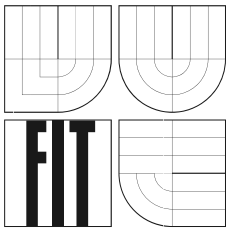
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Algoritmy pro zobrazování stínů ve scéně

Ročníkový projekt

2005 **Tomáš Burian**

Algoritmy pro zobrazování stínů ve scéně

Odevzdáno na Fakultě informačních technologií Vysokého učení technického v Brně
dne 10. května 2005

Prohlášení

Prohlašuji, že jsem tento ročníkový projekt vypracoval samostatně pod vedením Ing. Jana Pečivy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Tomáš Burian
10. května 2005

Abstrakt

Tento projekt se zabývá možnostmi realtimového zobrazování stínů ve 3D scéně. Popisuje v současnosti používané algoritmy a na demonstrační aplikaci ukazuje vlastnosti dvou z nich.

Implementovány byly dva algoritmy, z-pass a z-fail, založené na metodě stínových těles. Program je vytvořen v jazyce C++ s využitím grafického rozhraní OpenGL a volně šiřitelné knihovny Open Inventor.

Klíčová slova

Stín, stínová tělesa, stínový objem,OpenGL, Open Inventor, stencil buffer.

Poděkování

Rád bych poděkoval svému vedoucímu, panu Ing. Janu Pečivovi, za jeho trpělivost a poskytnuté rady.

Abstract

This work delas with possibilities of shadow rendering in 3D scene in real-time. Currently used algorithms are described and properties of two of them are showed in a demo application.

Two algorithms based on shadow volumes method, z-pass and z-fail, has been implemented. Programm is developed in C++ language with OpenGL graphic interface and Open Inventor library.

Keywords

Shadow, shadow volume, OpenGL, Open Inventor, stencil buffer.

Obsah

Kapitola 1

Úvod

1.1 Stíny v počítačové grafice

Stíny jsou nedílnou a běžnou součástí našeho světa. Podporují prostorové vnímání trojrozměrné reality, pochopení vzájemné polohy objektů, jejich tvaru a rozměrů. Podle stínu také můžeme usoudit polohu a vlastnosti světelného zdroje. Proto v počítačové grafice, která se snaží zobrazovat naši realitu co nejvěrněji, patří technikám vytvářející stíny důležité místo.

Vytváření realistických stínů bylo a stále je pro realtimovou počítačovou grafiku těžký úkol. A to ze dvou důvodů. Prvním z nich je přesnost a obecnost algoritmu. V historii vývoje těchto algoritmů se objevilo několik metod, které buď nezobrazovaly stíny správně nebo generované stíny sice byly správné, ale algoritmus nepracoval za všech podmínek (např. byla-li kamera umístěna ve stínu).

Druhým důvodem je rychlost těchto algoritmů. Pro interaktivní aplikace (hry, virtuální realita) je nutné, aby výpočetní složitost stínů příliš nebrzdila procesor. Tento požadavek s rostoucí výkonností grafických procesorů a přesunem mnoha výpočetních úkolů právě na tyto hardwarové jednotky pomale ustupuje do pozadí. Nicméně rychlost algoritmu je stále důležitá.

Tato práce má za cíl shrnout nejběžnější algoritmy pro zobrazování stínů ve 3D scénách, popsat jejich základní vlastnosti, a vybrané z nich implementovat. Podmínkou implementace bylo, aby techniky byly použitelné v prostředí open source grafické knihovny Open Inventor [1] a OpenGL. Zvolené techniky také musejí pracovat v reálném čase, aby mohly být použity ve hrách a dalších interaktivních aplikacích.

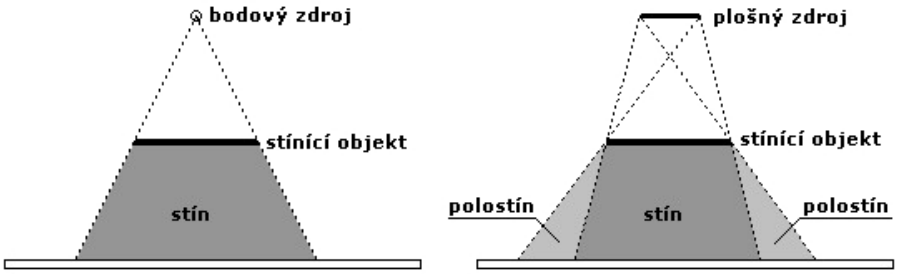
Ve zbytku této kapitoly je popsán vliv použitého světelného zdroje na tvar stínu a základní rozdělení stínů na *vržené* a *vlastní*.

V následující kapitole jsou charakterizovány současné metody používané pro zobrazování stínů v počítačové grafice.

Další kapitola popisuje postup vytvoření stínů pomocí zvolené metody stínových těles a její implementaci v grafické knihovně Open Inventor.

Následující část se zabývá aplikací, v níž jsou implementované metody použity, a popisuje její ovládání.

V předposlední kapitole jsou zhodnoceny dosažené výsledky a v závěru nastíněny možnosti jejich uplatnění ve vývoji komplexního enginu pro zobrazování stínů v rámci knihovny Open Inventor.



Obrázek 1.1: Bodový zdroj vrhající ostrý stín a plošný zdroj světla vytvářející měkký stín

1.2 Typy stínů

Tvar a velikost stínu závisejí na vzájemné poloze světelného zdroje, stínícího objektu a objektu, na který stín dopadá. Charakter stínu je pak ovlivněn tvarem a velikostí světelného zdroje.

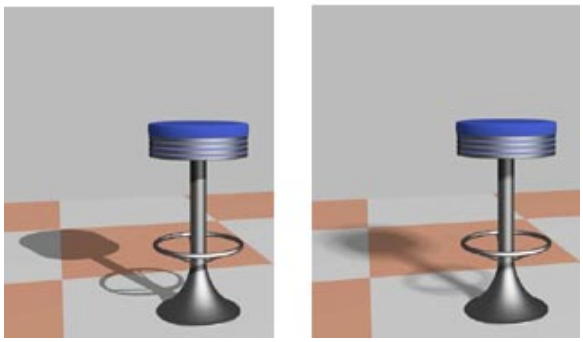
V reálném světě existují pouze *plošné zdroje světla*, které vytvářejí *měkké stíny* (*soft shadows*). Např. Slunce, které je velmi vzdáleno od Země, je plošný zdroj světla. Proto naše stíny vrhané slunečními paprsky na zem se skládají ze dvou částí: *plného stínu* (*umbrum*) a *polostínu* (*preumbrum*) (viz obrázek 1.1).

Polostín tvoří pozvolný přechod plného stínu k osvětlené části. Zatímco polostín se s rostoucí velikostí světelného zdroje zvětšuje, plný stín se naopak zmenšuje a může i úplně zmizet.

Plošné stíny se často v počítačové grafice zjednodušují a bývají nahrazeny *bodovými světelnými zdroji*. Tyto zdroje ovšem vytváří *ostré stíny* (*hard shadows*) (viz obrázek 1.1). Ostré stíny mají přesně vymezenou hranici zastíněné plochy.

Pro svoji jednoduchost se v počítačové grafice často používají právě bodové zdroje světla (vrhající ostré stíny), a měkké stíny se dotváří uměle. To lze provést například rozmazáním okrajů ostrých stínů nebo použitím více bodových zdrojů o menší intenzitě.

Příklad ostrého a měkkého stínu ukazuje obrázek 1.2.



