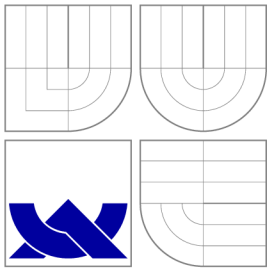
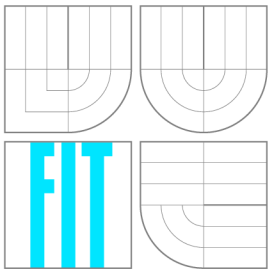


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SLEDOVÁNÍ PAPRSKU POMOCÍ K-D TREE

RAY TRACING USING K-D TREE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR MUSIL
AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ HAVEL
SUPERVISOR

BRNO 2010

Abstrakt

Práce se zabývá návrhem a implementací programu pro syntézu obrazu pomocí sledování parsků. Cílem je ověření vlastností optimalizační metody dělení prostoru pomocí KD stromu. Práce porovnává metody prostorového mediánu, objektového mediánu a cenového modelu použité pro určování dělících rovin při výstavbě KD stromu. Pro vyhodnocení je použito několik testovacích scén.

Abstract

This thesis describes the design and implementation of an application for picture synthesis using ray tracing. The goal is to verify properties of an space subdivision optimalization method using KD tree. The work compares spatial median, objects median and cost model methods for splitting plane determination during KD-tree construction. Several test scenes are used for evaluation.

Klíčová slova

sledování paprsků, dělení prostoru, KD strom, SAH

Keywords

ray tracing, space subdivision, KD tree, SAH

Citace

Petr Musil: Sledování paprsku pomocí k-D tree, bakalářská práce, Brno, FIT VUT v Brně, 2010

Sledování paprsku pomocí k-D tree

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jiřího Havla

.....
Petr Musil
17. května 2010

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Jiřímu Havlovi za jeho vstřícný přístup, ochotu a věcné připomínky, které mi poskytl během řešení této práce.

Obsah

Kapitola 1

Úvod

Sledování paprsků (ray tracing) patří mezi nejpoužívanější metody v oblasti fotorealistické počítačové grafiky. Slouží k převodu 3D scény do 2D rastru, k čemuž využívá fyzikální zákon optiky - šíření světla v prostoru. Tato vlastnost mu umožňuje jednoduše zobrazovat odrazy, lomy světla, stíny objektů a jiné jevy, které se klasickými scanline metodami řeší velice obtížně nebo jsou zcela nemožné. Hlavní výhodou raytracingu je kvalita vyprodukovaného zobrazení. Výpočet tohoto od reality těžko rozpoznatelného zobrazení je ale velice časově náročný a bez různých optimalizací a kvalitních technických prostředků v reálném čase nemožný.

Sledování paprsků řeší problém viditelnosti objektů ve scéně pomocí výpočtu průsečíku paprsků s objekty a nalezení nejbližšího průsečíku. Výpočty těchto průsečíku jsou velice časově náročné a právě to je důvod proč je sledování paprsku tak pomalé. Různé optimalizační metody se zaměřují na zrychlení samotného výpočtu průsečíku nebo na zmenšení počtu výpočtů. Jednou z metod zmenšení počtu výpočtů průsečíku je i dělení prostoru scény. Výsledky experimentů v této oblasti provedené v [4] dokázaly že průměrně nejrychlejší metodou dělení prostoru použitou pro sledování paprsku je právě KD strom.

Hlavním cílem této práce je popsat, navrhnou, implementovat a otestovat algoritmy pro syntézu obrazu pomocí sledování paprsku, které využívají pro svou činost optimalizační metody dělení prostoru pomocí KD stromu. Práce se zaměřuje především na algoritmy, které se používají při výstavbě KD stromu, algoritmy průchodu paprsku KD stromem a možné optimalizace použitelné pro KD stromy.

1.1 Rozvržení kapitol

V druhé kapitole je popsána metoda sledování paprku, její vlastnosti, některé termíny a základní algoritmus. Další čast této kapitoly je zaměřena na možnosti optimalizace sledování paprsku, především na možnosti dělení prostoru scény.

Třetí kapitola popisuje KD stromy. Zaměřuje se detailně na principy používané při výstavbě KD stromu a jejich vliv na kvalitu výsledného stromu, s důrazem na metody založené na teoretickém cenovém modelu. V druhé časti kapitoly jsou popsány algoritmy na průchod paprsku KD stromem a nalezení nejbližšího průsečíku.

Čtvrtá a pátá kapitola se zaměřuje na analýzu, návrh a implementaci programu pro syntézu obrazu pomocí sledování paprsku s využitím dělení prostoru pomocí KD stromu.

Šestá kapitola se věnuje testování naimplementovaných algoritmů, především jejich vlivu na výkon.

V poslední kapitole jsou shrnuty dosažené výsledky a jsou zde zmíněny možnosti dalšího rozšíření práce.

Kapitola 2

Sledování paprsku

Tato kapitola pojednává o použití metody sledování paprsků pro syntézu obrazu. Popisuje její výhody a nevýhody, používané termíny a základní algoritmus. Rovněž se zaměřuje na metody optimalizace sledování paprsku, především na metody dělení prostoru scény.

2.1 Popis

Sledování paprsku (ray tracing) je grafická metoda pro fotorealistickou syntézu obrazu. Byla poprvé představena v roce 1980[10].

Mezi výhody sledování paprsku patří snadné zobrazování některých optických jevů jako je třeba realistický průchod světla přes průsvitný materiál nebo odraz světla. Sledování paprsků je však velice náročné na výkon. Zobrazování složitých scén (řádově desetitisíce a více objektů) neni bez optimalizací a vhodného technického vybavení vůbec možné.

Sledování paprsků řeší problém viditelnosti objektů ve scéně pomocí výpočtu průsečíku paprsků s objekty a nalezení nejbližšího průsečíku. Paprsek je definován jako polopřímka s bodem vzniku a směrem.

