odvrátené strany polygónov. Nastavíme to funkciou `glCullFace` s parametrom `GL_FRONT`. Ak sa problémy s povrchovým akné vyskytnú aj napriek tomu, môžeme ešte použiť posunutie polygónov funkciou `glPolygonOffset`. Predtým je však potreba túto funkčnosť zapnúť príkazom `glEnable(GL_POLYGON_OFFSET_FILL)`.

Scéna z pohľadu svetla je teraz zaznamenaná v pamäti hĺbky. Odtiaľ ju potrebujeme dostať do tieňovej mapy. Vyberieme použitie hĺbkovej mapy, ktorú sme si pripravili v inicializačnej fáze. Funkciou `glCopyTexSubImage2D` potom skopírujeme obsah pamäte hĺbky do textúry predstavujúcej tieňovú mapu.

```
//Read the depth buffer into the shadow map texture
glBindTexture(GL_TEXTURE_2D, shadowMapTexture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT,
            shadowMapWidth, shadowMapHeight, 0,
            GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, NULL);
glCopyTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, 0, 0, shadowMapWidth,
            shadowMapHeight);
```

Po vykreslení scény z pohľadu svetla máme za sebou prvý vykresľovací priechod. V druhom priechode budeme už scénu vykresľovať z pohľadu pozorovateľa. Vymažeme teda pamäť hĺbky (aktuálne je pozmenená z prvého priechodu), nastavíme pohľadové teleso kamery (pozíciu, smer a projekciu kamery) a funkciou `glViewport` určíme vykresľovaciu oblasť, tentokrát na rozmery okna aplikácie.

V druhom prechode sa vykresľuje scéna celá zatienená. Použijeme teda osvetlovací model, ktorý reprezentuje zatienené oblasti, to znamená ambientnú zložku, malú časť difúznej zložky svetla a žiadnu odrazovú zložku. Vykreslíme scénu a tým končíme druhý priechod.

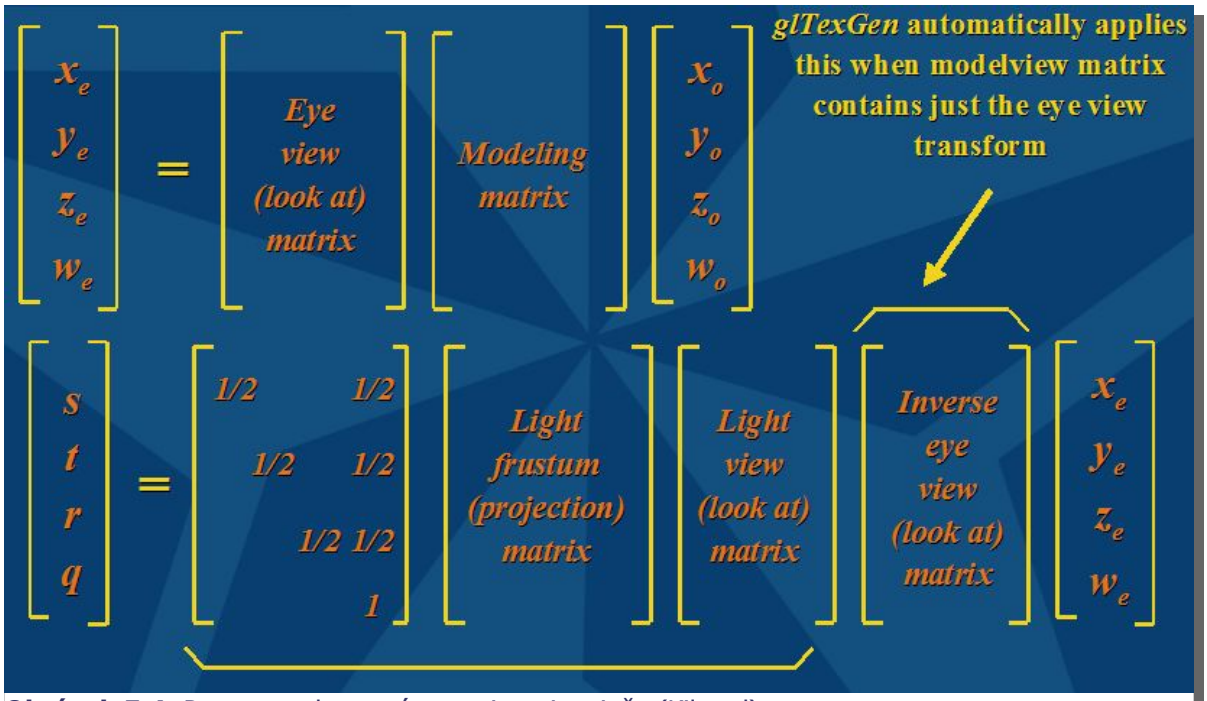
V treťom priechode budeme vykonávať test hĺbkovej mapy a na miestach, kde bude úspešný, vykreslíme osvetlené časti scény. Ešte pred testovaním hĺbkovej mapy ju však musíme správne namapovať na scénu. Využijeme na to princípy projekčného mapovania textúr. V tejto časti tiež použijeme transformačné matice pohľadu svetla, ktoré sme si uložili v prvom priechode. Na získanie homogénnych textúrovacích súradníc v priestore okna s, t, r, q použijeme vzťah:

$$[s,t,r,q]^T = S * P_{svetla} * L^{-1} * M[x_o, y_o, z_o, w_o]^T$$

Vzorec 5.1

kde S je maticou posunu (***bias matrix***), P je projekčná matica svetla, L^{-1} je pohľadová matica svetla, M je inverzia modelovej matice objektu a xo, yo, zo, wo sú homogénne súradnice objektu [28]. Súčin matice posunu, pohľadovej matice svetla a projekčnej matice svetla nám dáva tzv. maticu textúry (***texture matrix***), ktorú si ešte pred samotným generovaním súradníc musíme pripraviť. Ako dosiahneme zaradenie inverzie modelovej matice do výsledku sa dozvieme v nasledujúcom odstavci.

Textúrovacie súradnice môžeme v prostredí OpenGL automaticky generovať funkciou `glTexGen(coordinate, parameter, value)`. Parameter `coordinate` predstavuje jednu z textúrovacích súradníc (môže nadobúdať hodnoty `GL_S, GL_T, GL_R a GL_Q`). Ako druhý argument funkcie použijeme parameter `GL_TEXTURE_GEN_MODE` s hodnotou `EYE_LINEAR`, ktorý určuje, že súradnice budú generované na základe pozície oka pozorovateľa. Ak pozorovateľ zmení polohu vzhľadom na pozíciu objektu, generovanie súradníc sa taktiež zmení. K danej súradnici (***s***, ***t***, ***r*** alebo ***q***) musíme potom priradiť zodpovedajúcu rovinu pohľadu (***eye plane***) a to opätovným zavolaním `glTexGen` s parametrom `EYE_PLANE`. Hodnotou tohto parametru bude vektor, ktorý reprezentuje rovinu pohľadu. Roviny pohľadu tvoria riadky matice textúry, odtiaľ teda získame vektory, ktoré dosadíme ako hodnoty `EYE_PLANE` pre každú zo súradníc textúry.



Obrázok 5.4: Proces tvorby textúrovacej matice tieňa (Kilgard)

Vo vzťahu (5.1) vystupuje inverzná modelová matica objektu. Vynásobenie inverziou modelovej matice sa pri špecifikácii rovín pohľadu deje automaticky v hardware grafickej karty, čo nám ušetrí jednu transformáciu počítanú na CPU.

Po vytvorení matice textúry, označovanej aj ako tieňová matica (*shadow matrix*), a nastavení generovania súradníc môžeme pristúpiť k vykreslení scény a testu hĺbkovej mapy. Pri použití tieňovej mapy funkciou `glBindTexture` sa automaticky zapína aj test hĺbkovej mapy (za predpokladu, že sme porovnávanie správne nastavili v inicializačnej fáze). Ostáva teda už len nastaviť osvetlovací model pre nezatienené, osvetlené časti scény a scénu vykresliť.

Implementácia s použitím fragment programu

Implementáciou s použitím fragment programu (programovateľného zobrazovacieho reťazca, shaderu) môžeme realizovať algoritmus tieňových máp v dvoch vykresľovacích priechodoch. Prvý priechod vykresľujúci scénu z pohľadu svetla ostáva nezmenený, úpravy sa týkajú prechodu druhého. Implementácia tejto časti je realizovaná pomocou jazyku Cg.

Po nastavení pohľadového telesa kamery vypočítame maticu textúry (rovnakým spôsobom ako pri bežnom vykresľovaní vo fixnom zobrazovacom reťazci) a pošleme ju na spracovanie do GPU.

V programe spracúvajúcom vrcholy (*vertex shader*) získame súradnice tieňovej mapy vynásobením tieňovej matice (matice textúry) a aktuálnej pozície vrcholu. Súradnice sa ďalej posielajú do programu spracúvajúceho fragmenty (*fragment shader*, *pixel shader*).