Obrázek 1.2: Rozdíl mezi ostrým (vlevo) a měkkým (vpravo) stínem pro tentýž objekt

Často se rozlišují dva druhy stínů – *vlastní* (*self shadow*) a *vržený* (*cast shadow*). Vržený stín je stín, který vrhá jeden objekt na druhý (např. stín na povrchu scány) a který napomáhá rozpoznat umístění těchto objektů v prostoru. Vlastní stín se vytváří přímo na objektu, který jej způsobuje. Jednak do něj patří všechny od světla odvrácené plochy daného objektu a jednak stíny, které vrhá

jedna část tělesa na druhou a dochází k tzv. *samozastínění* (na ke světlu přivrácených plochách). Ne všechny algoritmy jsou schopny nalézt takovéto vlastní stíny.

Kapitola 2

Algoritmy pro zobrazování stínů

2.1 Globální osvětlovací modely

Realistické globální osvětlovací modely, jako je *radiosita*, vytváří stíny objektů automaticky. Princip metody spočívá v modelování vzájemného vyzařování a pohlcování světelné energie objekty ve scéně. Výsledkem jsou velmi realistické, ale statické scény. Největší výhodou je, že model vyzařování světelné energie je pohledově nezávislý a může být pro scénu předpočítán dopředu. Další výhodou je, že touto metodou lze modelovat průhledné objekty. Příklad scény vytvořené pomocí této metody je na obrázku 2.1.

Výpočetní náročnost je ovšem na současném hardwaru příliš velká na to, aby bylo možné tuto metodu použít v realtimové grafice. Proto se výzkum zaměřil na empirické metody, které měly zobrazovat stíny i v dynamických scénách.

Obrázek 2.1: Scéna vytvořená radiační metodou

2.2 Rovinné stíny

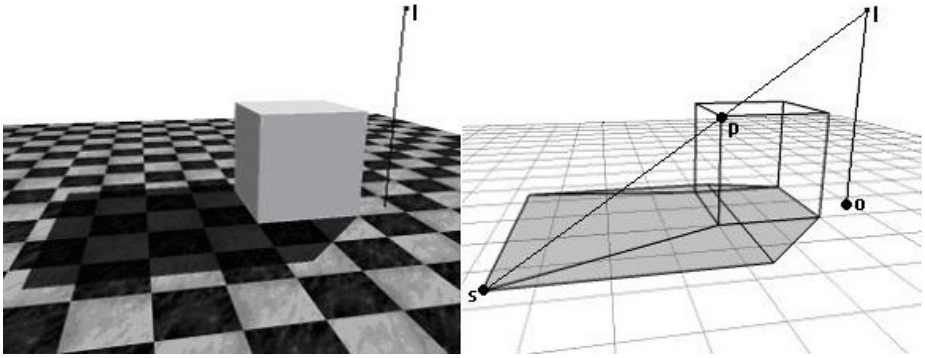
Tento základní algoritmus pracuje s polygonální reprezentací objektů. Stíny vznikají tak, že se pro každou plochu, na kterou může dopadat stín, nalezne transformace zobrazující do roviny této plochy libovolný objekt jako dvojrozměrný polygon. Promítneme tak siluetu tělesa z pozice světelného zdroje na povrchovou plochu (viz obrázek 2.2). V literatuře bývá tato metoda také označována jako projekční.

Pro zobrazení stínů tímto algoritmem lze s výhodou použít kombinaci *bufferu hloubky* (*z-buffer*) a *šablony* (*stencil buffer*). Nejprve je nutné vyřešit viditelnost scény a do šablony pro každý výsledný pixel poznamenat identifikátor viditelné plochy. Poté se postupně vytvářejí polygony vržených stínů a porovnává se hodnota pixelu s identifikátorem plochy uloženým v šabloně. Pokud jsou hodnoty shodné, znamená to, že daný pixel je ve stínu vyhodnocovaného světelného zdroje a musí být ztmaven. Detailnější popis lze nalézt v [2].

Algoritmus má ale určitá omezení. Jedním z nich je, že povrchem musí být rovinná plocha, jinak algoritmus nepracuje správně. Dále metoda rovinné transformace neuvažuje polohu objektů ve scéně a mohou tak vznikat *falešné stíny* (*fake shadows*), pro každé světlo je nutné zobrazit scénu zvlášť a také neumožňuje vytváření vlastních stínů.

Díky těmto omezení nelze metodu použít ve složitějších scénách, ale např. ve hrách lze její pomocí rychle vytvářet stíny, které sice nejsou přesné, ale dodávají prostoru hloubku.

Algoritmus vytváří ostré stíny, může ale být jednoduše rozšířen pro zobrazovaní měkkých stínů, malými změnami polohy světelného zdroje v několika průchodech scénou a rozmazáním výsledků.



Obrázek 2.2: Stín vytvořený projekcí siluety objektu na rovinnou plochu

2.3 Stínové mapy

Vytváření stínů pomocí stínových map probíhá kompletně v obrazovém prostoru, tzn. že algoritmus nemusí mít žádnou informaci o geometrii scény. Algoritmus pracuje s libovolnou reprezentací 3D objektů, na druhou stranu je použitelný pouze pro bodové zdroje světla.

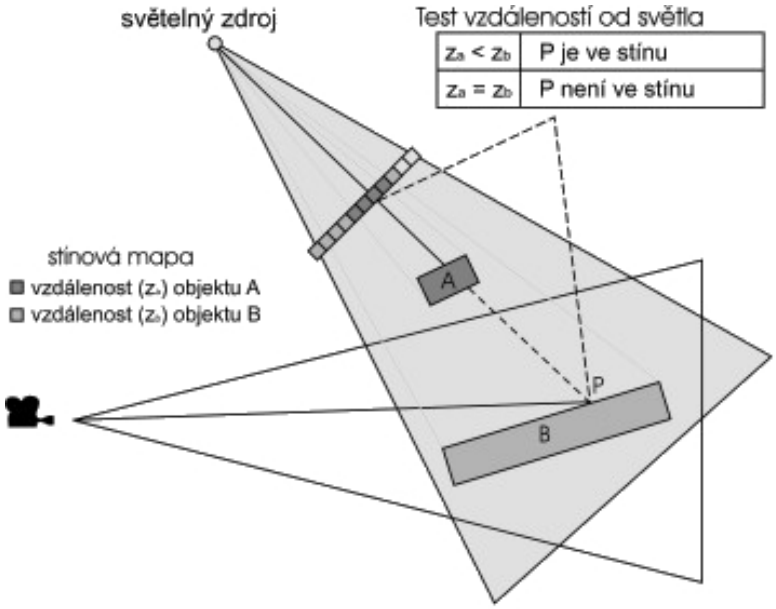
Principiálně se řešení stínů převádí na řešení viditelnosti pomocí *stínové (hloubkové) mapy* uložené v z-bufferu.

Nevýhodami algoritmicky jednoduché metody je její velká paměťová náročnost, nepřesnost a poměrně velký aliasing (z důvodu omezeného rozlišení stínových map). I přesto je tento algoritmus používán i velkými studii (Pixar použil tuto techniku při vytváření stínů v počítačově animovaném filmu Toy Story).

Hlavní myšlenku tohoto algoritmu publikoval v roce 1978 Lance Williams. Algoritmus zobrazuje scénu celkem dvakrát. V prvním průchodu se kamera pomyslně umístí do pozice zdroje světla a scéna se vyrendruje do z-bufferu (hloubkový buffer). Výsledkem je *stínová*, resp. *hloubková mapa*, která je vlastně funkcí mapující pixel nejbližšího objektu ve scéně z pozice světla do 2D obrazových souřadnic.

Tato mapa je poté použita při druhém průchodu, kdy se scéna zobrazuje ze skutečné pozice pozorovatele, pro určení zastíněných těles. Získaná mapa se promítne na zobrazované objekty. Hloubka každého vykreslovaného pixelu je porovnána s hodnotou ve stínové mapě. Pokud je hloubka pixelu větší, musí být před pixelem nějaké těleso blíže světelnému zdroji a pixel je tedy ve stínu, vykreslí se tmavší barvou. Obrázek 2.3 ukazuje princip této metody.