Rozlišujeme tyto druhy paprsků:

- Primární paprsek

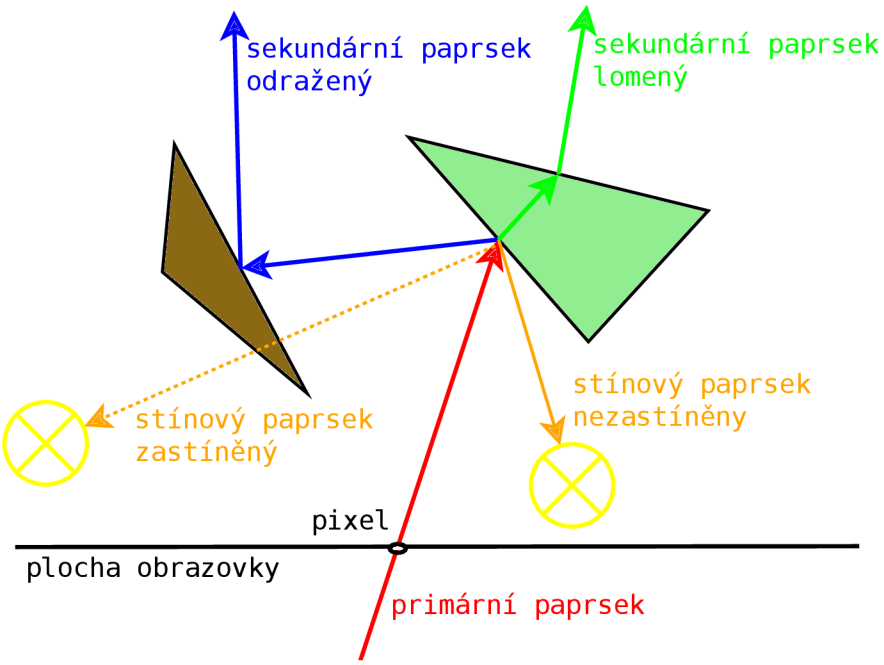
  Šíří se od pozorovatele skrz zobrazovaný bod dál do scény. Jejich počet je tedy stejný jako celkový počet zobrazovaných bodů scény.

- Sekundární paprsek

  Tento paprsek je generován po dopadu primárního nebo sekundárního paprsku na plochu tělesa. Jedná se o odražené paprsky nebo o lomené paprsky, které prošly skrz těleso. Počet těchto paprsků bývá často větší než počet primárních paprsků a je dán především hloubkou rekurze algoritmu a také vlastnostmi materiálů objektů, kdy pro neodrazivé nebo neprůsvitné tělesa nemusíme tyto parsky vytvářet.

- Stínový paprsek

  Jedná se o paprsky, které jsou generované z bodu dopadu primárních nebo sekundárních paprsků do světelných zdrojů. Slouží k určení osvětlení bodu. Pokud neni mezi bodem a světelným zdrojem jiné těleso, je tento bod osvětlen a světelný zdroj je použit při výpočtu osvětlovacího modelu.



Obrázek 2.1: Princip raytracingu a popis paprsků

Důležitou součástí je při syntéze obrazu pomocí sledování paprsku výpočet osvětlení. Nejčastěji je používán Phongův osvětlovací model rozšířený o složku odraženého a lomeného paprsku viz[11].

$$I = I_a + I_d + I_s + I_r + I_t \tag{2.1}$$

Kde jednotlivé složky jsou:

I_a Ambientní složka - vyjadřuje množství okolního světla

I_d Difuzní složka - vyjadřuje množství rozptýleného světla na povrchu. Vychází z Lambertova zákona difuzního odrazu

I_s Zrcadlová složka - vyjadřuje množství zrcadlově odraženého světla

I_r Odrazová složka - množství světla ze směru odrazu od jiného tělesa ve scéně

I_t Lomová složka - množství světla ze směru lomu. Často bývá snižováno koeficientem útlumu závislém na vzdálenosti, kterou paprsek urazí uvnitř tělesa.

2.2 Základní algoritmus

V reálném prostředí se světelné paprsky šíří od zdroje světla, některé zasáhnou povrch určitého objektu ve scéně, od kterého se odrazí, proletí ním nebo jsou pohlceny a jen některé z nich dopadnou na projekční rovinu. Simulace tohoto postupu je skoro nemožná z důvodu extrémně vysoké výpočetní náročnosti, kdy se ze zdroje světla šíří skoro nekonečný počet světelných paprsků, ale jen málo z nich ve skutečnosti dopadne na projekční rovinu. Proto je u sledování paprsku použit reverzivní postup, kdy paprsek promítneme od pozorovatele skrz projekční rovinu do scény a zkoumáme, s kterým nejbližším objektem se střetne a v jakém bodě. Určíme zda-li je tento bod osvětlen zdrojem světla a z vlastností tohoto osvětlení a vlastností objektu vypočteme dílčí barvu tohoto bodu.

Základní algoritmus využívá rekurzivní funkce pro výpočet jednoho paprsku, která je volána pro každý zobrazovaný bod.

```
Barva Raytrace( Paprsek, Hloubka ) {
    Průsečík = vypočítej průsečík Paprsku s každým tělesem a ulož nejbližší
    if( !Průsečík )
        Paprsek opustil scénu
        return defaultní barvu
    pro každé Světlo ve scéně
        SP = vytvoř stínový paprsek z Průsečíku do Světla
        vypočítej průsečík SP s každým tělesem scény
        pokud existuje průsečík, který je blíž než Světlo
            označ Světlo za zastíněné
    Barva = vypočti osvětlovací model se všemy nezastíněnými světly
    if( Hloubka ) {
        SP = vytvoř sekundární odražený paprsek
        Barva +=Raytrace( SP, Hloubka-1 )
        SP = vytvoř lomený paprsek
        Barva += Raytrace( SP, Hloubka-1 )
    }
    return Barva
}
```

Tento základní algoritmus ray tracingu je poměrně jednoduchý, ale trpí nedostatky v oblasti výkonu a časové náročnosti. Důvodem je výpočet průsečíku paprsku s objektem, který je použit pro určení nejbližšího průsečíku a při určování zastínění světla. V případě nejbližšího průsečíku testuje navzájem všechny primární a sekundární paprsky s každým objektem a v při výpočtu zastínění testuje každý stínový paprsek s objekty, dokud nenajde stínící objekt. Časová složitost je tedy lineární a závisí na počtu objektů ve scéně, počtu světel a také na požadované hloubce rekurze.

2.3 Urychlování sledování paprsku

Urychlování sledování paprsků je velice důležité, protože jednoduchý algoritmus raytracingu je velice časově náročný. Podle [10] spotřebuje počítání průsečíku paprsku s objektem až 95% strojového času z celkové doby zobrazení. Proto se optimalizační algoritmy zaměřují právě na výpočet průsečíků. Při použití optimalizačních metod můžeme urychlit výpočty o dva až tři řády. Urychlovací metody se často vhodně kombinují.

Podle [11] rozdělujeme metody urychlování na:

- Urychlení výpočtu průsečíků
  - Rychlejší výpočet průsečíku
    Použití prediktorů průsečíku před samotným výpočtem bodu, předvypočítané hodnoty, lepší analytické metody, jednoduché obálky
  - Zmenšení počtu výpočetů průsečíku
    Dělení prostoru, koherence paprsků, paměť překážek
- Snížení počtu paprsků
  Adaptivní podvzorkování, řízení hloubky rekurze

- Sledování více paprsků naráz

  Paralelní zpracování, využití SIMD jednotek nebo GPGPU

V následujících odstavcích se budu věnovat metodám urychlování sledování paprsku pomocí dělení prostoru, mezi které patří i hlavní téma práce - KD stromy.