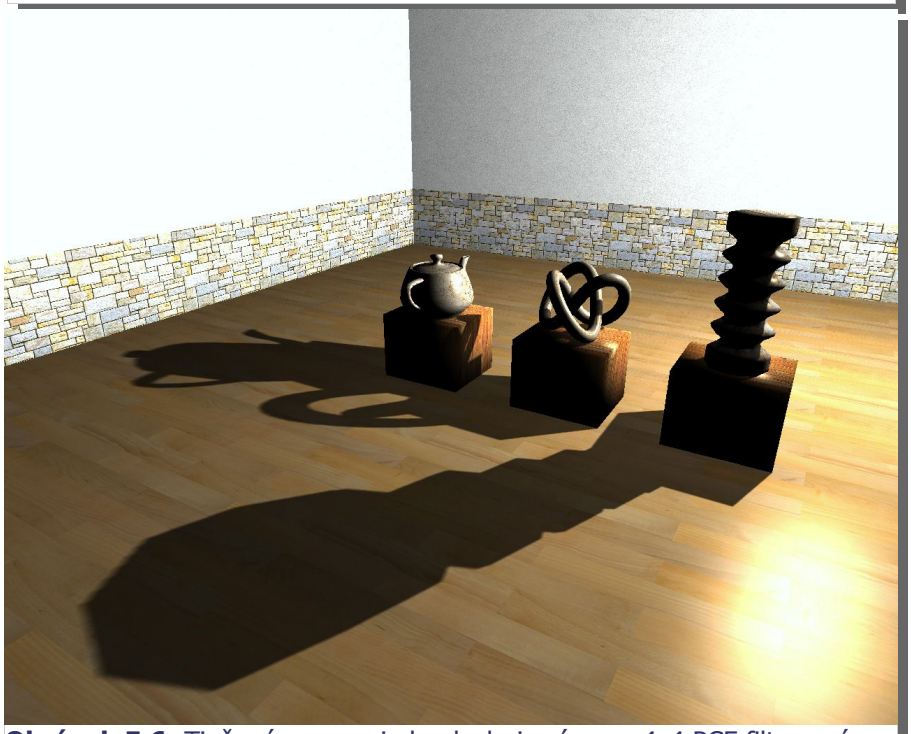
V programe spracúvajúcom fragmenty použijeme súradnice z vertex shaderu na získanie hodnoty z hĺbkovej mapy. Ak fragment programu dopredu oznámime, že budeme pracovať s tieňovou (hĺbkovou) mapou, tak sa pri získaní hodnoty z textúry (*texture look-up*) vykonáva test hĺbkovej mapy automaticky. Návratová hodnota funkcie `tex2Dproj`, ktorá má na starosti získanie hodnoty zo zadaných súradníc v textúre, potom určuje, či je daný fragment zatienený alebo nie. V prípade tieňa je to 0, v prípade osvetleného fragmentu zase 1. Na tomto mieste môžeme použiť viacej „nazretí" do textúry, teda porovnať daný fragment

s viacerými hodnotami (preskúmať okolie) v tieňovej mape a použiť to na aplikáciu PCF. Výsledok sa potom použije pri výpočte osvetlenia. V demonštračnej aplikácii sa používa vzťah na výpočet osvetlenia:

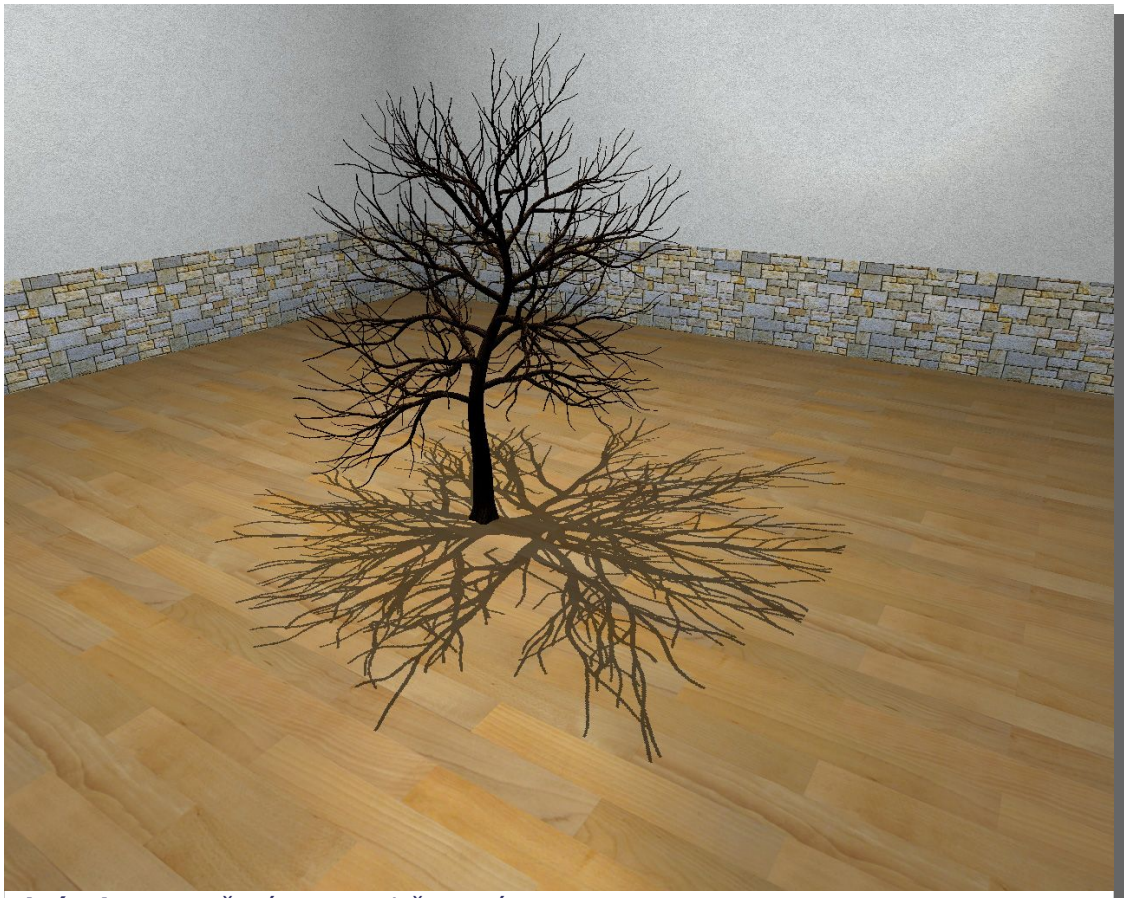
```
oColor.xyz = shadowCoeff.xyz * specular + (shadowCoeff.xyz + 0.5) *
                    diffuse + ambient + emissive
```

kde `shadowCoeff` je miera zatienenia fragmentu z intervalu <0, 1>.