Tato metoda je často zmiňována jako *stínová paměť hloubky*. Detailnější popis lze nalézt v [7] a [4].



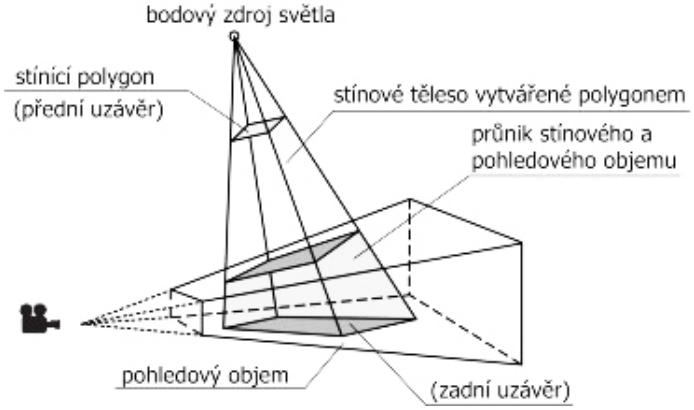
Obrázek 2.3: Princip vytvoření stínu pomocí hloubkové mapy

2.4 Stínová tělesa

Tento algoritmus řeší zobrazování stínů v objektovém prostoru, kde paprsky světla a stínící objekty vytvářejí tzv. *stínová tělesa* (*stínový objem*, *shadow volume*), viz [3]. Tato metoda, díky své robustnosti a hardwarové podpoře v grafických kartách, patří společně s metodou stínové paměti hloubky 2.3 mezi nejčastěji používané algoritmy pro výpočet stínů v reálném čase.

Algoritmus v základní podobě pracuje s polygony a bodovými světly a poskytuje tedy pouze ostré stíny. Základní myšlenkou je vytvoření tzv. stínového tělesa pro každý ze stínících objektů. Konstrukci takového tělesa ukazuje obrázek 2.4.

Jek je z obrázku patrné, stínové těleso ohraničuje část prostoru ve scéně, ze kterého není přes stínící objekt světelný zdroj vidět, a vymezuje tak zdrojem neosvětlený prostor. Stínové těleso je potřeba vytvořit pro každý ze stínících objektů. Známe-li všechna stínová tělesa řešení stínů lze převést na řešení viditelnosti objektů ve scéně vůči těmto tělesům.



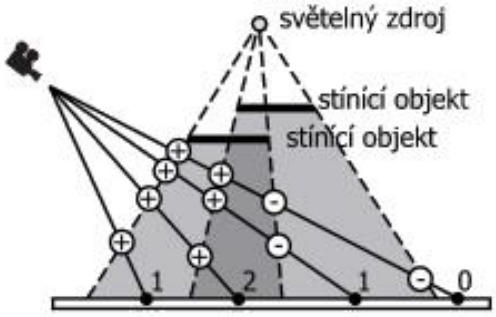
Obrázek 2.4: Vytvoření stínového objemu bodovým zdrojem světla a jeho průnik s pohledovým tělesem

Určení stínového tělesa pro jeden samostatný stínící polygon je snadné. V případě obecného stínícího objektu však musíme nejprve nalézt jeho obrys - ten určuje hranici stínu. Obrys je tvořen obrysovými hranami, tzv. *silhouette edges*. Každá obrysová hrana definuje jednu stěnu stínového tělesa. Vytváření stínových těles ze siluety není nutnou podmínkou. Je možné vytvořit stínové těleso pro každou (ke světlu přivrácenou) plochu objektu, čímž se však zvýší počet stínových těles a prodlužuje výpočet stínů. Podrobněji v [2].

Depth-pass algoritmus

Řešení viditelnosti objektů ve scéně a stínového tělesa se provádí v prostoru rastru. Představa, jak zobrazovat stíny ve scénách za pomoci stínových těles, je založena na vysílání testovacích paprsků ze středu promítání každým pixelem směrem k zobrazovanému povrchu ve scéně. Během cesty paprsku se počítá, kolikrát paprsek vstoupil do nějakého stínového tělesa a kolikrát je opustil 2.5.

Pokud je rozdíl těchto hodnot nenulový, znamená to, že paprsek neopustil všechna stínová tělesa, do kterých vstoupil a bod na povrchu tělesa, ke kterému dorazil, musí ležet ve stínu. Můžeme si to také představit tak, že ke kameře přivrácené plochy stínového tělesa „přesouvají“ vše, co se nachází za nimi, do stínu, zatímco odvrácené plochy jejich účinek ruší. K tomu, aby těleso nebo jeho část ležely ve stínu, pak musí platit, že se nacházejí za přivrácenými a současně před odvráceným plochami stínových těles.

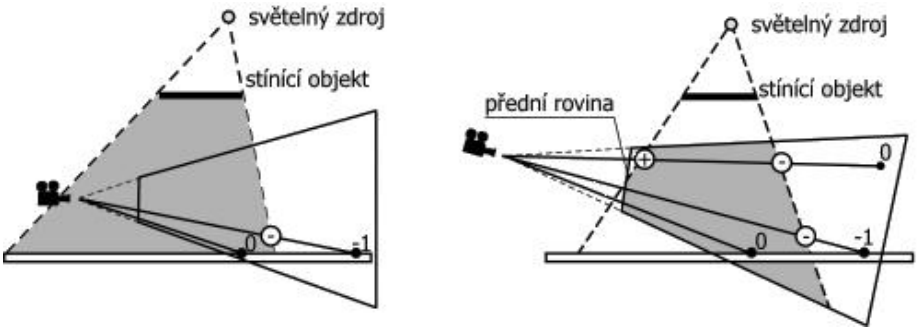


Obrázek 2.5: Princip depth-pass algoritmu

K rozlišení těchto situací můžeme využít hloubkový test. Nejprve vyřešíme viditelnost scény bez stínových těles za pomoci paměti hloubky a přitom výsledek v paměti zachováme. Každému pixelu přiřadíme čítač, jehož počáteční hodnota bude rovna počtu stínových těles, uvnitř kterých se nachází kamera. Nyní stačí provést hloubkový test polygonů stínových těles s ostatními objekty ve scéně. Bude-li pro zvolený pixel test úspěšný, tj. polygon stínového tělesa se nachází před všemi objekty ve scéně, pak v případě přivrácené stínové plochy čítač inkrementujeme a v případě odvrácené plochy dekrementujeme. Pokud po zpracování všech ploch stínových těles zůstane v čítači nenulová hodnota, pak pixel leží ve stínu.

K implementaci popsaného algoritmu se používá *šablona* (*stencil buffer*). V anglické literatuře se metoda označuje jako *depth-pass* (*z-pass*), protože k aktualizaci čítačů pixelů v šabloně dochází jen v případě, že byl hloubkový test úspěšný.

Při konkrétní realizaci depth-pass algoritmu však musíme brát v úvahu, že některé stěny stínového tělesa mohou být ořezány přední nebo zadní rovinou pohledového tělesa. Ve stínovém tělese pak mohou vniknout trhliny, které způsobí chybné určení stínů. Příklad takové situace je na obrázku 2.6. Ke špatnému určení stínů také dochází, když se kamera (včetně přední pohledové roviny) sama nachází ve stínovém tělese.



Obrázek 2.6: Selhání depth-pass algoritmu: vlevo situace, kdy se kamera nachází uvnitř stínového tělesa, vpravo ořezání stínového tělesa přední rovinou pohledového tělesa (oba dolní paprsky)

Depth-fail algoritmus

Problémům s přední rovinou pohledového tělesa se lze vyhnout, pokud obrátíme smysl algoritmu *depth-pass*, tj. pokud budeme měnit hodnotu čítače (iniciálně nastaveného na nulu) jen v případě, že hloubkový test selže (stěna stínového tělesa bude za testovaným pixelem). Tímto způsobem tedy vlastně budeme počítat navštívená a opuštěná stínová tělesa paprskem, který směřuje z nekonečné vzdálenosti k zobrazovanému (nejbližšímu) povrchu - tedy přesně opačným směrem, než v *depth-pass* algoritmu.