Dělení prostoru umožňuje vhodně uspořádat informace o scéně tak, abychom byli schopni určit, které objekty se nacházejí v dané prostorové oblasti. Pro určení nejbližšího průsečíku a pro určování zastínění světla tedy nepracujeme se všemy objekty ve scéně, ale jen s objekty, které se nacházejí v prostorové oblasti kterou paprsek prochází. Tím je redukován počet výpočtů průsečíků a dochází k urychlení celého procesu. Dělení prostoru sebou ale na druhou stranu přináší nutnost zavedení pomocných struktur, které se vytvářejí při načítání scény a uchovávají o ní informace.

Rozlišujeme tyto dva druhy dělení prostoru:

- Uniformní

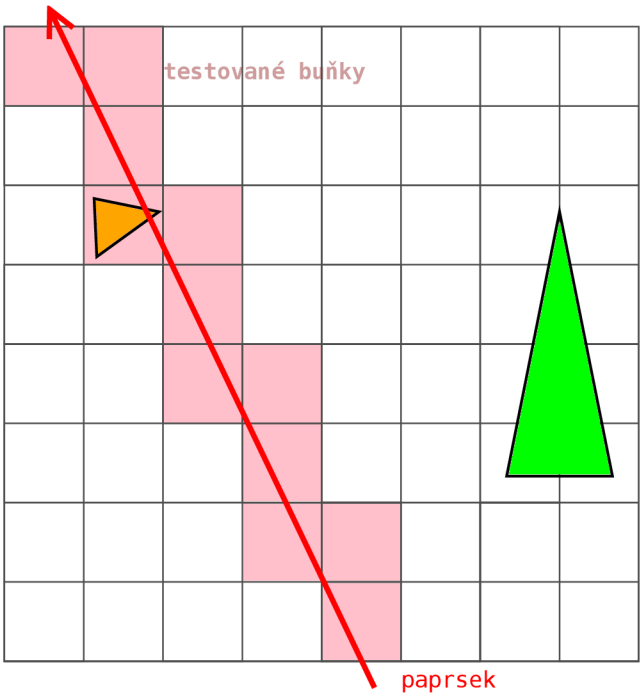
  Dělení na stejně velké buňky

- Neuniformní

  Velikost buňky se mění adaptivně

2.3.1 Uniformní dělení prostoru

Jde o rozdělení prostoru do buněk o stejné velikosti, které jsou kolmé na souřadnicový systém. Buňky mohou být prázdné nebo obsahovat i více objektů, přitom objekt může zasahovat do více buněk. Vzhledem k tomu, že má každá buňka pevně definované své sousedy, je jednoduché a rychlé přes ně procházet. Hlavními nevýhodou je to, že se rozdělení nepřizpůsobuje testované scéně, a proto vzniká poměrně hodně buněk, které neobsahují žádný objekt a jen zabírají místo v paměti. Proto má uniformní dělení prostoru velké paměťové nároky a neefektivně využívá pamět.



Obrázek 2.2: Uniformní dělení prostoru

Pro průchod paprsku přes uniformě rozdělený prostor se používá algoritmus 3D-DDA, což je analogie na Bresenhamův algoritmus pro kreslení přímky v 2D prostoru. Uniformní dělení prostoru neni vhodné pro řídce obsazené scény, což jsou scény, kde se většina objektů nachází v malém prostoru a rozsáhlé oblasti scény jsou prázdné. Protože je nutné při průchodu zpracovávat velké množství prázdných buněk a také buňky, které naopak obsahují poměrně velký počet objektů, může dojít v nejhorším případě i k tomu, že úspora času při použití uniformního dělení prostoru nemusí být významná viz [4].

Typickým představitelem uniformního dělení prostoru je metoda dělení prostoru do pravidelné mřížky.

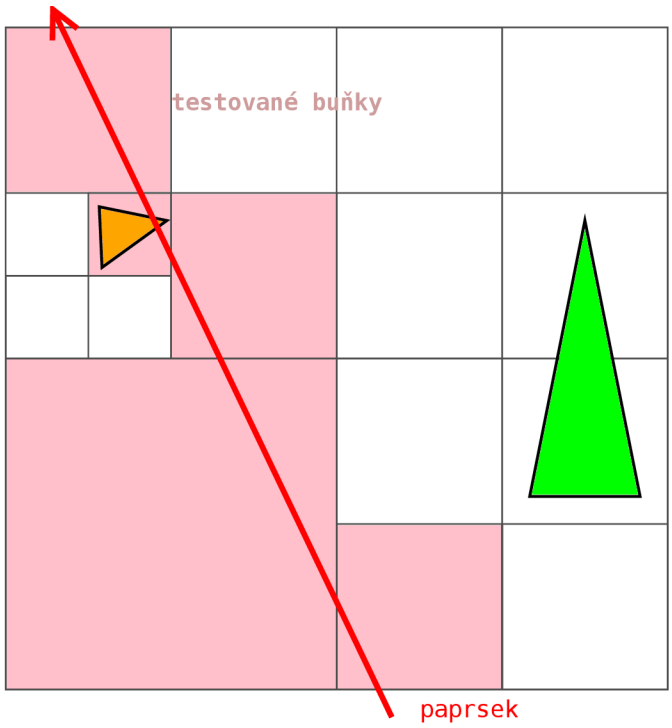
2.3.2 Neuniformní dělení prostoru

Jedná se o rozdělení prostoru do buněk, které adaptivně mění svoji velikost v závislosti na rozložení objektů ve scéně. Obecně lze říci, že toto rozdělení lépe pracuje s pamětí a provádí méně kroků výpočtu než uniformní dělení prostoru. Na druhou stranu ale potřebuje složitější datové struktury většinou stromového typu, jejíž implementace a efektivní průchod paprsku je poměrně složitý.

Nejčastěji používanými strukturami jsou Octree a BSP stromy, hlavně jejich speciální případ - KD strom. V následujících odstavcích se budu věnovat těmto strukturám s tím, že KD stromům věnuji celou nasledující kapitolu.