Obrázok 5.5: Tieňové mapy v jednoduchej scéne

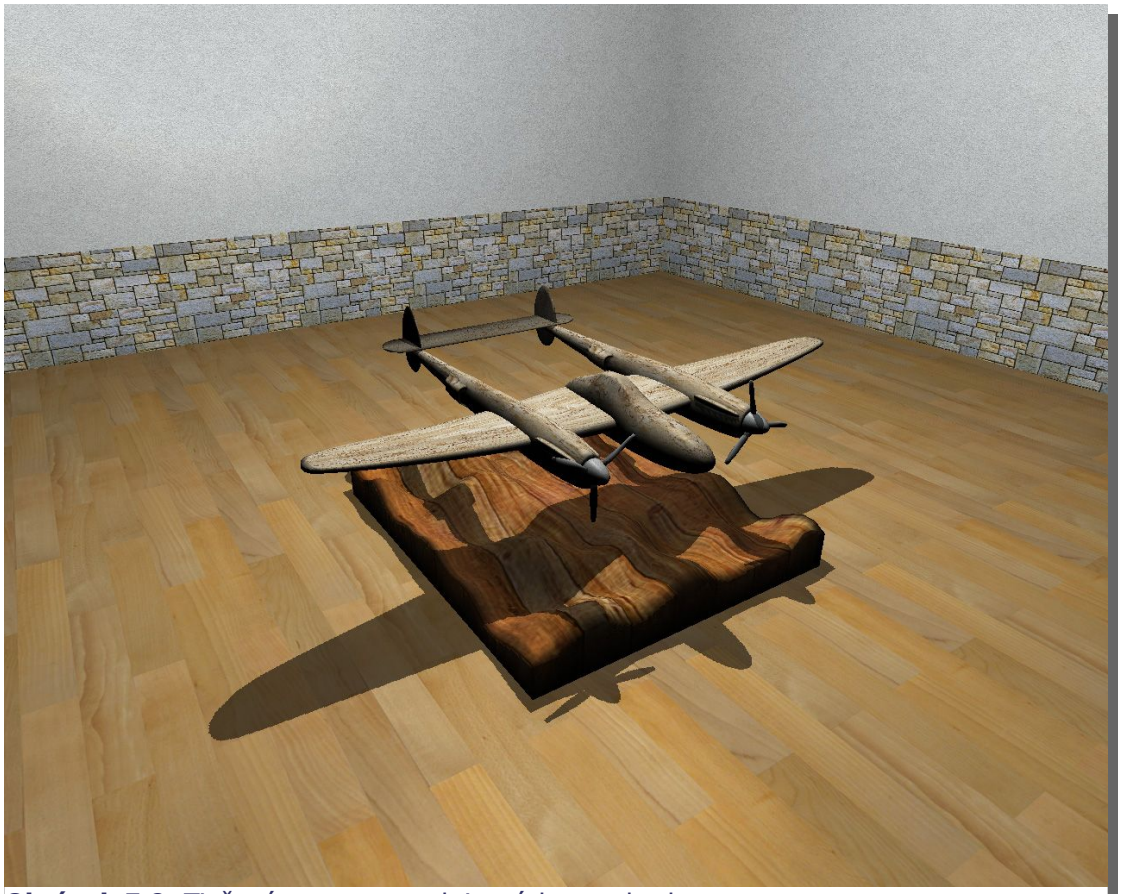


Obrázok 5.6: Tieňové mapy v jednoduchej scéne so 4x4 PCF filtrovaním

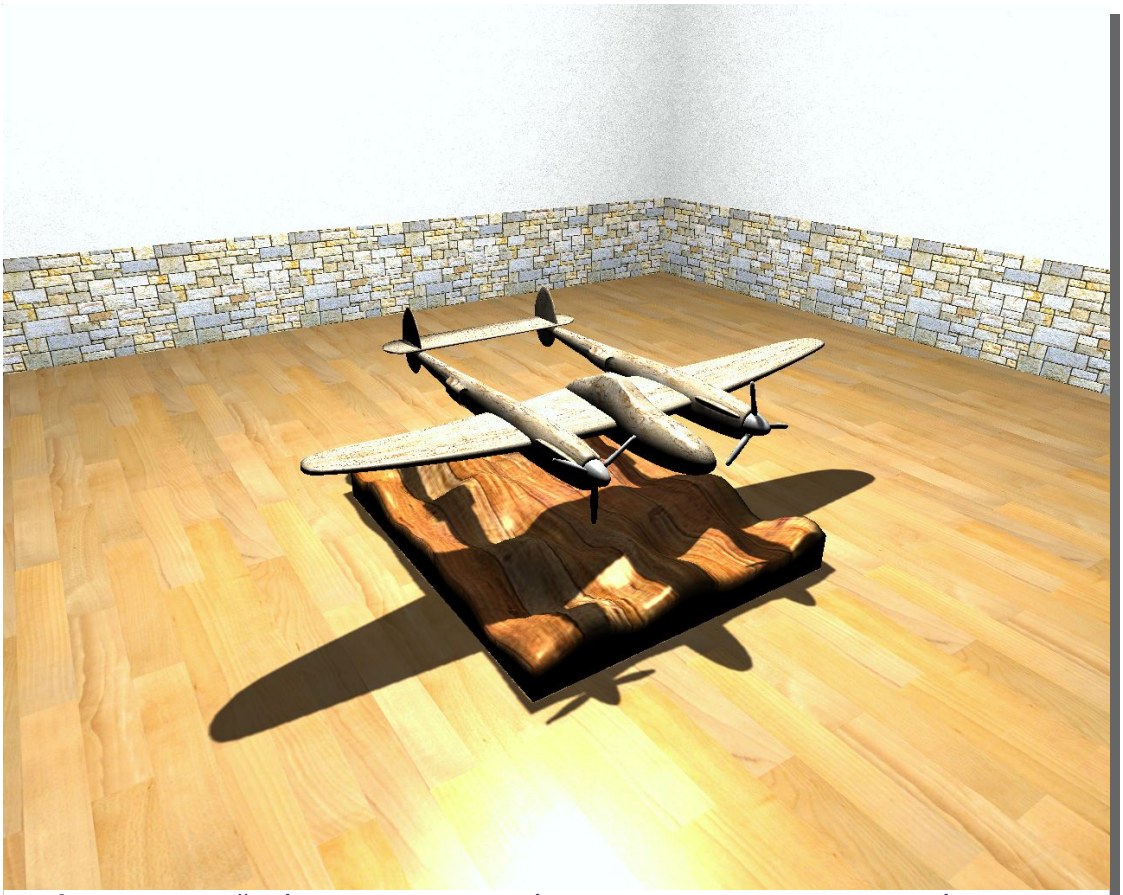


Obrázok 5.7: Tieňové mapy v zložitej scéne

Obrázok 5.8: Tieňové mapy v zložitej scéne so 4x4 PCF filtrovaním



Obrázok 5.9: Tieňové mapy v na zakrivených povrchoch



Obrázok 5.10: Tieňové mapy v na zakrivených povrchoch so 4x4 PCF filtrovaním

5.1.3 Tieňové telesá

Implementácia algoritmu tieňových telies má tri hlavné úlohy – nájdenie siluety, vytvorenie tieňových telies a riešenie viditeľnosti. Prvé dve z nich pracujú s geometrickými dátami objektu, preto je dobré si pred vlastnou implementácia algoritmu pripraviť štruktúru dát objektu (modelu), ktorý má byť pôvodcom tieňa. Ukážeme si, ako by mohol takýto model vyzerať.

```
// Face structure
struct Face
{
      unsigned int p[3];     // indices which make this face
      int neighbour[3];      // indices of neighbour face
      M3DVector4f PlaneEq;  // plane equation - calculate
      bool visible;          // is face visible from light?
};
```

Základom je štruktúra `Face`, ktorá predstavuje plochu polygónu. Každý polygón má dve plochy, jednu privrátenú (otočenú k pozorovateľovi) a jednu odvrátenú (otočenú od pozorovateľa). Odvrátené plochy bývajú najčastejšie skryté a nevykresľujú sa, privrátené sú naopak tie, ktoré sú pre pozorovateľa viditeľné a tvoria tvar objektu.

Štruktúra plochy polygónu obsahuje vrcholy, z ktorých sa polygón skladá, resp. indexy do poľa vrcholov. Pracujeme s trojuholníkmi, takže to bude pole troch indexov. Uchovávať indexy je šetrnejšie na pamäť ako uchovávať súradnice vrcholov, pretože jeden vrchol môže byť obsiahnutý vo viacerých polygónoch a uchovávali by sme tak redundantnú informáciu.

Ďalším poľom, ktoré štruktúra `Face` obsahuje, je pole, ktoré obsahuje indexy susedných plôch polygónov. Táto informácia bude dôležitá pri výpočte siluety modelu.

`PlaneEq` je vektorom o štyroch zložkách, ktorý obsahuje parametre všeobecnej rovnice roviny a, b, c, d. Vypočítava sa počas inicializácie modelu.

Poslednou zložkou štruktúry `Face` je booleanovská premenná `visible`. Tá určuje, či je daná plocha privrátená alebo odvrátená od svetla. Jej hodnota sa vypočítava na začiatku algoritmu.

Štruktúra `Face` bude ďalej jednou zo zložiek hierarchicky vyššej štruktúry (triedy) `Model`.

```
// Model class
class Model
{
private:
      int nPoints;
      int nFaces;

      M3DVector3f *points;
      M3DVector3f *normals;
```

```
        Face *faces;

        M3DVector3f position;
        M3DVector3f rotation;
        M3DVector4f local_light_position;

        // Sets connectivity info for finding silhouette
        void SetConnectivity();
        void CalculateNormals();

public:
        Model();
        ~Model();
        void Draw();
        void Init();
        // Calculate local light position
        void CalcLocalLP(float LightPosition[4]);
        // Casts stencil shadow volume
        void CastStencilShadow(float factor, float offset);
};
```

Trieda `Model` reprezentuje 3D model objektu. Obsahuje pole vrcholov, normál, plôch a môže obsahovať ďalšie potrebné informácie o modeli ako textúrovacie súradnice, jeho pozíciu v scéne, natočenie a pod.

Pole vrcholov pozostáva z dynamického poľa trojzložkových vektorov pre uloženie troch súradníc vrcholu ***x, y, z***. To isté by sa dalo povedať o poli normál.

Jednotlivé plochy polygónov, z ktorých sa objekt skladá, sú uložené v poli `faces`, ktorého typom je vyššie popísaná štruktúra Face.

Pri hľadaní siluety, resp. pri určovaní orientácie plochy vzhľadom na svetelný zdroj sa využíva lokálna pozícia svetla, to znamená, pozícia svetla v objektovom priestore. Vypočítava sa v metóde `CalcLocalLP` a ukladá do štvorzložkového vektoru `local_light_position`.

Metóda triedy `Model SetConnectivity` prechádza pole plôch polygónov a rieši susednosť jednotlivých plôch. Napĺňa tým pole indexov `neighbour` v štruktúre `Face`.

Hlavné ťažisko algoritmu tieňových máp je potom riešené v metóde `CastStencilShadow`. Tu prebieha tvorba tieňových telies a zápis do pamäti šablóny (stencilu). Metóda má dve hlavné časti, hovoríme aj, že má dva priechody. Jedna časť spracováva predné plochy polygónov (*front faces*) tieňového telesa, druhá potom plochy odvrátené (*back faces*).

Ešte pred nimi sa však vypočíta orientácia plôch polygónov vzhľadom na svetelný zdroj. Viditeľnosti plochy zo svetla sa vypočíta pomocou skalárneho súčinu lokálnej pozície svetla (`local_light_position`) s rovnicou roviny (`PlaneEq`). Podľa znamienka výsledku

sa určí, či je plocha otočená ku svetlu (výsledok je kladný) alebo je od svetla odvrátená (výsledok je záporný).

Nasleduje tvorba tieňového telesa. Tieňové teleso budeme vykreslovať iba do pamäti šablóny, preto musíme na začiatku metódy vypnúť zápis do pamäti farby, taktiež vypnúť zápis do hĺbkovej pamäti, zapnúť maskovanie (*culling*) plôch polygónov, vypnúť výpočet osvetlenia, a povoliť zápis do pamäti šablóny. Aj keď je zápis do hĺbkovej pamäti vypnutý, samotný hĺbkový test musí ostať zapnutý, pretože sa používa pri rozhodovaní o zápise do pamäti šablóny.

```
glDisable(GL_LIGHTING);
glShadeModel(GL_FLAT);
glDepthMask(GL_FALSE);
glDepthFunc(GL_LESS);
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
glEnable(GL_STENCIL_TEST);
glEnable(GL_CULL_FACE);
```

V prvom priechode sa hodnota v pamäti šablóny sa zvyšuje a podľa typu algoritmu, depth-pass alebo depth-fail, sa spracúvajú predné, resp. zadné plochy polygónov. Podľa toho nastavíme maskovanie na `glCullFace(GL_BACK)` v prípade depth-pass, resp. `glCullFace(GL_FRONT)` v prípade depth-fail. Podľa použitého algoritmu sa tiež rozhodneme, kedy budeme zapisovať do pamäti šablóny. Použijeme `glStencilOp(GL_KEEP, GL_KEEP, GL_INCR)` pre depth-pass a `glStencilOp(GL_KEEP, GL_INCR, GL_KEEP)` pre depth-fail. V prvom prípade sa teda bude hodnota v šablóne zvyšovať, ak uspeje hĺbkový test viditeľnosti pre predné plochy. V druhom prípade sa bude hodnota zvyšovať, ak hĺbkový test neuspeje pre odvrátené plochy.

```
// FIRST PASS, stencil operation increases stencil value
if (zfail)
{
	glCullFace(GL_FRONT);
	glStencilFunc(GL_ALWAYS, 0x0, 0xff);
	glStencilOp(GL_KEEP, GL_INCR, GL_KEEP);
}
else
//z-pass
{
	glCullFace(GL_BACK);
	glStencilFunc(GL_ALWAYS, 0x0, 0xff);
	glStencilOp(GL_KEEP, GL_KEEP, GL_INCR);
}
```

Nasledujúci krok je relevantný iba pre algoritmus depth-fail. Jedná sa o tvorbu a vykreslenie uzáverov. Predný uzáver (*front cap, light cap*) pozostáva z privrátených plôch

polygónov objektu, ktoré sú otočené ku svetlu. Prejdeme teda všetky takéto plochy objektu a vykreslíme do pamäti šablóny ich kópie, ktoré predstavujú predné uzávery. Zadný uzáver je zase tvorený odvrátenými plochami polygónov objektu, ktoré sú viditeľné zo svetla. Predĺžime ich pozdĺž steny tieňového telesa a vykreslíme do pamäti šablóny.