Tento nový algoritmus, označovaný jako *depth-fail*, (*z-fail*), se do konfliktu s přední ořezávací rovinou nedostává a je navíc nezávislý na počáteční poloze kamery. Oproti metodě *depth-pass* je však tentokrát potřeba hlídat případné ořezání zadní rovinou pohledového objemu a nepřipustit, aby ořezáním vznikly „otvory“ do stínového tělesa. Dalším požadavkem je uzavření stínového objemu (tělesa). K bočním stěnám stínového tělesa proto musíme přidat přední (*front cap, light cap*) a zadní uzávěr (*back cap, dark cap*), viz obrázek 2.4.

Přední uzávěr je tvořen (ke světlu) přivrácenými plochami stínícího objektu, zatímco zadní uzávěr získáme projekcí odvrácených ploch do vzdálenosti, o kterou jsme prodloužili boční stěny stínového tělesa.

Problému s ořezáním zadní rovinou pohledového tělesa se lze vyhnout umístěním této roviny do nekonečné vzdálenosti od kamery. Pokud zajistíme, že stínové těleso bude vždy uzavřené a nebude ořezáno zadní rovinou, pak společně s algoritmem *depth-fail* získáme univerzální metodu označovanou jako *robust stencil volumes*.

Oba algoritmy, *depth-pass* i *depth-fail*, se zpravidla implementují současně a teprve během zobrazování se rozhoduje o tom, zda je možné použít jednodušší a a rychlejší *depth-pass* nebo univerzální a složitější *depth-fail*. Pro metodu stínových těles je navržena řada optimalizačních technik, blíže v [5].

Existují také možná rozšíření této metody pro generování měkkých stínů.

Kapitola 3

Implementace

3.1 Vybrané metody

Pro svoji univerzálnost byly pro generování stínů v rámci knihovny Open Inventor vybrány výše popsané algoritmy *depth-pass* a *depth-fail* používají metodu stínových těles. Cílem bylo vytvořit demonstrační aplikaci, na které by se prověřilo použití těchto metod pod danou knihovnou.

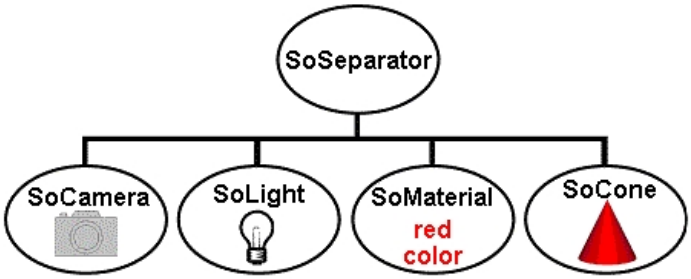
3.2 Open Inventor

Open Inventor je velmi populární knihovna pro tvorbu realtimové 3D grafiky postavená nad OpenGL. Programátorovi poskytuje rozsáhlou množinu C++ tříd, které skrývají vlastní OpenGL API a posunují ho na mnohem vyšší úroveň. Vývoj aplikací tak může probíhat mnohem rychleji. Navíc, aplikace napsané v Open Inventoru jsou obyčejně rychlejší než ty přímo psané v OpenGL, protože před předáním zobrazované scény OpenGL knihovna provádí její optimalizaci.

Coin3D norské firmy Systems In Motion je jedním ze tří knihoven kompatibilních s Open Inventor API (dalšími jsou verze Open Inventoru firem TGS a SGI). Její výhodou je, že je k dispozici pod pod GPL licencí. Právě pod touto knihovnou byly vyvinuty algoritmy pro zobrazování stínů. Pojmem Inventor, resp. Open Inventor, bude nadále rozuměna knihovna Coin3D. Open Inventor API je podrobně popsáno v knize [8].

Design Open Inventoru vychází z konceptu grafu scény. Tedy, scéna je složena z *uzlů* - anglicky *nodes*. Nody jsou různých typů. Jedny nesou informace o geometrii těles (krychle, kužel, model tělesa), další různé atributy (barva, textury, souřadnice objektu) a také existují speciální nody, které obsahují seznam jiných nodů, anglicky zvané groups. A právě tyto groupy umožňují organizovat ostatní nody do hierarchických struktur zvaných grafy. Takovýto graf nám pak reprezentuje naši scénu.

Obrázek 3.1 ukazuje jednoduchý graf scény. Z něj je patrné, že kořen grafu tvoří objekt typu `SoSeparator`. Ten se pro své speciální vlastnosti (např. dokáže scénu pod sebou předkompilovat do OpenGL display listu a tím urychlit proces renderování) pro oddělení jednotlivých objektů tvořících výslednou scénu. Na obrázku dále vidíme, že kořen má čtyři syny: kamera, světlo, materiál a kužel. První z nich - kamera - je speciální nod, který určuje umístění pozorovatele a některé další atributy pohledu do scény. Světlo (`SoLight`) osvětluje scénu bílým světlem. Následující nod, tedy materiál, udává optické vlastnosti kužele, resp. udává jeho barvu. Posledním nodem je pak vlastní kužel, což je nod specifikující geometrii tělesa. Pro další informace lze doporučit tutoriál [6] o Open Inventoru.



Obrázek 3.1: Jednoduchý graf scény v Open Invetoru

3.3 Struktura grafu scény pro zobrazení stínů

Vyvinutá metoda zobrazování stínů v rámci knihovny Open Inventor je navržena obecně, aby bylo možné zobrazovat stíny pro jakoukoli scénu. V průběhu jejího vývoje se vyskytla sice určitá omezení a požadavky na modely objektů (budou objasněny níže), ale ty by měly být v průběhu jejího dalšího vývoje odstraněny.

Pro generování stínů objektů ve scéně, která má kořen v uzlu `sceneRoot` je v aplikaci vytvořen nový kořenový uzel celé scény typu `SoSeparator` pojmenovaný `superRoot`, který mu bude v grafu nadřazen. Kromě něj jsou je třeba vytvořit i další uzly.

Popis všech důležitých uzlů (všechny jsou typu `SoSeparator`):

- `superRoot` - nově vytvořený „super kořen“. Všechny další uzly budou k němu připojeny. K tomuto uzlu je také přidána kamera pro zobrazení scény včetně stínů.
- `sceneRoot` - kořen scény pro níž chceme generovat stíny. Obsahuje objekty tvořící povrch scény, modely těles a uzly s definicí světel.
- `shadowRoot` - nově vytvořený uzel, ke kterému budou připojeny všechny uzly stínových těles objektů ve scéně.
- `wireRoot` - nový uzel, budou k němu připojeny „drátěné modely“ stínových těles.

Jak bylo uvedeno v části 2.4, metoda stínových těles pracuje s bodovými zdroji světla. V demonstrační aplikaci jsou použita světla typu `SoPointLightManip`, která uživateli umožňují měnit polohu světelného zdroje a tím i interaktivně měnit tvary stínů objektů.

Pro modely objektů ve scéně platí omezení, že musejí být složeny z trojúhelníků, konkrétně plášťů trojúhelníků, které jsou vytvořeny pomocí indexů na jejich vrcholy. V Open Inventoru tomu odpovídá typ `SoIndexedTriangleStripset`. Při implementaci bohužel vyvstalo omezení, že indexy vrcholů jednotlivých trojúhelníků musejí být orientovány proti směru hodinových ručiček. Snahou dalšího vývoje bude toto omezení odstranit a zobrazovat stíny pro tělesa nezávisle na typu polygonů.