Octree

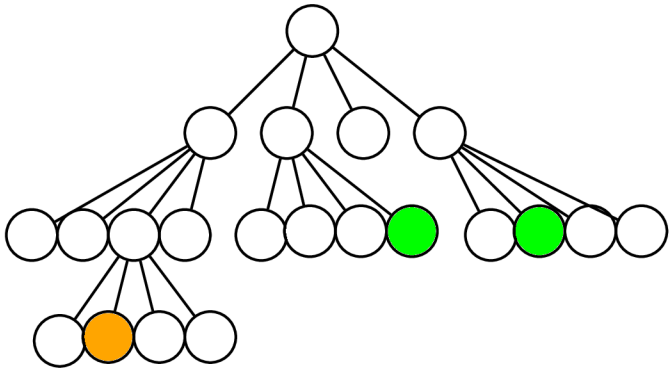
Octree (Octal tree) je struktura uchovávající rozdělení 3D prostoru, jejíž obdobou v 2D prostoru je Quadtree. Je to nejčastěji stromová nebo hašhovaná struktura buněk kolmých k souřadnicovému systému, přitom platí, že každá buňka se může rozdělit na osm menších stejně velkých buněk (oktanů).



Obrázek 2.3: Quadtree (obdoba Octree ve 2D)

Při výstavbě Octree struktury jako stromu vycházíme z kořenového uzlu, který ukazuje na celý prostor. Prostor rozdělíme na osm stejných částí - listů (oktanů), které navážeme na nadřazený (kořenový) uzel. Toto provádíme rekurzivně, za předpokladu ukončujících podmínek, kterými jsou maximální hloubka stromu a minimální počet odkazů na objekt v listu.

Průchod paprskem přes Octree strukturu je mnohem komplikovanější než v případě uniformního rozdělení prostoru. Pokud vezmeme v úvahu buňku rozdělenou na oktany, může paprsek procházet vždy přes maximálně čtyři oktany. Je tedy nutné určit, které a v jakém pořadí viz [4]. Pro průchod se používá Glassnerův algoritmus.



Obrázek 2.4: Stromová struktůra Quadtree podle obrázku 2.3

Úsporu času při použití Octree lze obecně srovnávat s BSP stromy, až na případy řídce obsazené scény, kdy může vznikat velký počet prázdných buněk, které zabírají místo v paměti a musí se s nimi počítat při průchodu paprskem.

Existuje varianta Octree nazvaná Octree-R, která dělí buňku také na oktany, ale s různou velikostí. Tato vlastnost zrychluje průchod paprsku, ovšem na úkor výrazně složitější implementace.

BSP stromy

BSP strom (Binary space partitioning tree) je metoda dělení prostoru, která byla původně vymyšlena pro řešení viditelnosti v počítačové grafice. Je to stromová struktura, která není nutně založena na obdelníkovém rozložení prostoru. Prostor je rekurzivně dělen na dva podprostory pomocí dělící roviny. Důležitou vlastností BSP stromů je, že se velice dobře přizpůsobují geometrii scény.

U BSP stromu rozlišujeme dva druhy dělící roviny:

- polygonálně sladěnou (polygon-aligned)

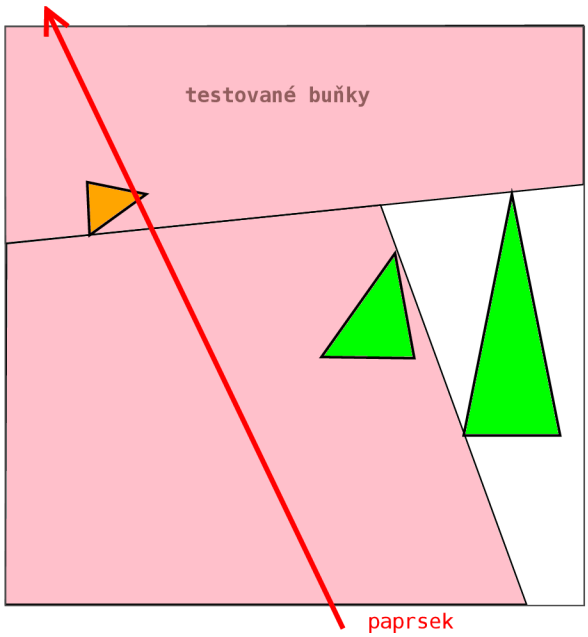
  Dělící rovina se určuje pomocí ploch procházejících polygony ve scéně. Vyžaduje však, aby byla scéna vytvořena jen z polygonů, což je příliš omezující, a proto se u sledování paprsku nepoužívá viz [6]

- osově sladěnou (axis-aligned)

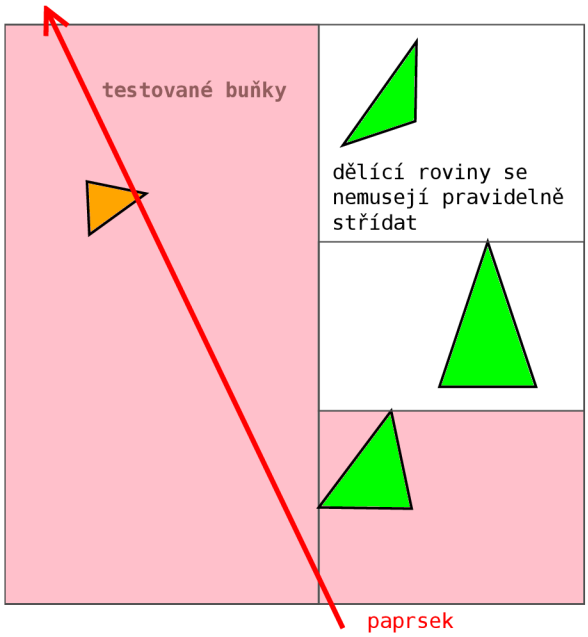
  Výsledný strom je někdy také nazýván jako ortogonální BSP strom. Dělící rovina je vždy rovnoběžná s některou z os souřadnicového systému. Speciálním případem je KD strom, kde se pravidelně střídají dělící roviny viz[11].

BSP stromy se obvykle konstruují hierarchicky zhora dolů. Při výstavbě se používá rekurzivní funkce, která dělí prostor na dva podprostory za předpokladu ukončujících podmínek, kterými jsou maximální hloubka stromu a minimální počet odkazů na objekt v listu. Při rozdělování na podprostory je využívá heuristická funkce pro určení vhodné dělící roviny, kterou si daný nelistový uzel ukládá.

Existuje několik algoritmů pro průchod paprsku přes obecný BSP strom. Většina z nich využívá pro svoji funkci zásobník nebo jsou rekurzivní. Jejich implementace je poměrně složitá. Při porovnání efektivnosti algoritmů průchodu pro polygon-aligned a axis-aligned je axis-aligned při použití pro průchod paprsku podle[4] asi třikrát efektivnější.