Spoločným krokom obidvoch algoritmov je nájdenie a vykreslenie stien tieňového telesa do pamäti šablóny. Steny tieňového telesa majú svoj počiatok v obrysových hranách objektu videného zo svetla. Je teda treba nájsť siluetu objektu. Budeme prechádzať plochy polygónov, ktoré sú privrátené ku svetlu. To, že sa jedná o siluetnú hranu zistíme tak, že susedný polygón, resp. jeho plocha, je odvrátená od svetla, alebo že polygón na danej hrane nemá žiadneho suseda. Ak zistíme, že sa jedná o siluetnú hranu, môžeme začať vykreslovať jednu zo stien tieňového telesa. Stena je tvorená obdĺžnikom, dva jeho body už poznáme, sú to body, ktoré vytvárajú siluetnú hranu. Ďalšie dva body získame predĺžením týchto bodov v smere vektora, ktorý je určený pozíciou svetla a jedným z bodov siluetnej hrany. Prechodom cez všetky siluetné hrany takto zhotovíme všetky steny tieňového telesa a vykreslíme ich do pamäti šablóny, čím končí prvý priechod algoritmu.

V druhom priechode sa spracúvajú zadné plochy polygónov v prípade algoritmu depth-pass a predné plochy v prípade algoritmu depth-fail. Hodnota v pamäti šablóny sa znižuje.

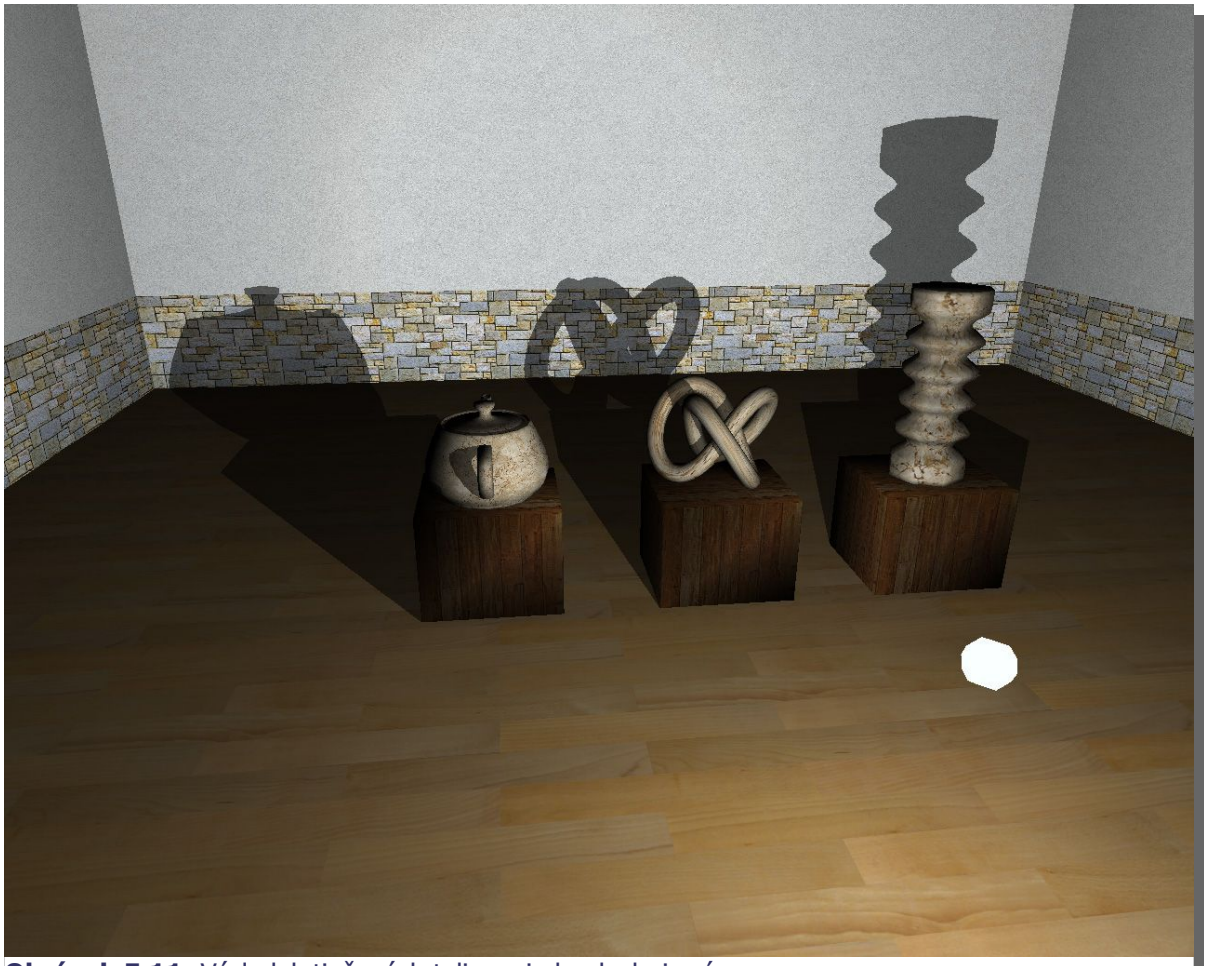
```
// SECOND PASS, stencil operation decreases stencil value
if (zfail)
{
      glCullFace(GL_BACK);
      glStencilFunc(GL_ALWAYS, 0x0, 0xff);
      glStencilOp(GL_KEEP, GL_DECR, GL_KEEP);
}
else
//z-pass
{
      glCullFace(GL_FRONT);
      glStencilFunc(GL_ALWAYS, 0x0, 0xff);
      glStencilOp(GL_KEEP, GL_KEEP, GL_DECR);
}
```

Okrem toho sa druhý priechod od prvého v ničom nelíši. Po jeho skončení máme v pamäti šablóny zaznamenané oblasti, ktoré ležia v tieni.

Ostáva nám už len finálne vykreslenie scény s vrhnutými tieňmi. Ešte pred výpočtom tieňových telies vykreslíme celú scénu ako zatienenú. Potom pre všetky objekty vrhajúce tieň zavoláme funkciu `CastStencilShadow`, čím dosiahneme vyznačenie tieňov v pamäti šablóny. Zatienené oblasti majú nenulovú hodnotu šablóny, osvetlenú scénu budeme teda vykresľovať na miestach s nulovou hodnotou šablóny.

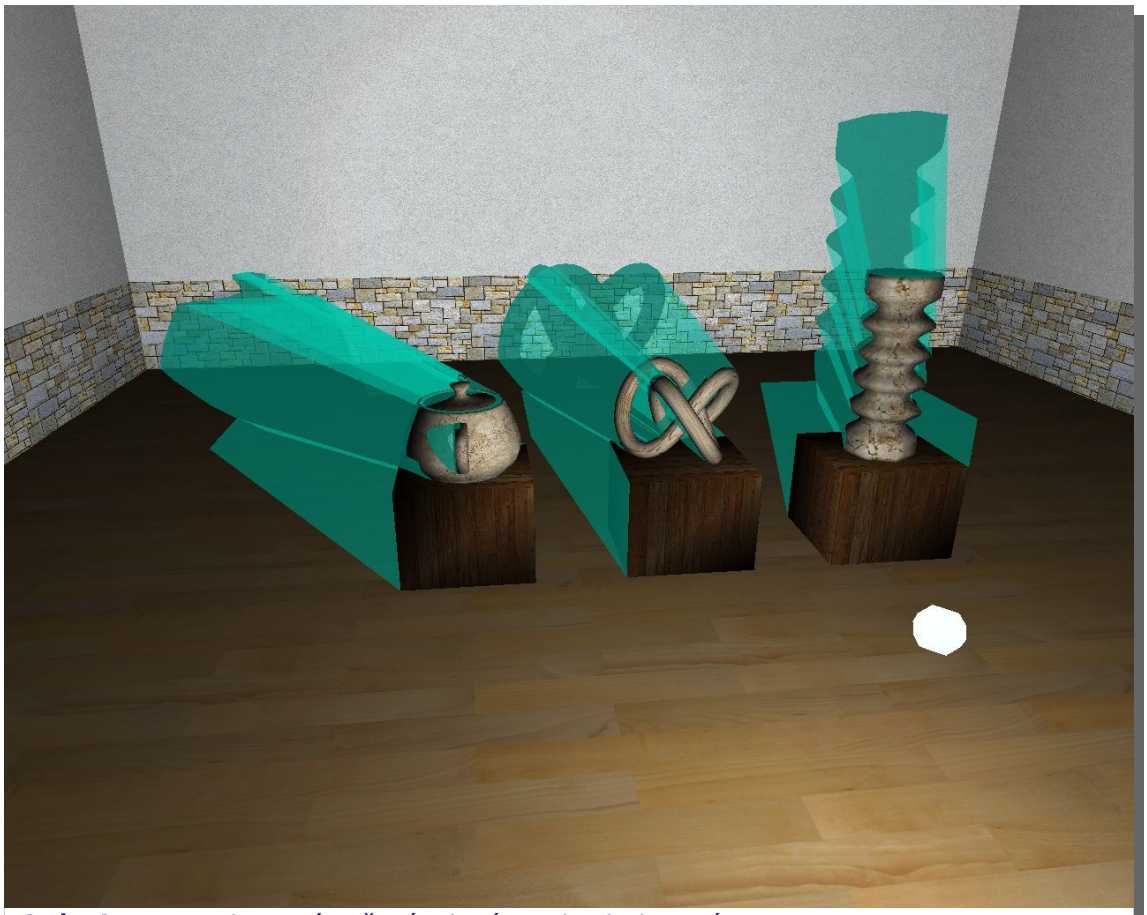
```
// lighting pass
glEnable(GL_LIGHTING);
```

```
glLightfv(light->id, GL_SPECULAR, light->specular);
glLightfv(light->id, GL_DIFFUSE, light->diffuse);

glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glDepthFunc(GL_EQUAL);
glStencilFunc(GL_EQUAL, 0, ~0);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

DrawScene();

glDepthMask(GL_TRUE);
glDepthFunc(GL_LEQUAL);
glStencilFunc(GL_ALWAYS, 0, ~0);
```