3.4 Světla ve scéně

V průběhu renderingu stínů je nutné světla ve scéně vypínat a zapínat. Proto byla pro společnou správu světel vytvořena třída `SceneLights`. Každé světlo, pro které mají být stíny vyřešeny je

nutné do této třídy přidat metodou `addLight(SoLight * light)`. Odebrat světlo je možné pomocí metody `removeLight(SoLight * light)`.

3.5 Stínová tělesa

Třída `ShadowVolume` slouží pro vytvoření stínového tělesa pro objekty ve scéně. Pro každý objekt, který má ve scéně vrhat stíny, je nutné vytvořit objekt této třídy pomocí konstruktoru:

```
ShadowVolume(SoSeparator * root,
             SoSeparator * shadow,
             SoSeparator * wire,
             SoSeparator * object);
```

Argumenty konstruktoru tvoří ukazatele na důležité uzly v grafu scény. Proměnná `root` značí kořen scény, `shadow` uzel, ke kterému se připojují uzly stínových těles, `wire` je uzel, ke kterému se připojí drátěné modely stínových těles a konečně `object` je kořenový uzel objektu, pro který se má stínové těleso vytvořit.

Při vytváření objektu stínového tělesa se podle parametrů získaných z konstruktoru naleznou uzly s vrcholy trojúhelníků tělesa, jejich indexy a normálové vektory. Také se hledá transformační matice pro převod vrcholů objektu z jeho lokálních souřadnic do globálních, platných ve scéně.

Tato třída vytváří stínová tělesa pro všechna světla typu `SoPointLightManip`, která nalezne v grafu scény.

3.6 Nalezení stínících trojúhelníků

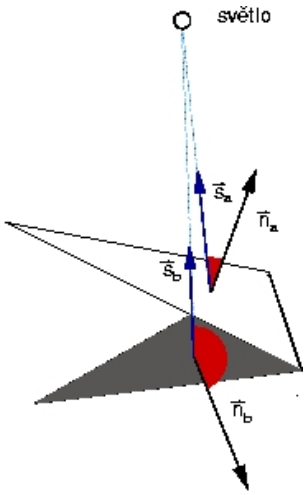
Aby bylo možné zkonstruovat stínové těleso, je nutné nalézt obrys tělesa, který určuje hranici stínu. Obrys je tvořen obrysovými hranami, přičemž každá obrysová hrana definuje jednu stěnu stínového tělesa. Vytváření stínových těles ze siluety ale není nutnou podmínkou. Je možné vytvořit stínové těleso pro každou (ke světlu přivrácenou) plochu objektu. Tento postup, který je sice pomalejší, je použit ve vytvořené aplikaci. Důvodem pro toto zjednodušení byla skutečnost, že nalezení obrysových hran v obecně definovaném objektu není algoritmicky triviální problém.

Stínová tělesa jsou tedy vytvořena pro každý ke světlu přivrácený trojúhelník. Zda je trojúhelník přikloněn ke světlu nebo ne lze poznat ze skalárního součinu normály trojúhelníka a vektoru směřujícího ke světelnému zdroji z jednoho jeho bodu (vrcholu), tzv. světelný vektor. Z obrázku 3.2 je patrné, že úhel mezi normálou a světelným vektorem příslušného trojúhelníka, který je přikloněn ke světlu, je menší než 90 stupňů. Tedy $\vec{n}_a \cdot \vec{s}_a > 0$. Stačí tedy testovat, zda výsledek skalárního součinu těchto dvou vektorů je nezáporný. Hrana mezi trojúhelníkem přivráceným ke světlu a odvráceným je *obrysová hrana*.

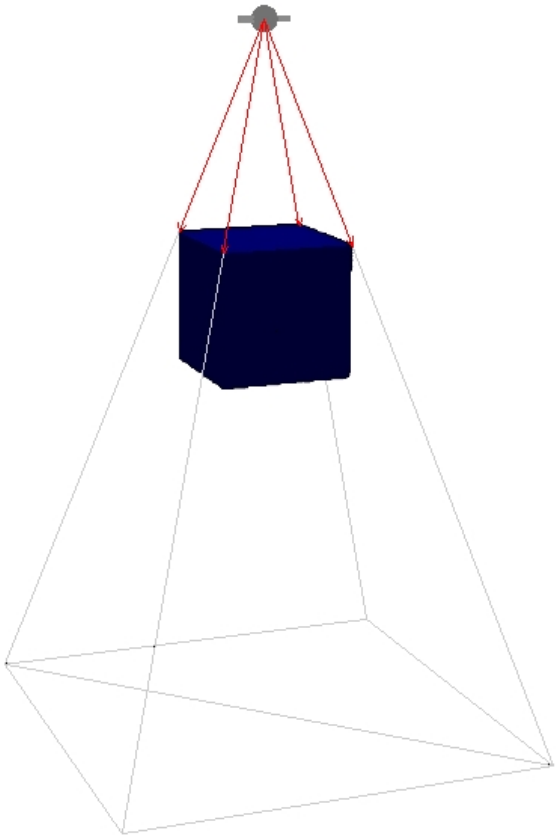
3.7 Konstrukce stínového tělesa

Pro každý trojúhelník, který je přivrácen ke světlu, vytvoříme stínové těleso tak, že vrhneme paprsek z pozice světla přes jeho vrcholy. Takto získaným vektorem promítneme vrchol příslušný vrchol do zadní stěny stínového tělesa.

Tímto způsobem vytvoříme plášť stínového objemu, abychom jej uzavřeli (jak to vyžaduje algoritmus *depth-pass*), tak vyplníme trojúhelníky přední a zadní uzávěr. Tvoří jej vrcholy ke světlu



Obrázek 3.2: Rozpoznání ke světlu přivráceného trojúhelníka



Obrázek 3.3: Konstrukce stínového objemu, červeně jsou zvýrazněny vektory ze zdroje světla k vrcholům přivrácených trojúhelníků

přivrácených trojúhelníků, resp. jejich promítnuté obrazy. Stínová tělesa mají v programu pevně danou hloubku.

Stínová tělesa se úspěšném nalezení všech potřebných údajů v grafu scény přidají ke kořenovému uzlu všech stínových těles `shadowRoot`. Pro zobrazení stínových těles samotných jsou drátěné modely těchto těles přidány k uzlu `wireRoot`.

3.8 Renderování stínů

Pro vykreslení stínů pomocí stínových stínových těles jsou implementovány dva algoritmy *depth-pass* a *depth-fail* využívající hardwarovou podporu bufferu šablony (stencil bufferu). Protože ale Open Inventor neposkytuje možnosti k nastavení vlastností stencil bufferu (pouze jej dokáže vypnout a zapnout), je tato část kódu vytvořena přímo v OpenGL.