Obrázek 2.5: Jedna z možných konstrukcí BSP, dělící rovina je polygon-aligned



Obrázek 2.6: Jedna z možných konstrukcí BSP, dělící rovina je axis-aligned

Kapitola 3

KD - tree

Tato kapitola pojednává o KD stromech, jako o optimalizační metodě používané při sledování paprsků. Zaměřuje se detailně na principy používané při výstavbě KD stromu a jejich vlastnostech na kvalitu výsledného stromu s důrazem na metody založené na teoretickém cenovém modelu. Popisuje také algoritmy na průchod paprsku KD stromem a nalezení nejbližšího průsečíku a věnuje se popisu struktur pro uložení KD stromu v paměti.

3.1 Popis

V[4] je porovnáno 12 odlišných algoritmů dělení prostoru používaných při vrhání paprsků pro různé druhy scén. Z této práce vyplývá, že algoritmus, který využívá KD strom je průměrně nejrychlejší.

KD strom (k dimensionální strom) je struktura pro rozdělení prostoru. Je to speciální případ ortogonálního BSP stromu, ve kterém se pravidelně střídají dělící roviny[11]. Používá se pro určení nejbližšího souseda a v počítačové grafice při sledování paprsků.

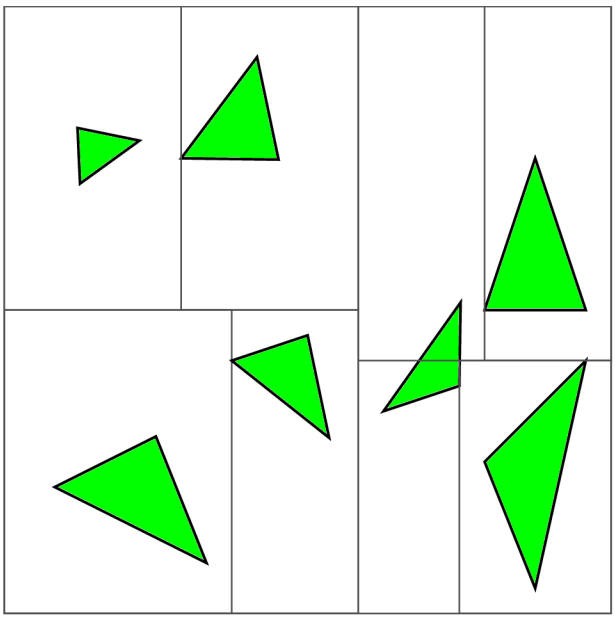
Jak už z názvu vyplývá, je to stromová struktura. Kořen stromu zahrnuje celý prostor scény. Prostor musí být vždy pravoúhlý a rovnoběžný se souřadnicovým systémem. Každý nelistový uzel stromu dělí prostor na dva menší pomocí roviny, která je rovnoběžná s některou z os souřadnicového systému. Uzel si pamatuje bod, ve kterém je rovina dělena. Objekty, jež se nacházejí ve scéně, se uchovávají pouze v listových uzlech a to nejčastěji ve formě lineárního seznamu. Listové uzly mohou být i prázdné, nemusejí obsahovat žádný objekt.

Použití KD stromu na sledování paprsku je vhodné především u statických (neměných) scén viz [4][8]. Důvodem je to, že strom musíme před použitím pro sledování paprsku vystavět, což v případě statické scény stačí pouze jednou před začátkem vykreslování. U dynamické scény bychom museli po každé změně ve scéně přestavět část nebo i celý strom. Doba výstavby stromu je sice z celkové doby syntézy obrazu scény poměrně malá, ale i tak nás u dynamických scén dost omezuje.

Když porovnáme obecný BSP strom a jeho speciální případ KD strom při sledování paprsků, zjistíme že BSP stromy o něco lépe zohledňují rozložení prostoru. Na druhou stranu ale vyžadují při výstavbě a hlavně při trasování paprsku složité a neefektivní algoritmy viz [4], proto jsou ve výsledku většinou pomalejší než KD stromy, které díky svým omezením mohou používat vysoce efektivní algoritmy.

Analyticky by mělo použití KD stromu pro sledování paprsku změnit časovou složitost závislou na počtu objektů z lineární na logaritmickou. Tento předpoklad však nebyl zcela

potvrzen viz [4].



Obrázek 3.1: Jedna z možných variant KD stromu ve 2D prostoru

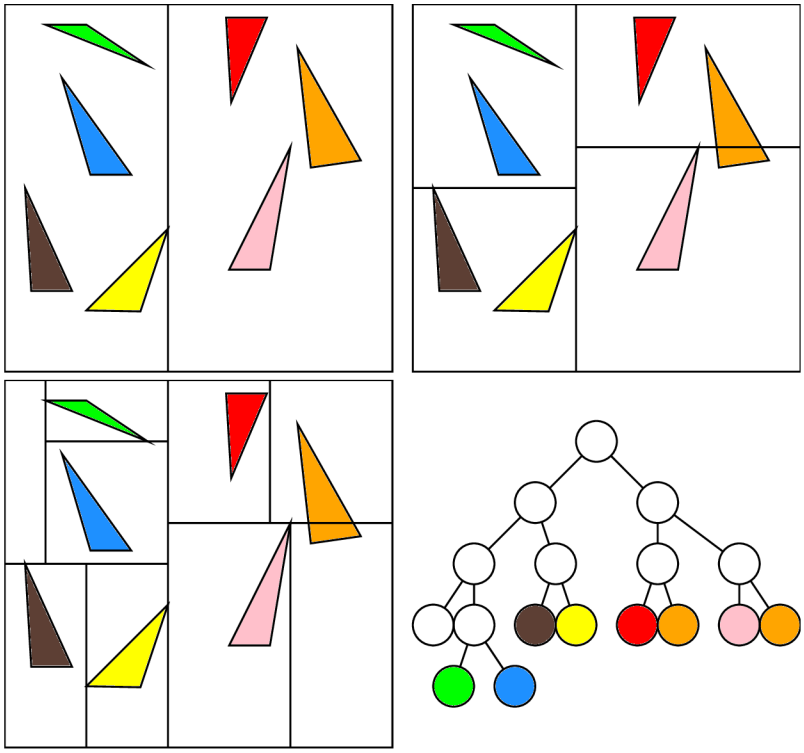
3.2 Konstrukce

Dobrá konstrukce KD stromu je velice důležitá. Obecně platí, že čím větší práci si dáme s výstavbou stromu, především s určením pozice dělících rovin, tím rychlejší bude poté syntéza obrazu pomocí sledování paprsku.

KD stromy se vytvářejí od kořene k listům, většinou pomocí rekurzivní funkce, která dělí prostor vždy na dvě poloviny. Rekurze je ukončena při dosažení maximální hloubky stromu nebo pokud je v prostoru uzlu méně objektů než je minimální požadovaný počet objektů anebo pokud je prostor prázdný.