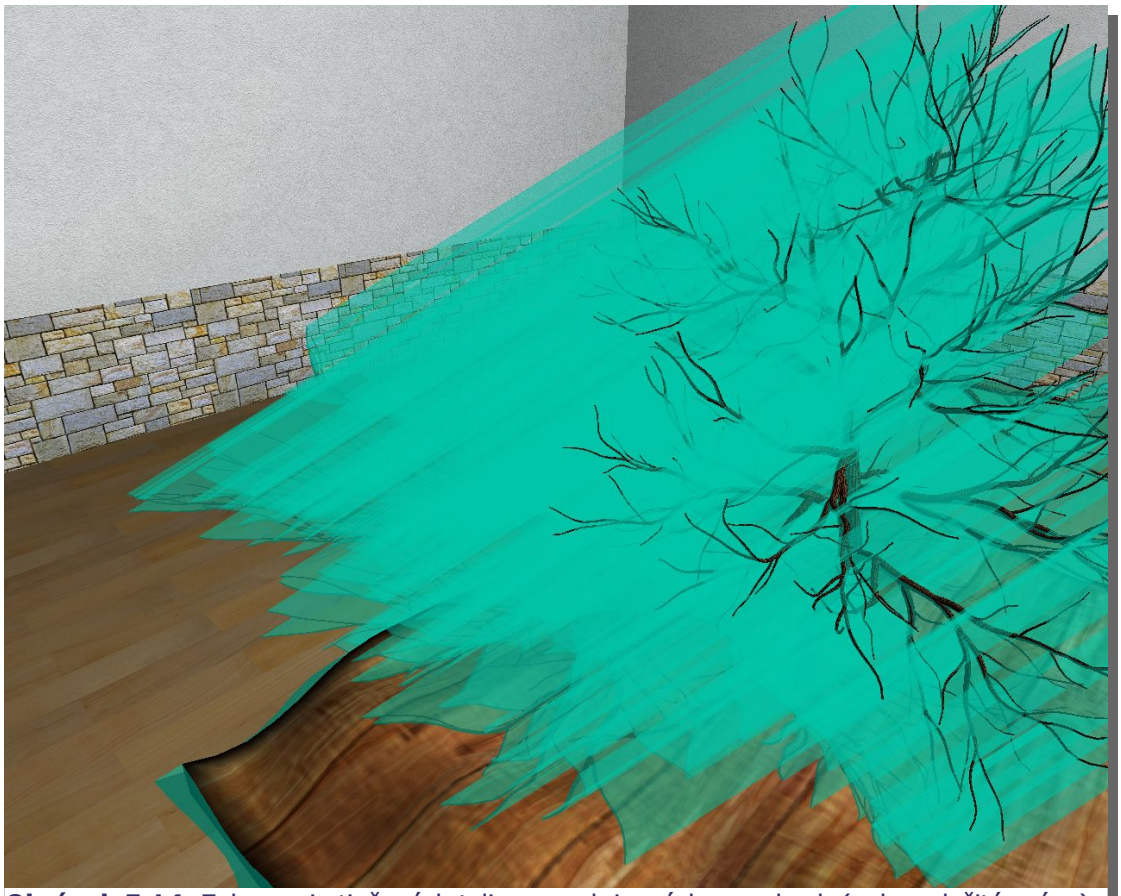
Obrázok 5.11: Výsledok tieňových telies v jednoduchej scéne

Nakoniec ešte nastavíme niektoré dôležité stavy, aby bolo možné bezproblémovo vykresliť ďalší snímok. Zapneme zapisovanie do pamäti hĺbky a nastavíme funkciu hĺbkového a šablónového testu.