Základ zdrojového kódu pro vytvoření stínů pomocí *depth-pass* algoritmu:

```
...
// Nastavení a vymazání color bufferu, z-bufferu a stencil bufferu

// Vykreslení scény s ambientním osvětlením a
// uchování scény v z-bufferu
// (zde je nutné vypnout všechna světla ve scéně)
// Bude tvořit zastíněné oblasti
setAmbientLightOnly();
renderScene();

// Nastavení stencil bufferu
// Hodnoty v color bufferu a z-bufferu se nemění
glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_ALWAYS, 0, ~0);
glColorMask(0, 0, 0, 0);
glDepthMask(0);

// Vykreslení předních stěn stínových těles
// Hodnota ve stencil bufferu se inkrementuje vždy,
// když hloubkový test uspěje
glCullFace(GL_BACK);
glStencilOp(GL_KEEP, GL_KEEP, GL_INCR);
renderShadowVolumes();

// Vykreslení zadních stěn stínových těles
// Hodnota ve stencil bufferu se dekrementuje vždy,
// když hloubkový test uspěje
glCullFace(GL_FRONT);
glStencilOp(GL_KEEP, GL_KEEP, GL_DECR);
renderShadowVolumes();

// Vykreslení scény s plným osvětlením
// (světla se zapnou/obnoví na jejich původní hodnotu)
// Vykreslují se pouze ty části scény, pro které je ve stencil
```

```
// bufferu hodnota 0
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glDepthFunc(GL_EQUAL);
glStencilFunc(GL_EQUAL, 0, ~0);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
setNormalLight();
renderScene();

// Zpětné nastavení původního stavu
glDepthMask(GL_TRUE);
glDepthFunc(GL_LEQUAL);
glDisable(GL_STENCIL_TEST);
...
```

Pro algoritmus *depth-fail* je kód podobný s touto obměnou:

```
...
// Vykreslení zadních stěn stínových těles
// Hodnota ve stencil bufferu se inkrementuje vždy,
// když hloubkový test selže
glCullFace(GL_FRONT);
glStencilOp(GL_KEEP, GL_INCR, GL_KEEP);
renderShadowVolumes();

// Vykreslení předních stěn stínových těles
// Hodnota ve stencil bufferu se dekrementuje vždy,
// když hloubkový test selže
glCullFace(GL_BACK);
glStencilOp(GL_KEEP, GL_DECR, GL_KEEP);
renderShadowVolumes();
...
```

Vykreslování scény v Open Inventoru probíhá pomocí renderovací akce, která prochází graf scény od kořene k listům zprava doleva. Postupně tedy mění vlastnosti stavového automatu OpenGL (materiály, osvětlení, transformace objektů) a vykresluje uzly s objekty scény.

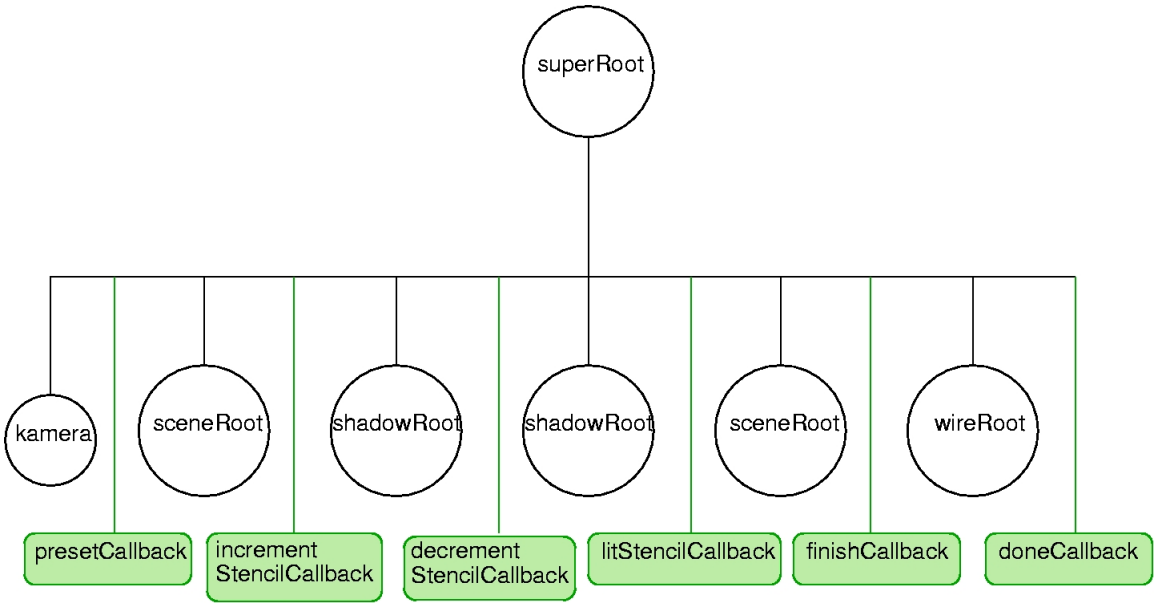
Výše uvedený pseudokód však vyžaduje několikrát měnit nastavení stencil bufferu a dalších parametrů a také projít některé grafy scén dvakrát. To renderovací akce nedokáže. Open Inventor ale poskytuje prostředky v podobě *callbacků*, které lze vložit do grafu scény, a které se aktivují vždy, když na ně narazí renderovací akce při průchodu grafem scény. Ke každému *callbacku* přísluší programátorem definovaná funkce, která se zavolá, když je *callback* aktivován. Do této funkce je možné vložit kód pro nastavení stencil bufferu a také jí lze předat z *callbacku* vlastní parametr.

Je tedy nutné vytvořit tyto *callbacky*:

- `presetCallback` - zapnutí a inicializace color a z-bufferu. Vypnutí světel, scéna se vykreslí jen s ambientním světlem.
- `incrementStencilCallback` - inicializace stencil bufferu, inkrementace stencil bufferu.
- `decrementStencilCallback` - dekrementace stencil bufferu.

- `litStencilCallback` - zapnutí světel a vykreslení scény s plným osvětlením tam, kde je v maska šablony rovna nule.
- `finishStencilCallback` - obnovení nastavení před vykreslováním stínů, světlo pro zobrazení drátěných modelů stínových těles.
- `doneCallback` - dokončení vykreslování.

Callbacky `incrementStencilCallback` a `decrementStencilCallback` nastavují vykreslování předních, resp. zadních, stěn stínových těles podle toho, zde je použit algoritmus *depth-pass* nebo *depth-fail*.



Obrázek 3.4: Graf scény se stíny

Jak bylo popsáno v části 3.3, je v programu vytvořen nový kořen celé scény se stíny, tzv. `superRoot`. K němu jsou postupně přidány uzly se scénou a stínovými tělesy s vhodně umístěnými *callbacky*, jak je znázorněno na obrázku 3.4. Uzly jsou tedy přidány takto:

```
superRoot->addChild(camera);
superRoot->addChild(presetCallback);
superRoot->addChild(sceneRoot);
superRoot->addChild(incrementStencilCallback);
superRoot->addChild(shadowRoot);
superRoot->addChild(decrementStencilCallback);
superRoot->addChild(shadowRoot);
superRoot->addChild(litStencilCallback);
superRoot->addChild(sceneRoot);
superRoot->addChild(finishStencilCallback);
superRoot->addChild(wireRoot);
superRoot->addChild(doneCallback);
```

Kapitola 4

Demonstrační aplikace

4.1 Popis aplikace

Pro předvedení výsledků zobrazených stínů byla vytvořena jednoduchá demonstrační aplikace. V programu jsou předem připraveny dvě scény s modely objektů a světlem, jehož polohu lze interaktivně měnit. Argumentem příkazového řádku `-demo` lze zvolit, jaká scéna se má zobrazit. Pro první scénu je tedy třeba spustit program takto: `shadows -demo 1`. Tato scéna je nastavena jako výchozí a zobrazí se po spuštění programu bez parametrů.

Dále je možné zvolit, jaký algoritmus se má použít pro výpočet stínů. K tomu slouží parametr `-zfail`. Jako výchozí nastavení je v aplikaci použit algoritmus *depth-pass*. Pro zobrazení stínů pomocí *depth-fail* je nutné zadat `shadows -zfail 1`. Příkazem `shadows -h` nebo `shadows -help` se zobrazí nápověda ke spuštění programu.

Spuštěním aplikace se otevře okno se scénou. Scénu je možné ovládat ve dvou režimech. V prvním, tzv. „průzkumném“, lze myší scénu otáčet a pohybovat s kamerou pozorovatele. Stisknutím `Esc` se program přepne do „ovládacího“ režimu. Zde lze myší měnit polohu světelného zdroje (a tím i výsledné stíny objektů) a klávesami `S`, resp. `D`, lze zobrazit a skrýt drátěné modely stínových těles.