Nejprve je vytvořen uzel, který bude označen jako kořenový. Na tento uzel je zavolána rekurzivní funkce, které je předán jako parametr seznam všech objektů ve scéně. Rekurzivní funkce nejprve najde vhodnou dělící rovinu s tím, že její směr je znám, protože pro KD strom platí, že se směr dělících rovin pravidelně střídá. Objekty jsou rozděleny podle dělící roviny na ty co leží před, za a na dělící rovině. Jsou vytvořeny dva synovské uzly (levý a pravý) a je na ně zavolána rekurzivní funkce, které jsou jako parametry předány pro pravý uzel objekty, které leží za a na dělící rovině a pro levý uzel objekty, ležící před a na dělící rovině. Rekurze pokračuje dokud nejsou naplněny podmínky pro její ukončení. V tomto případě již není uzel dále dělen, ale je označen jako list a uloží si seznam objektů nacházejících se v jeho prostoru, tedy ty, které mu byly předány jako parametr.

Obrázek 3.2 ukazuje konstrukci KD stromu. Pro zjednodušení je nákres tvořen ve dvojrozměrném prostoru, v případě trojrozměrného prostoru by to však bylo obdobné. Je vidět, že pro hloubku zanoření dva v pravém podprostoru prochází dělící rovina přes oranžový trojúhelník. K jeho rozlišení od červeného a růžového trojúhelníkU dochází až v následující iteraci rekurze. Důsledkem toho, že oranžovým trojúhelníkem prochází dělící rovina je to, že je ve výsledném stromu obsažen ve více než jednom listu.



Obrázek 3.2: Příklad konstrukce KD stromu ve 2D prostoru

Rekurzivní algoritmus konstrukce KD stromu, upraveno z[6]:

```
BuildKDTree() {
    kořen = new KDUzel
    objekty = všechny objekty scény
    kořen.Subdivide(objekty, Maximální hloubka)
}

Subdivide(objekty, hloubka) {
    if( !Hloubka nebo objekty.count() ≤ MINIMUM){
        označ uzel za list a ulož objekty
        return
    }
    rovina = urči dělící rovinu
    objektyVLevo, objektyVPravo = vytvoř seznam objektu
    while(pro všechny objekty) {
        if(objekt leží za rovina)
            objektyVPravo+=objekt
        if(objekt leží před rovina)
            objektyVLevo+=objekt
        if(objekt leží na rovina) {
            objektyVPravo+=objekt
            objektyVLevo+=objekt
    }
    levýUzel = new KDUzel
    pravýUzel = new KDUzel
```

```
    levýUzel.Subdivide(objektyVLevo, hloubka-1)
    pravýUzel.Subdivide(objektyVPravo, hloubka-1)
}
```

Při výstavbě KD stromu řešíme problém,jak najít vhodnou dělící rovinu, která dělí prostor na dva podprostory. Směr roviny známe, musíme však určít její polohu. Existuje několik metod na určení této polohy viz[4]:

- Prostorový medián

  Rozdělení prostoru vždy ve středu na stejné poloviny. Jednoduchá a rychlá metoda, která vede k prostorově vyváženému stromu. Výsledný strom připomíná Octree. Nezohledňuje však úplně nejlépe rozložení objektů ve scéně. V porovnání s ostatními metodami vzniká dost prázdných listů a také dost objektů, které jsou umístěny ve více listech.

- Objektový medián

  Snaží se rozdělit prostor tak, aby na obou stranách byl stejný počet objektů. Rychlost nalezení roviny má logaritmickou složitost závislou na počtu objektů[4]. Rozložení prostoru zohledňuje lépe než prostorový medián. Mohla by se zdát ideální metodou, protože vytváří vyvážené stromy, které jsou důležité například u binárního vyhledávacího stromu. Tento předpoklad ale neplatí při trasování paprsku a metoda má výkonové nedostatky v porovnání s jinými metodami. Důvodem je, že metoda vůbec nezohledňuje rozměry prostoru.

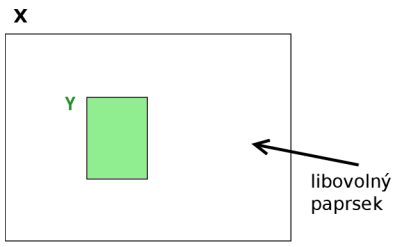
- Cenový model

  Obecně poskytuje nejlepší výsledky pro sledování paprsku, proto ho popíši v detailech níže.

3.2.1 Cenový model

Metoda je založena na teoretickém cenovém modelu, který při výstavbě KD stromu odhaduje průměrnou cenu na průchod libovolného paprsku přes tento strom. Byl představen MacDonaldem a Boothem v roce 1989[4].

Cena zohledňuje náklady na průchod stromem a náklady na výpočet průsečíku paprsku s objektem. Cenový model musí být použitelný na scény s různorodým rozložením objektů v prostoru. Proto si rozložení objektů zjednodušuje pomocí geometrické pravděpodobnosti.



Obrázek 3.3: Ukázkové rozložení prostorů pro výpočet cenového modelu

Předpokládejme, že máme dva prostory X a Y a platí, že Y je obsaženo v X. Situace je znázorněna na obrázku 3.3. Potom pravděpodobnost, že paprsek libovolného směru a pozice procházející prostorem X současně prochází i prostorem Y je:

$$P_{[Y|X]} = \frac{SA(Y)}{SA(X)} \tag{3.1}$$

Kde funkce $SA()$ vyjadřuje plochu prostoru. Obecně to tedy znamená, že čím větší poměrnou plochu zabírá prostor Y v prostoru X, tím je vyšší pravděpodobnost, že libovolný paprsek při průchodu prostorem X prochází i prostorem Y. Pro třírozměrný prostor tedy platí:

$$P_{[Y|X]} = \frac{Y_H * Y_W + Y_W * Y_D + Y_D * Y_H}{X_H * X_W + X_W * X_D + X_D * X_H} \tag{3.2}$$

Těchto vlastností využívá metoda nazvaná SAH (Surface Area Heuristic) viz [4]. Je to heuristická metoda, která se snaží maximalizovat volný prostor nejlépe v blízkosti kořene stromu. Vychází z výše zmíněné vlastnosti o pravděpodobnosti průchodu libovolného paprsku podprostorem. Hlavní myšlenkou je vytvářet velké podprostory, které obsahují malé množství objektů nebo jsou zcela prázdné a malé podprostory, obsahující velké množství objektů. Při trasování paprsku bude tedy paprsek s vyšší pravděpodobností procházet prostory s málo objekty (bude počítat málo průsečíku) a s nižší pravděpodobností prostory s hodně objekty (bude počítat hodně průsečíků). Z uvedeného vyplývá, že je výrazně minimalizován počet výpočtů průsečíku.