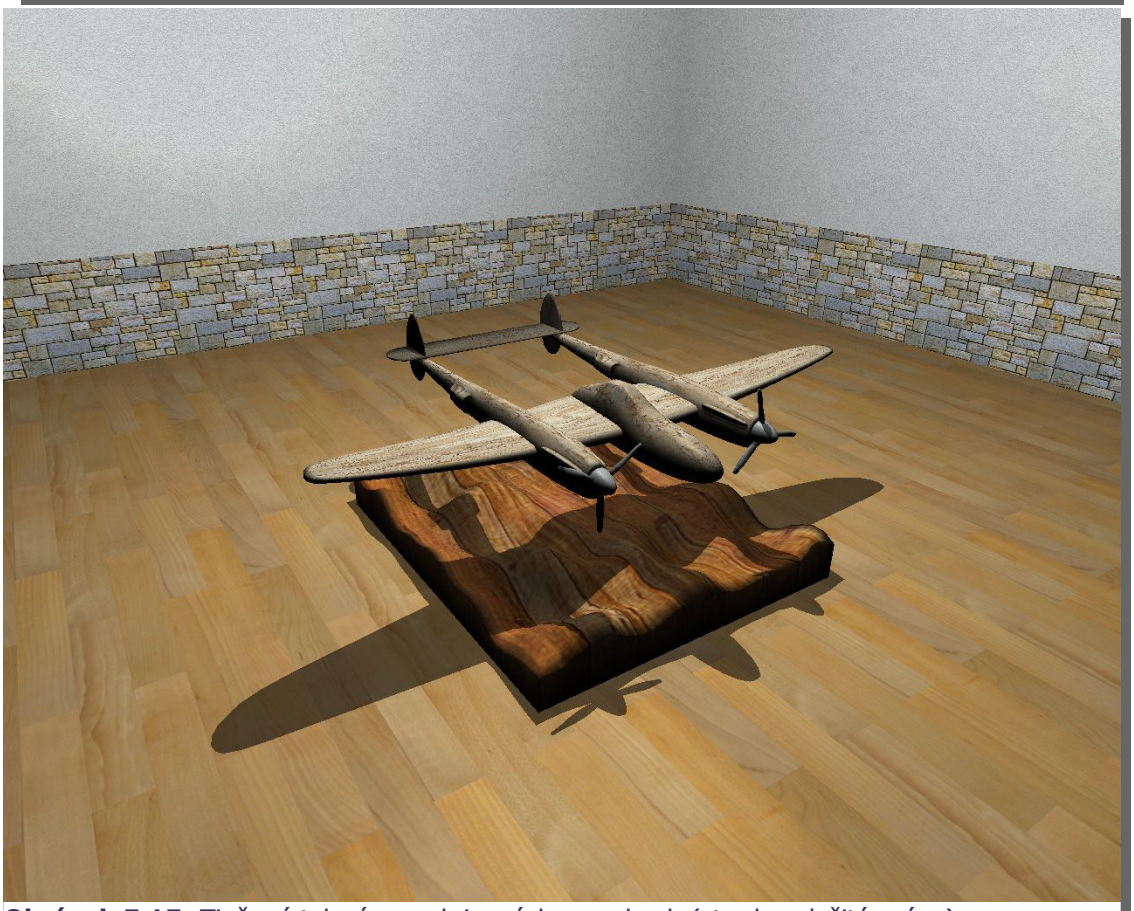


Obrázok 5.12: Zobrazené tieňové teliesá v jednoduchej scéne

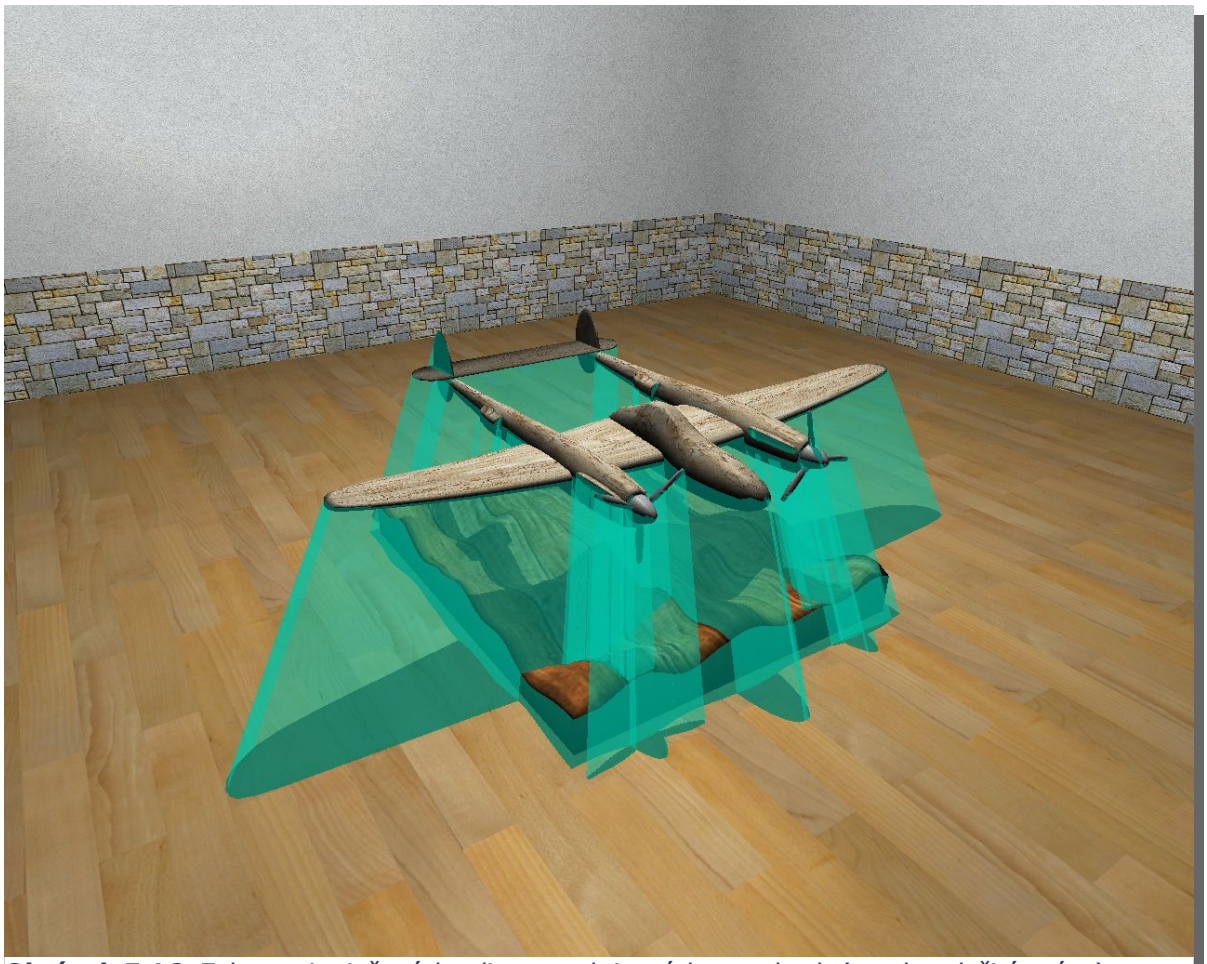
Obrázok 5.13: Tieňové telesá na zakrivených povrchoch (polygonálne zložitá scéna)



Obrázok 5.14: Zobrazenie tieňových telies na zakrivených povrchoch (polyg. zložitá scéna)



Obrázok 5.15: Tieňové telesá na zakrivených povrchoch (stredne zložitá scéna)



Obrázok 5.16: Zobrazenie tieňových telies na zakrivených povrchoch (stredne zložitá scéna)

5.2 Porovnanie vybraných techník

V tejto časti sa budeme zaoberať výhodami a nevýhodami jednej techniky voči druhej. Zhodnotíme aplikáciu techník na konkrétne modelové situácie. Budeme sa zaoberať možnosťou rozšírenia algoritmu, obtiažnosťou jeho implementácie, závažnosťou problémov algoritmu a jeho výpočetnou náročnosťou. Porovnanie sa bude týkať kvality výsledkov, schopnosťou vysporadúvať sa s rôznymi situáciami vo scéne, robustnosťou algoritmov, vlastnosťami a možnosťami jednotlivých techník. Medzi vlastnosti budeme počítať napr. schopnosť vrhať vlastné tiene a charakter vrhaných tieňov (či interagujú s ostatnými objektmi v scéne). Medzi možnosti bude patriť zase možnosť rozšírenia na mäkké tiene alebo napr. možnosť urýchliť výpočet presunutím určitých úloh na GPU. Jednotlivé testy kvality a výkonnosti sa nachádzajú v prílohe 2.

5.2.1 Rovinné tiene

Kvalita

Ak je pozorovateľ od sledovaného objektu dostatočne vzdialený, nevšimne si chýbajúcich vlastných tieňov a naopak je dôležitá informácia o tieni, ktorý vrhá objekt pod seba, čiže na zem. Podľa týchto tieňov sa vieme najlepšie zorientovať v scéne a určiť polohu,

ale aj tvar objektu. Zakrivenie tieňov je taktiež dôležité, ale iba pri detailnejšom pohľade, kedy pozorovateľovi dáva aj informáciu o okolí. K použiteľnosti tejto techniky prispieva aj fakt, že tiene sú vždy geometricky presné a ich okraje sú kontinuálne a ostré.

Výpočetná náročnosť

Jednoduchosť algoritmu sa prejavila na jeho malej výpočetnej náročnosti. Vytvorenie matice tieňa je otázkou pár vektorových a skalárnych súčinov a realizuje sa iba raz pre každú tieňovú rovinu. Ďalej sa prevedie transformácia vybraného objektu maticou tieňa, čo nie je nič iné ako maticové násobenie s modelovo-pohľadovou maticou objektu. Vykreslenie tieňa potom pozostáva vo vykreslení neosvetlenej, neotextúrovanej, transformovanej verzie objektu. Závisí teda na počte polygónov daného objektu, ktorý tieň vrhá, z koľkých polygónov sa skladal pôvodný objekt, z toľkých sa bude skladať aj jeho tieň. Náročnosť algoritmu je teda lineárna. Nutno ešte dodať, že tieň sa vykresľuje v drvivej väčšine prípadov rýchlejšie ako samotný objekt, a to kvôli už spomínanej absencii osvetlenia a textúrovacej informácie.

Výhody a nevýhody

Hlavnou nevýhodou tejto techniky je samozrejme fakt, že dokáže zobrazovať tiene iba v rovine, zatiaľčo ostatné techniky produkujú tiene aj na zakrivených povrchoch. Druhou veľkou nevýhodou je neschopnosť vrhať vlastné tiene, ktoré taktiež ostatné dve techniky zvládajú.

Použitie

Hoci je technika rovinných tieňov na prvý pohľad na nižšej kvalitatívnej úrovni, ako ostatné dve techniky, existuje pre ňu veľmi užitočné použitie. V real-time grafických 3D aplikáciách často riešime problém vrhania tieňov na rovinné plochy. Modelovou situáciou nech je scéna budov alebo osôb vrhajúcich tiene na zem. Skoro v každom realistickom zobrazení nájdeme prípady rovinných tieňov. Vďaka tomu, že sa jedná o mimoriadne rýchlu techniku, uplatňuje sa táto technika v situáciách, ktoré nie sú vizuálne dôležité, a na ktorých sa dá tým pádom ušetriť vykresľovací čas na iné vizuálne efekty.

Implementácia a odstraňovanie problémov

Problém double blendingu sa dá jednoducho vyriešiť pomocou buffera šablóny. Buffer šablóny je bežnou súčasťou každého grafického akcelerátoru, takže je toto riešenie bezproblémové. Riešenie problému s nepravými tieňmi navrhli Michael Herf a Paul Heckbert [6]. Je potreba pozmeniť stávajúcu transformáciu, ktorá zobrazuje objekt z 3D do 2D, na transformáciu z 3D do 3D a to tak, že táto transformácia zobrazí telesá spôsobujúce nepravé tiene mimo pohľadového priestoru. Telesá mimo pohľadového priestoru potom nebudú vykreslené a teda ani nedôjde k vykresleniu ich nepravých tieňov.

Možnosti rozšírenia

Rozšíriteľnosť nedosahuje úrovni ostatných dvoch techník. Napriek tomu však pár vylepšení existuje. Dokonca jedna z techník produkujúcich mäkké tiene využíva princíp rovinných tieňov [31].

Prehľad vlastností

- ✔ rýchlosť (výpočetná nenáročnosť)
- ✔ jednoduchosť
- ✔ geometrická presnosť
- ✔ využiteľnosť

- ✗ neschopnosť vrhať vlastné tiene
- ✗ neschopnosť vrhať tiene na zakrivené plochy

5.2.2 Tieňové mapy

Kvalita

Kvalita tieňov pri tejto technike je závislá na viacerých okolnostiach. Vo veľkej miere závisí na rozlíšení použitej tieňovej mapy a na pohľadovom telese svetla pri tvorbe tieňovej mapy. Tieto faktory vplývajú na aliasing tieňovej mapy, ktorý spôsobuje vizuálne nežiadúce a rušivé „zubaté" okraje tieňov. Takéto tiene nie sú potom geometricky presné. Čím menšie je rozlíšenie tieňovej mapy, tým je aliasing horší. Okrem toho tu máme problémy s vlastnými tieňmi, pri ktorých môže vznikať povrchové akné (*surface acne*).

Kvalitou teda táto technika zaostáva za kvalitou techniky tieňových telies, kde žiadny aliasing nenastáva. Situácia sa však zlepšuje s použitím tieňovej mapy s vyšším rozlíšením, ktoré môžeme dosiahnuť buď prostým zvýšením rozmerov mapy, alebo vďaka niektorému z rozšírení. Pri dostatočnom rozlíšení môžeme dosiahnuť rovnaké výsledky ako pri tieňových telesách. Pri použití PCF sa dokonca karta obracia v prospech tieňových máp. Prechody na okrajoch tieňov sú hladké a dokonca aj pri nižších rozlíšeniach tieňovej mapy môžeme dosiahnuť dobre vyzerajúce výsledky, napodobujúce mäkké tiene.