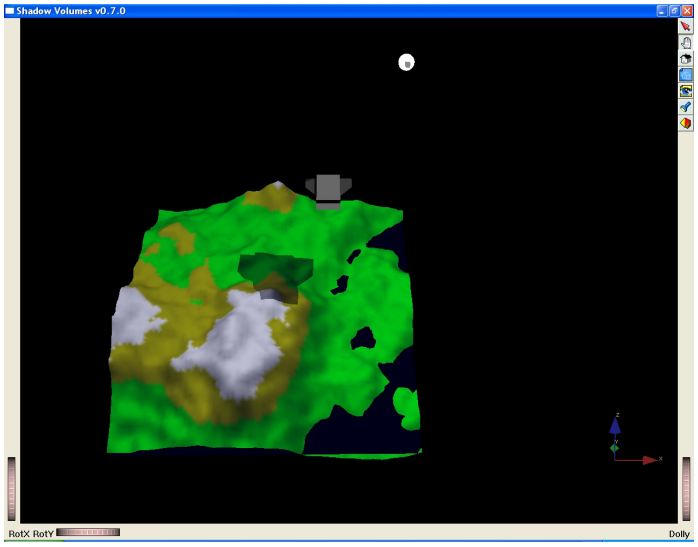
4.2 Demonstrační scéna 1

Tato scéna slouží pro ukázání základních vlastností obou algoritmů. Ve scéně jsou dva objekty umístěné nad sebou v otevřeném boxu, jak ukazuje obrázek 4.1.

4.3 Demonstrační scéna 2

Druhá scéna s jednoduchým modelem rakety nad povrchem krajiny použitým z [6] je ukázkou toho, že stíny se správně zobrazí i na členitém terénu. Ukázka scény je na obrázku 4.2.

Obrázek 4.1: Ukázková scéna 1



Obrázek 4.2: Ukázková scéna 2

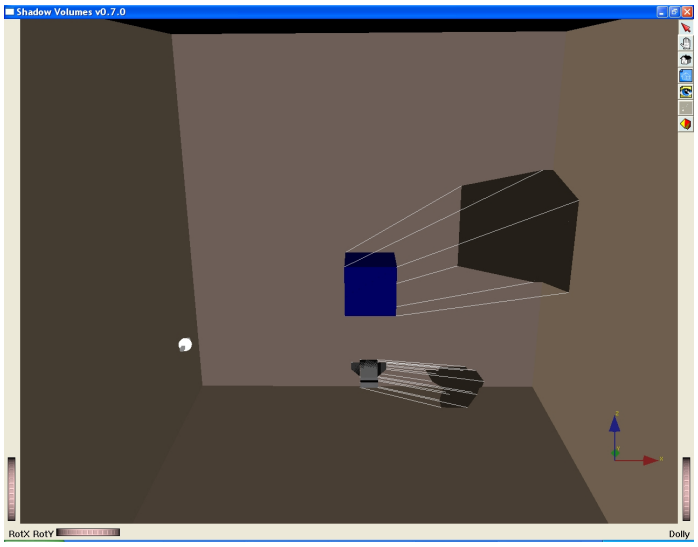
Kapitola 5

Zhodnocení výsledků

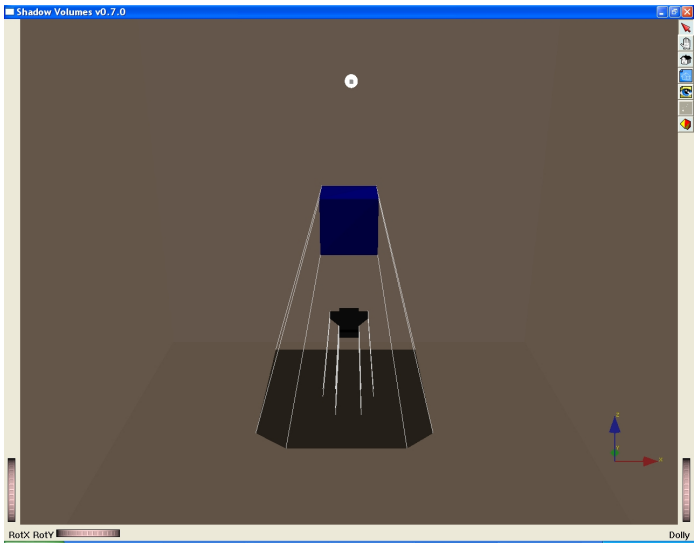
5.1 Depth-pass algoritmus

Stíny vytvořené tímto algoritmem naplnily předpoklady a potvrdily jeho vlastnosti. V první scéně je na stěnách povrchu boxu dobře patrné lámání stínů, viz obrázek 5.1. Na obrázku 5.2 lze pozorovat zastínění jednoho objektu druhým a na modelu rakety i *samozastínění*, obrázek 5.3. Na scéně číslo 2 je dobře vidět, že stíny se správně zobrazí i na členitém terénu, obrázek 5.4. Pro všechny tyto situace algoritmus určil stíny správně.

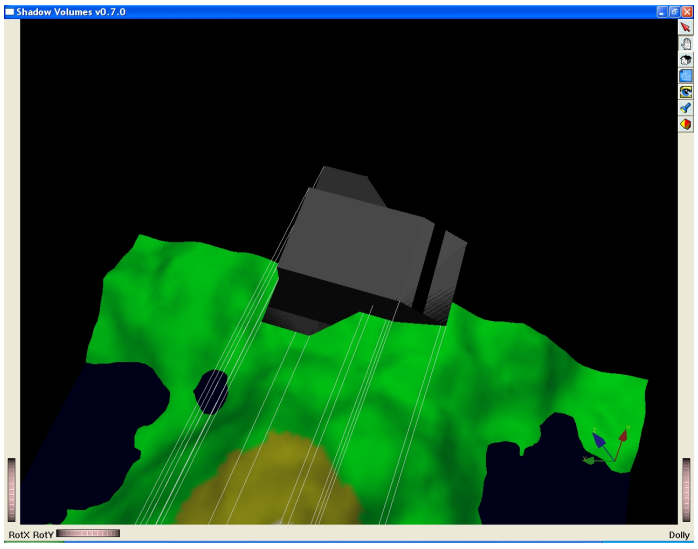
Pouze v případě, že se kamera (přední rovina pohledového objemu) dostane do zastíněné oblasti, dojde díky ořezání stínových těles touto rovinou k špatnému určení stínů. Tento problém ilustruje obrázek 5.5. Algoritmus je i přesto použitelný ve scénách, kdy kamera bude vždy umístěna nad všemi objekty scény nebo bude zaručeno, že se nemůže nacházet ve stínu.



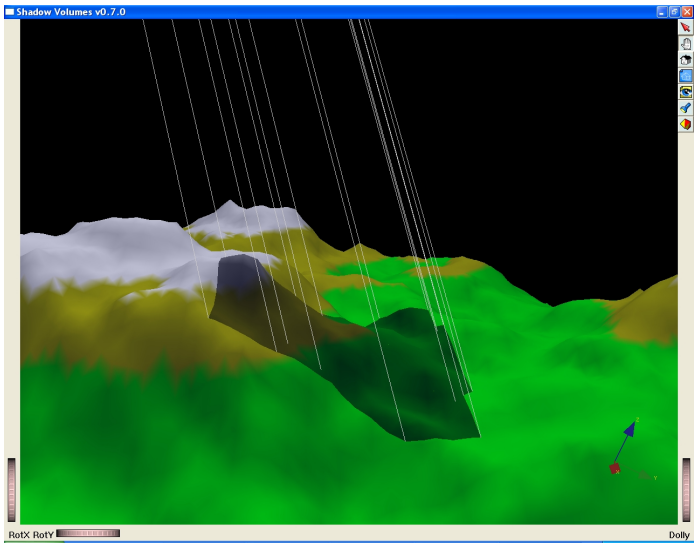
Obrázek 5.1: Scéna 1 (z-pass): lámání stínů na stěnách boxu, ve kterém je scéna umístěna