Cenu rozděleného uzlu vypočítame metodou SAH jako viz [9][5]:

$$C_v(p) = \kappa_T + \kappa_I \left[P_{[V_L|V]} N_L + P_{[V_P|V]} N_P\right] \tag{3.3}$$

a cenu nerozděleného uzlu jako:

$$C_v(\ ) = \kappa_I N_V \tag{3.4}$$

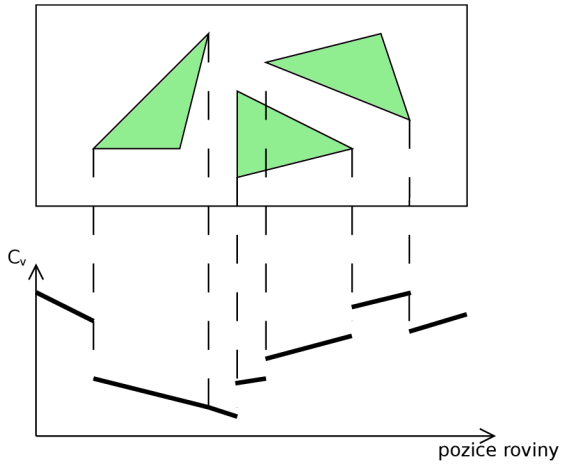
kde:

- $C_v(p)$ je vypočtená cena uzlu po rozdělení rovinou p
- $C_v(\ )$ je cena nerozděleného uzlu
- κ_T je cena pro zanoření do nasledující hloubky KD stromu. Je nutno ji experimentálně určit pro konkrétní implementaci.
- κ_I je cena pro výpočet průsečíku. Je nutno ji experimentálně určit pro konkrétní implementaci.
- $P_{[V_L|V]}$ je pravděpodobnost, že libovolný paprsek procházející prostorem uzlu bude procházet levým podprostorem, který je vytvořen dělením prostoru pomocí roviny p
- $P_{[V_P|V]}$ je pravděpodobnost, že libovolný paprsek procházející prostorem uzlu bude procházet pravým podprostorem, který je vytvořen dělením prostoru pomocí roviny p

- N_L je počet objektů, které se budou po rozdělení pomocí roviny p nacházev v levém podprostoru
- N_P je počet objektů, které se budou po rozdělení pomocí roviny p nacházev v pravém podprostoru
- N_V je počet všech objektů v uzlu

Nejlepší pozice dělící roviny je podle metody SAH ta, která ma nejnižší cenu. Zároveň však platí, že není vhodné uzel dále dělit, pokud je cena nerozděleného uzlu nižší než je nejnižší cena rozděleného uzlu. Tato podmínka zabraňuje vytváření uzlů, které by při průchodu paprsku již nebyli efektivní.

Při určování vhodné pozice dělící roviny řešíme problém, v které pozici je cena nejnižší. Nemůžeme zkoumat všechny možnosti pozice, protože jejich počet je nekonečný. Pokud si však vykreslíme závislost ceny na pozici roviny pro jednoduchý případ scény, což ukazuje obrázek 3.4 zjistíme, že výsledkem je lineární funkce nespojitá v místech hranic objektů. Právě v těchto bodech má funkce nejnižší hodnoty, a proto jsou vhodné pro výpočet nejnižší ceny.



Obrázek 3.4: Závislost ceny uzlu na pozici dělící roviny pomocí SAH

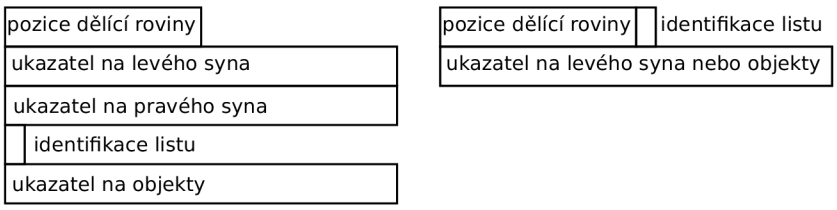
Existují i jiné heuristické metody na určení ceny rozdělení. Jsou to obvykle specializované metody jako třeba PERSAH, který je určený pro zobrazování pomocí perspektivní projekce, PARSAH pro paralerní projekci nebo SPHSAH pro sférickou projekci. Jejich přesnější popis je však již nad rámec této práce.

3.3 Uspořádání paměti

Správné uspořádání uzlů KD stromu v paměti může zvýšit rychlost při sledování paprsku v řádu desítek procent. Důvodem je hirarchie pamětí počítače. Nejrychlejší paměť (cache), která se nachází přímo u procesoru, má jen omezenou kapacitu. Přístup do ostatní paměti s vyšší kapacitou je mnohdy mnohonásobně pomalejší. Proto se snažíme do cache paměti načíst co nejvíce spolusouvisejících uzlů. Zároveň je vhodné, aby datová struktura popisující uzel byla co možná nejmenší a byla správně zarovnaná v paměti.

Datová struktura popisujíci uzel obsahuje většinou neceločíselný údaj o pozici dělící roviny a ukazatele na dva synovské uzly. Musíme počítat s možností, že se uzel stane listem,

proto musí ještě dále obsahovat identifikaci toho jestli se jedná o list a ukazatel na seznam objektů.



Obrázek 3.5: Ukázka neoptimalizované a optimalizované struktury uzlu

Jak je vidět z obrázku 3.5, optimalizovaná a vhodně zarovnaná struktura uzlu je 2,5 krát menší než původní struktura. Toho je dosaženo tím, že při alokování paměti pro synovské uzly byly tyto uzly umístěny vedle sebe. Na levý uzel se tedy dostaneme přímo přes ukazatel a na pravý přičtením velikosti datové struktury k ukazateli. Rovněž je využito toho, že pokud se uzel stane listem, tak již nemá žádné synovské uzly, a proto zbylý ukazatel použijeme jako ukazatel na seznam objektů.