Výpočetná náročnosť

Oproti dvom ostatným technikám, pracujú tieňové mapy v obrazovom priestore, čo im dáva výhodu v nezávislosti na geometrii objektu. Dosahujú tak veľmi dobré výsledky aj pri komplexných scénach s veľkým počtom polygónov. Výpočetná náročnosť stúpa s rastúcim rozlíšením tieňovej mapy, ktoré ovplyvňuje aj pamäťovú náročnosť. Čím väčšia tieňová mapa, tým viac obsadenej pamäte na grafickej karte. Na výkon má veľký vplyv aj zvyšovanie počtu vzoriek, ktoré sa používajú v teste hĺbkovej mapy pri PCF.

Výhody a nevýhody

Výhoda tieňových máp tkvie hlavne v nezávislosti techniky na geometrii objektu, s čím je spojená dobrá výkonnosť pri polygonálne zložitých objektoch. Plusom je určite aj modifikovateľnosť algoritmu a s tým spojené množstvo rozšírení, riešiacich problémy s aliasingom, obohacujúcich funkčnosť a tým pádom aj rozširujúcich použiteľnosť.

Nevýhodou naopak je, že v základnej verzii neposkytujú tieňové mapy príliš použiteľné riešenie. Je potreba pomerne veľa vecí upraviť a odladiť aby sme získali vizuálne kvalitné výsledky. Techniku sprevádzajú neduhy s aliasingom, či už spôsobené nízkym rozlíšením tieňových máp alebo nepresnosťou hĺbkovej pamäti. Objekty vrhajúce tiene sa musia nachádzať vnútri pohľadového telesa svetla. Ak chceme čo najpresnejšie rozdelenie hĺbkovej

pamäti, musíme zaistiť aj optimálny tvar tohto telesa. Problémy sa vyskytujú aj pri použití všesmerových bodových svetiel, kedy je treba situáciu riešiť netriviálnym rozšírením.

Použitie

Čím ďalej tým viac sa technika tieňových máp a jej modifikácie používajú v moderných počítačových hrách. Tie predstavujú zobrazovacie systémy pracujúce v reálnom čase s vysoko komplexnými scénami, a presne na takéto situácie je algoritmus tieňových máp najvhodnejší. Dokáže vykresľovať tiene pre veľké priestranstvá s množstvom objektov s pomerne malou výpočetnou náročnosťou (záleží samozrejme na kvalite tieňov). Príkladom môže byť osídlená krajina, zaplnená mestská časť alebo hustý les.

Implementácia a odstraňovanie problémov

Implementáciu tejto techniky by som hodnotil ako najobtiažnejšiu. Keďže jej princíp nie je založený na geometrických poznatkoch, je zo začiatku trochu ťažšia na pochopenie. Vyžaduje dobré znalosti o hĺbkovej pamäti grafickej karty a jej vzorkovaní, o prevode súradníc medzi súradnicovými systémami, je treba ovládať techniku projekčného mapovania a nakoniec znalosť prostredia grafickej knižnice a jej rozšírení.

Problémov pri implementácii je viac než dosť, ich riešenie nie je vždy úplne triviálne a vyžaduje znalosti z vyššie spomínaných oblastí. Nakoniec sa však dá každý vážnejší problém odstrániť a dosiahnuť tak kvalitných výsledkov.

Možnosti rozšírenia

Tieňové mapy majú medzi vybranými technikami najlepšiu rozšíriteľnosť. Algoritmus je veľmi dobre modifikovateľný a vďaka tomu vzniklo množstvo vylepšení rozširujúcich funkčnosť. Okrem spomenutých rozšírení v časti 3.5, existujú aj metódy na produkovanie mäkkých tieňových máp [32].

Prehľad vlastností

- ✔ výkonnosť pri polygonálne zložitých scénach
- ✔ kvalita (vlastné tiene, PCF filtrovanie)
- ✔ rozšíriteľnosť
- ✔ použiteľnosť v moderných zobrazovacích systémoch (počítačových hrách)

- ✗ kvalita (bez rozšírení)
- ✗ perspektívny a projekčný aliasing
- ✗ aliasing vlastných tieňov
- ✗ náročné na implementáciu

5.2.3 Tieňové telesá

Priamym „súperom“ techniky tieňových máp je metóda tieňových telies. Obidve patria k najpoužívanejším technikám počítačovej grafiky v zobrazovacích systémoch pracujúcich v reálnom čase. Majú kvalitatívne veľa spoločného, ale zároveň sa v mnohých aspektoch líšia.

Kvalita

Kvalita tieňov vytvorených algoritmom tieňových telies je veľmi dobrá. Tiene sú geometricky vždy presné, narozdiel od techniky tieňových máp. Ich okraje sú hladké a spojité, netrpia aliasingom ako tieňové mapy. Objekty správne zatieňujú jeden druhého, tiene sa správne ohýbajú aj na zakrivených povrchoch a bez problémov sú aj vlastné tiene. Technika produkuje ostré tiene, rozšíriť jej možnosti na produkovanie mäkkých tieňov nie je také jednoduché, ale je to možné.

Výpočetná náročnosť

Čo sa týka výpočetnej náročnosti je značný rozdiel, či sa použije depth-pass alebo depth-fail varianta algoritmu. Depth-fail algoritmus je vďaka nutnosti uzatvárať tieňové telesá prednými a zadnými uzávermi zložitejší a tým pádom pomalší. Predný a zadný uzáver sú z polygonálneho hľadiska časti objektu, ktoré sú osvetlené. Táto časť geometrie sa teda musí dva krát vykresľovať (do pamäti šablóny) naviac oproti depth-pass algoritmu. Pri objektoch s komplexnejšou geometriou to môže mať zásadný vplyv na celkový výpočetný čas. Náročnosť s komplexnejšími objektmi samozrejme narastá aj v prípade algoritmu depth-pass, avšak nie až tak podstatne. Existuje viacero rozšírení, ktoré upravujú vo väčšej, či menšej miere výkonnosť techniky [23].

Výhody a nevýhody

Veľkou výhodou techniky tieňových telies s použitím algoritmu depth-fail je jej robustnosť. Dostávame tak techniku, ktorá bude vo všetkých situáciách (poloha kamery, objektu, svetla, ...) pracovať správne a podávať kvalitné výsledky. Oproti tieňovým mapám jej napr. nerobí problém zobraziť tiene v miestnosti s bodovým svetlom v strede. Výhodou je už aj sama kvalita tieňov, musíme si ale vystačiť s ostrými tieňmi. Ak by sme chceli okraje tieňov „rozmazať" ako v prípade tieňových máp, nenájdeme žiadne jednoduché a výpočetne nenáročné riešenie.

Nevýhodami techniky tieňových telies je jej závislosť na geometrickej informácii. S narastajúcim počtom polygónov objektu sa stáva technika kvôli zvýšenej výpočetnej náročnosti menej použiteľná. Technika nedokáže správne vypočítať tiene pre *non-manifold* (nevyrobiteľné) objekty a objekty s rastrovou priesvitnosťou (*sprite*, *billboard*). V týchto prípadoch majú tieňové mapy jasne navrch.

Použitie

Technika tieňových telies nájde použitie v zobrazovacích systémoch pracujúcich v reálnom čase, ktoré vyžadujú presnosť a kvalitu zobrazenia ale zároveň nepracujú s príliš geometricky komplexnými objektmi. Ako príklad modelu zobrazovacieho systému vhodného na použitie techniky tieňových telies by som uviedol prehliadač modelov, jednoduchšiu počítačovú hru alebo zobrazovací systém virtuálnej reality. Jej uplatnenie však môže byť aj v zložitejších zobrazovacích systémoch, a to ako čiastočné riešenie výpočtu tieňov, napr. pre bodové zdroje. Príklad môže byť moderná počítačová hra, ktorá na zobrazenie polygonálne bohatších objektov, rastlín a podobných záležitostí používa algoritmus tieňových máp, ale pre bodové svetelné zdroje využíva výhod tieňových telies.

Implementácia a odstraňovanie problémov

Náročnosť implementácie by som hodnotil ako strednú. Na rozdiel od algoritmu tieňových máp musíme mať geometrické informácie o objekte uložené v dátovej štruktúre. Tá ale býva v real-time systémoch bežnou súčasťou, takže to neprináša vážnejšie problémy.

S väčšinou problémov sa vysporiada úprava algoritmu depth-fail. Pre úplnú správnosť zobrazovaných tieňov a robustnosť algoritmu je treba posunúť zadnú orezávaciu rovinu kamery do nekonečna. Pre praktické použitie je taktiež vhodne nastaviť osvetlovací model (hlavne jeho ambientnú zložku), aby zatienené oblasti nezasahovali do osvetlených.

Možnosti rozšírenia

Rozšíriteľnosť algoritmu je celkom dobrá. Viacero rozšírení vylepšuje výkonnosť a znižuje výpočetnú náročnosť [23]. Určitá časť výpočtu sa dá preniesť na GPU (vertex shader). Existuje taktiež rozšírenie na výpočet mäkkých tieňov [33].