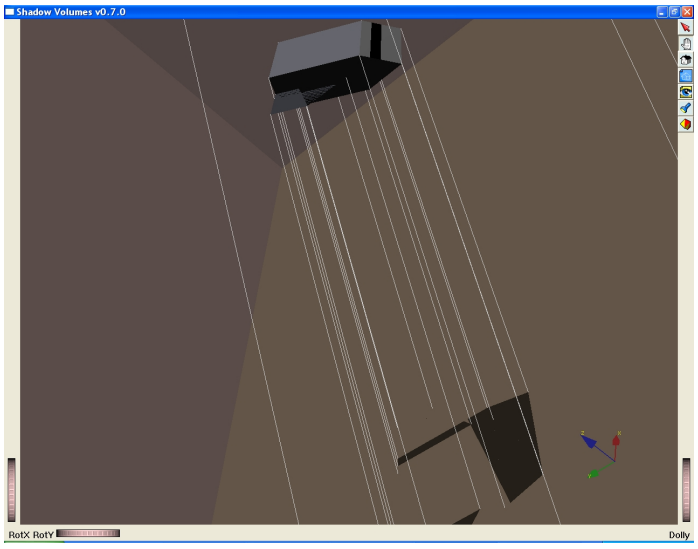
Obrázek 5.2: Scéna 1 (z-pass): zastínění jednoho objektu druhým



Obrázek 5.3: Scéna 2: ukázka vlastních stínů na modelu rakety



Obrázek 5.4: Scéna 2 (z-pass): stíny se zobrazí správně i na členitém terénu krajiny

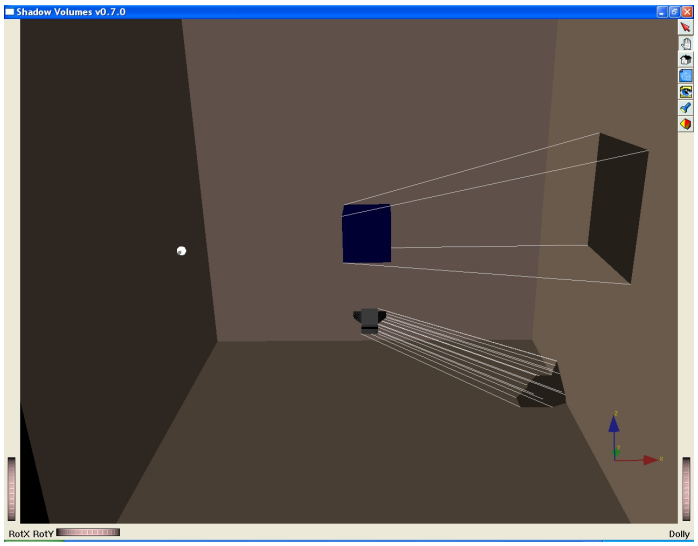


Obrázek 5.5: Scéna 1 (z-pass): špatné určení stínu v případě, kdy se sama kamera nachází ve stínu

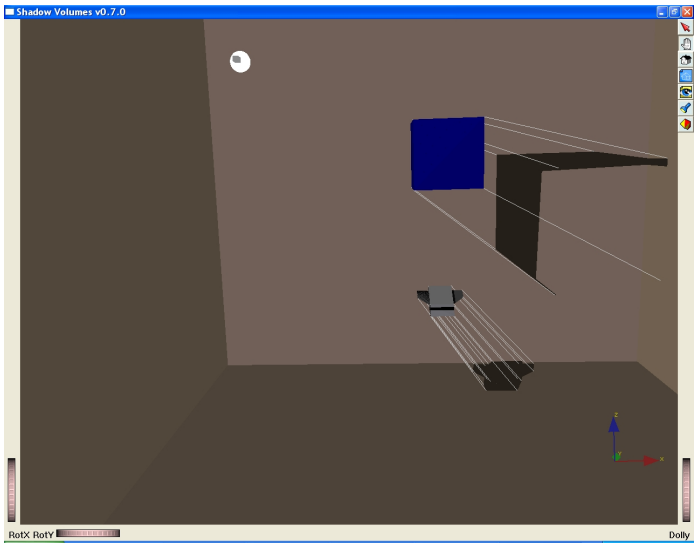
5.2 Depth-fail algoritmus

Výsledky tohoto robustního algoritmu bohužel nesplnily to, co se od nich čekalo. Algoritmus je univerzální a stíny jím generované by měly být správně určené za všech okolností. Implementovaný algoritmus ale zobrazí stíny správně jen v určitých situacích - obrázek 5.6.

Neuniverzálnost implementovaného algoritmu je patrně způsobena malým rozlišením *z-bufferu* a ořezáváním zadního uzávěru stínového tělesa zadní rovinou pohledového objemu, viz obrázek 5.7. Pro další použití je nutné tento problém vyřešit.



Obrázek 5.6: Scéna 1 (z-fail): správně vyřešené stíny lze získat v situacích, kdy nedochází k ořezání stínových těles pohledovým objemem



Obrázek 5.7: Scéna 1 (z-fail): nepřesné určení stínů algoritmem z-fail (stín krychle)

Kapitola 6

Závěr

6.1 Závěr

Ročníkový projekt stručně shrnul současné metody pro zobrazování stínů v počítačové grafice. Z jejich středu byly pro implementaci pod grafickou knihovnou Open Inventor vybrány algoritmy *depth-pass* a *depth-fail* založené na metodě stínových těles. Bylo potvrzeno, že jsou vhodné pro obohacení scén o stíny v rámci Open Inventor API.

Podle získaných výsledků lze konstatovat, že algoritmus *depth-pass* je již reálně použitelný, byť s určitými omezeními. Druhý algoritmus, *depth-fail*, nezobrazuje stíny podle očekávání, a je nutné jeho nedostatky odstranit.

6.2 Příští práce

Další práce se budou především týkat opravením chyb algoritmu *depth-fail* a získáním skutečně robustního algoritmu pro zobrazování stínů. Dále by bylo dobré implementovat v aplikaci oba algoritmy a podle postavení objektů ve scéně vůči světelným zdrojům rozhodnout, zda bude použit na výpočet náročnější *depth-fail* nebo použití *depth-pass* bude dostačující.

Cílem by pak mohlo být vytvoření stínového enginu, který by se postaral správné a výpočetně optimalizované zobrazení stínů ve scéně za všech okolností. Stíny totiž více přibližují počítačem vytvořené scény k reálnému světu.

Literatura

[1] Coin3d. http://coin3d.org.

[2] David Ambrož. Shadows techniques. http://www.shadowstechniques.com/intro_cz.html.

[3] Frank C. Crow. Shadow Algorithm for Computer Graphics. Proceedings of SIGGRAPH '77, vol.11, no. 3, pp. 242-248.

[4] Cass Everitt. Shadow Mapping. http://developer.nvidia.com/attach/6393.

[5] Cass Everitt and Mark J. Kilgard. Optimized Stencil Shadow Volumes. http://developer.nvidia.com, 2003. GDC 2003 presentation.

[6] Ing. Jan Pečiva. Open inventor tutoriál. http://www.root.cz/clanky/open-inventor/.

[7] Jiří Žára, Bedřich Beneš, Jiří Sochor, and Petr Felkel. *Moderní počítačová grafika*. Computer Press, 2004. ISBN 80-251-0454-0.

[8] Josie Wernecke. *The Inventor Mentor*. Addison-Wesley Professional, 1994. ISBN 0201624958.

Obrázky 1.1, 1.2, 2.2, 2.3, 2.4, 2.5, 2.6 převzaty z http://www.shadowstechniques.com.