Prehľad vlastností

Všeobecne

- ✔ vlastné tiene
- ✔ zakrivené tiene na zakrivených povrchoch
- ✔ geometrická presnosť

- ✗ závislosť na geometrii objektu
- ✗ nevhodný na zložité, non-manifold alebo priesvitné objekty

Depth-pass

- ✔ nepotrebuje uzávery tieňových telies
- ✔ vykresľuje menej geometrie
- ✔ jednoduchší na implementáciu
- ✔ rýchlejší z dvojice depth-pass, depth-fail
- ✔ nepotrebuje posunúť zadnú orezávaciu rovinu do nekonečna

- ✗ nesprávne zobrazenie tieňov v prípade zatienenej kamery
- ✗ nie je robustný

Depth-fail

- ✔ robustnosť

- ✗ potrebuje uzávery tieňových telies
- ✗ potrebuje posunúť zadnú orezávaciu rovinu do nekonečna
- ✗ pomalší z dvojice depth-pass, depth-fail
- ✗ vykresľuje viacej geometrie (uzávery)
- ✗ zložitejší na implementáciu

6 Záver

Práca sa zaoberala najpoužívanejšími technikami počítačovej grafiky, ktoré poskytujú výsledky v reálnom čase. Najprv popísala problém vrhania tieňov všeobecne, rozdelila tiene do viacerých kategórií a potom sa venovala jednotlivým technikám, najskôr zvlášť, a nakoniec dohromady. Vysvetlila princípy algoritmov, vymenovala vlastnosti vybraných techník, ich problémy a obmedzenia a spôsoby rozšírenia. Ukázala, ako je možné realizovať techniky v praxi, zaoberala sa implementačnými špecifikami a problémami. Nakoniec zhodnotila vybrané techniky z viacerých pohľadov. Z kvalitatívneho pohľadu pozerala na dosiahnuté vizuálne výsledky, z hľadiska výkonnosti si všímala výpočetnú náročnosť, z hľadiska použiteľnosti skúmala, pre aké modelové situácie je technika vhodná, hodnotila implementačnú náročnosť a nakoniec modifikovateľnosť techniky.

Problém vrhania tieňov je jednou z najbežnejších úloh v počítačovej grafike. Zobrazenie tieňov je pre realistické vnímanie scény veľmi dôležité, preto sa už od počiatkov počítačovej grafiky venuje mnoho úsilia na spoľahlivé, presné a čo možno najrýchlejšie riešenie tejto úlohy. Existuje mnoho algoritmov a ich rozšírení, ktoré sa o to snažia. Väčšina vychádza z troch základných postupov, vymyslených z dnešného pohľadu v dávnej minulosti. Sú to práve algoritmy rovinných tieňov, tieňových máp a tieňových telies, a práve preto sa táto diplomová práca venuje práve nim. Aj dnes, v dobre programovateľných grafických čipov, sa najprv musíme naučiť základné znalosti o princípoch fungovania počítačovej grafiky, aby sme sa mohli pohnúť ďalej a vytvoriť niečo nové a prínosné.

Aj napriek tomu, že základy boli položené pomerne dávno, nové techniky a nové myšlienky neustále pribúdajú. Môžeme tomu vďačiť rozvíjajúcim sa prostriedkom a podmienkam, ktoré tak krok po kroku približujú svet počítačovej grafiky k realistickému zobrazeniu sveta. Ešte pred pár rokmi v počítačových hrách (ktoré spolu s filmovým priemyslom najviac hýbu svetom grafiky) bol namiesto tieňa postavy zobrazený tmavý kruh. Dnes každý objekt vo virtuálnom svete vrhá presný a vizuálne kvalitný tieň. Dá sa predpokladať, že s postupom vývoja môžeme očakávať fyzikálne presné zobrazenie mäkkých tieňov spolu s inými, doteraz nemysliteľnými javmi v zobrazovacích systémoch pracujúcich v reálnom čase.

Oblasť problematiky vrhania tieňov je značne rozmerná. Nie je možné v jednej práci obsiahnuť všetky techniky a smery, ktorými sa uberá. Dúfam však, že podáva prehľad najzaujímavejších a najpoužívanejších techník súčasnosti.

Literatúra

[1] King, Y.: Ground-Plane Shadows. *Game Programming Gems.* Rockland, Massachusetts. Charles River Media. 2000. ISBN 1-58450-049-2. str. 562-566.

[2] Hallingstad, A. O.: Projective Shadows using Stencil Buffer [online]. 2003. Dostupné z WWW: <http://www.devmaster.net/articles/projectiveshadows/>

[3] Ambrož, D.: Techniky generování stínů v počítačové grafice. 2006. Dostupné z WWW: <http://www.shadowstechniques.com>

[4] Blinn, J.: Me and My (Fake) Shadow. *Jim Blinn's Corner.* Jan. 1988. str. 53-61.

[5] Vlachos, A.: Carving Static Shadows into Geometry. *Game Programming Gems 4.* Rockland, Massachusetts. Charles River Media. 2004. ISBN 978-1584502951. str. 427-435

[6] Herf, M., Heckbert P.: Fast Soft Shadows. *Technical Sketch.* New Orleans. SIGGRAPH '96 Visual Proceedings. Aug. 1996. str. 145

[7] Williams, L.: Casting curved shadows on curved surfaces. New York. ACM SIGGRAPH Computer Graphics. Vol. 12. Aug. 1978. ISSN 0097-8930. str 270-274

[8] Kilgard, M.: Shadow Mapping with Today's OpenGL Hardware. *GDC 2001 presentation.* 2001. Dostupné z WWW: <http://www.nvidia.in/object/gdc2001_shadows.html>

[9] Stamminger, M., Drettakis, G.: Perspective Shadow Maps. REVES – INRIA Sophia-Antipolis. Francúzsko. 2002.

[10] Kozlov, S.: Perspective Shadow Maps: Care And Feeding. GPU Gems 2: *Programming Techniques for High-Performance Graphics and General-Purpose Computation.* Addison-Wesley Professional. 2005. ISBN 0-321-33559-7. str. 217-244

[11] Dimitrov, R.: Cascaded Shadow Maps. NVIDIA. Aug. 2007. Dostupné z WWW: <developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf>

[12] Zhang, F., Sun, H., Nyman, O.: Parallel-Split Shadow Maps on Programmable GPUs. *GPU Gems 3*. Addison-Wesley Professional. 2007. ISBN 9780321545428. Dostupné z WWW: <http://appsrv.cse.cuhk.edu.hk/~fzhang/pssm_project/>

[13] Tobias, M., Tiow-Seng, T.: Anti-aliasing and Continuity with Trapezoidal Shadow Maps. Eurographics Symposium on Rendering. 2004. Dostupné z WWW: <http://www.comp.nus.edu.sg/~tants/tsm.html>

[14] Naigovzin, I.: Real-time shadow mapping. Israel Institute of Technology. Haifa, Israel. Mar. 2006.

[15] Bunnel, M., Pellacini, F.: Shadow Map Antialiasing. *GPU Gems.: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education. 2004. ISBN 0321228324

[16] Fernando, R..: Percentage-Closer Soft Shadows. Game Developers Conference 2005 Presentation. San Francisco. Mar. 2005

[17] Heidrich, W., Seidel, H. P.: Efficient rendering of anisotropic surfaces using computer graphics hardware. Image and Multi-dimensional Digital Signal Processing Workshop (IMDSP).1998.

[18] Brabec, S., Annen, T., Seidel, H. P.: Shadow Mapping for Hemispherical and Omnidirectional Light Sources. Max-Planck-Institut fur Infomatik. Saarbrucken, Nemecko. 2002.

[19] Fernando, R., Fernandez, S., Bala, K., Greenberg, D. : Adaptive Shadow Maps. SIGGRAPH 2001.

[20] Crow, F.: Shadow Algorithm for Computer Graphics. SIGGRAPH 1977. Vol.11, No. 3. str. 242-248.

[21] Brabec, S., Seidel, H. P.: Shadow Volumes on Programmable Graphics Hardware. EUROGRAPHICS 2003. Vol. 22, No. 3.

[22] Heidmann, T.: Real Shadows Real Time. Iris Universe, No. 18, str. 23-31, Silicon Graphics Inc., Nov. 1991. Dostupné z WWW: <http://developer.nvidia.com/docs/IO/2585/ATT/RealShadowsRealTime.pdf>

[23] Kilgard, M. J., Everit, C., McGuire, M., Hughes, J. F., Egan, K. T.: Fast, Practical and Robust Shadows. NVIDIA Corporation. Nov. 2003. Dostupné z WWW: <http://graphics.cs.brown.edu/games/FastShadows/index.html>

[24] Carmack, J.: E-mail to private list. Publikované firmou NVIDIA. 23.5.2000. Dostupné z WWW: <http://developer.nvidia.com/docs/IO/2585/ATT/CarmackOnShadowVolumes.txt>

[25] Bilodeau, B., Songy, M.: Talk at Creative Labs Inc. sponsored game developer conference. Los Angeles. Máj 1999.

[26] Kilgard, M. J., Everit, C.: Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. NVIDIA Corporation, Austin, TX. Mar. 2002. Dostupné z WWW: <http://developer.nvidia.com/view.asp?IO=robust_shadow_volumes>

[27] Lengyel, E.: The Mechanics of Robust Stencil Shadows. *Gamasutra.* Oct. 2002. Dostupné z WWW: <http://www.gamasutra.com/features/20021011/lengyel_01.htm>

[28] Everit, C., Rege, A., Cebenoyan, C.: Hardware Shadow Mapping. NVIDIA Corporation. Dostupné z WWW: <http://developer.nvidia.com/object/hwshadowmap_paper.html>

[29] Burian, T.: Algoritmy pro zobrazování stínu ve scéne. Ročníkový projekt. Fakulta Informačních Technologií VUT Brno. 2005

[30] Octavian, B., Porter, B., Molofee, J.: Shadows – NeHe Tutorial Lesson 27. [online]. Dostupné z WWW: <http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=27>

[31] Haines, E.: Soft Planar Shadows Using Plateaus. Autodesk, Inc. Ithaca, New York. Jún 2001.

[32] Brabec, S., Seidel, H. P..: Single Sample Soft Shadows using Depth Maps. Max-Planck-Institut für Informatik. Saarbrücken, Nemecko. 2002.

[33] Assarsson, U.: A Real-Time Soft Shadow Volume Algorithm. PhD thesis. Department of Computer Engineering, Chalmers University of Technology. ISBN 91-7291-333-9. Oct. 2003.

Zoznam príloh

Příloha 1. CD s demonštračnou aplikáciou. Okrem toho sa na ňom nachádza elektronická verzia práce, zdrojové súbory a manuál k demonštračnej aplikácii.

Příloha 2. Porovnanie výkonnosti vybraných techník

Příloha 3. Porovnanie kvality vybraných techník