3.4 Průchod paprsku KD stromem

Algoritmus trasování paprsku slouží k průchodu paprsku přes KD strom za účelem nalezení nejbližšího průsečíku. Algoritmy dělíme dle[4] na:

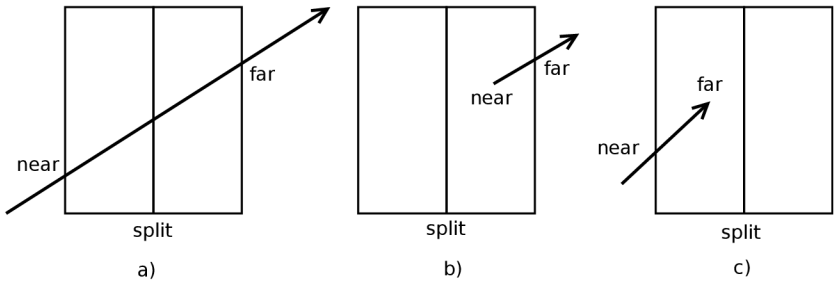
- sekvenční - je to původní jednoduchá metoda, ve které však může při trasování jednoho paprsku algoritmus navštívit jednu buňku i vícekrát než jednou. Proto se pro svoji nízkou efektivitu už nepoužívá.
- rekurzivní
- iterativní

Rekurzivní a iterativní přístup se používá nejčastěji, proto je blíže popíši.

3.4.1 Rekurzivní algoritmus

Využívá rekurzivní funkce pro sestup stromem. Poměrně efektivní metoda, ale vyžaduje neustále volání rekurzivní funkce, což není optimální.

Algoritmus nejprve najde vstupní a výstupní bod. To jsou body, ve kterých paprsek vstupuje a vystupuje ze scény. Pokud však výchozí bod paprsku leží uvnitř scény, je vstupní bod přepočten na výchozí bod paprsku. Podobně u výstupního bodu, pokud leží dál než konečný bod, tak je do tohoto bodu posunut.



Obrázek 3.6: Určení vstupního a výstupního bodu v buňce

Důležitá je pozice vstupního a výstupního bodu vůči pozici dělící roviny. Pokud leží vstupní bod až za dělící rovinou znamená to, že paprsek neprochází jedním podprostorem, a proto ho nemusíme při hledání průsečíku zpracovávat(obrázek 3.6b). To samé platí i v případě, že výstupní bod leží před dělící rovinou(obrázek 3.6c). V případě, že vstupní bod leží před a výstupní za dělící rovinou, musíme zpracovat oba podprostory(obrázek 3.6a).

Samotný algoritmus využívá rekurzivní funkce, která je zavolána na kořen. Funkce nejprve určí vstupní a výstupní bod prostoru a podle předchozích pravidel se rekurzivně volá pro jeden nebo oba podprostory. Pokud funkce sestoupí do listového uzlu, pokusí se najít průsečík objektů v listu s paprskem. Je-li nalezen průsečík, je vybrán nejbližší a je vrácen.

Rekurzivní algoritmus průchodu paprsku KD stromem upraveno z [4]:

```
GetFirstIntersection( paprsek, vzdálenost) {
    tnear, tfar = urči vstupní a výstupní bod scény
    if(tnear ≤ 0) near = 0
    if(tfar ≥ vzdálenost) far = vzdálenost
    return LocateLeaf(paprsek, tnear, tfar, kořen)
}

LocateLeaf(paprsek, tnear, tfar, uzel) {
    if( uzel je list ){
        otestuj všechny objekty v listu na průsečík s paprskem
        pokud průsečík existuje, vrať najbližší jinak vrať žádný
    }
    near a far = synovské uzly podle směru paprsku
    if(tnear > split nebo tfar < split ){
        rozgenerujeme pouze jeden podprostor
        LocateLeaf(paprsek, tnear, tfar, near)
        return  průsečík,pokud byl nalezen
    }
    else{
        rozgenerujeme oba podprostory
        LocateLeaf(paprsek, tnear, tfar, near)
        return  průsečík,pokud byl nalezen
        LocateLeaf(paprsek, tnear, tfar, far)
        return  průsečík,pokud byl nalezen
    }
}
```

3.4.2 Iterativní algoritmus

Využívá stejný princip vstupních a výstupních bodů jako předchozí rekurzivní algoritmus. Rekurzivní zanořování je nahrazeno iterací se zásobníkem. Často bývá efektivnější než rekurzivní algoritmus, protože nevyžaduje neustále volání funkcí.

Iterativní algoritmus průchodu paprsku KD stromem upraveno z [4]:

```
GetFirstIntersection( paprsek, vzdálenost) {
    tnear, tfar = urči vstupní a výstupní bod scény
    if(tnear ≤ 0) near = 0
    if(tfar ≥ vzdálenost) far = vzdálenost
```

```
    uzel = kořen
    while(1){
        if( uzel neni list ) {
            near a far = synovské uzly podle směru paprsku
            if(tnear > split nebo tfar < split ){
                rozgenerujeme pouze jeden podprostor
                uzel = near
            }
            else{
                rozgenerujeme oba podprostory
                uzel = near
                vlož far na vrchol zásobníku
            }
        }
        else{
            uzel je list
            otestuj všechny objekty v listu na průsečík s paprskem
            pokud průsečík existuje, vrať najbližší
        }
    pokud je zásobník prázdný vrať průsečík nenalezen
    uzel = vyndej z vrcholu zásobníku
    }
}
```

Kapitola 4

Analyza a návrh

Tato kapitola se zabývá definováním požadavků na program, jejich analýzou a návrhem řešení.

4.1 Definování cílů

Cílem je vytvořit program na syntézu obrazu pomocí metody sledování paprsků, který ověří vlastnosti některých optimalizačních metod. Jedná se o optimalizační metody dělení prostoru, konkrétně o dělení prostoru pomocí KD stromu.

Požadavky na systém jsou:

- realistická syntéza obrazu
- přenositelnost programu
- optimalizace metodou dělení prostoru pomocí KD stromu
- možnost měnit nastavení programu

Program bude vytvořen především pro určení vlastností konkrétních optimalizačních metod, proto se nepředpokládá jeho použití jako obecného syntetizátoru obrazu. Tím je dáno i jeho ovládání, které je jednoduché. Nepředpokládá se použití GUI interface nebo vstupních souborů pro nastavení programu. Všechny vlastnosti programu budou nastavitelné pomocí parametrů při překladu.

Program musí umožnit změnu nastavení:

- zobrazení
- použité scény
- kamery
- použití optimalizace metodou dělení prostoru pomocí KD stromu
- metody pro určení dělících rovin při výstavbě KD stromu
- použití optimalizované struktury KD uzlu

Pro návrh programu bude použit objektově orientovaný přístup, kvůli možnosti následujícího snadného rozšíření programu o další vlastnosti.

4.2 Struktura programu

Program bude pracovat následujícím zpusobem:

1. načtení a vytvoření modelů a materiálů
2. vytvoření scény z modelů a materiálů
3. vytvoření a umístění světel
4. nastavení kamery a vykreslování
5. vystavění KD stromu
6. syntéza scény pomocí sledování paprsků
7. vykreslení scény na obrazovku

V programu bude využito několik existujících knihoven. Jedná se o grafickou knihovnu OpenGL a její nástavbu GLUT. Tyto knihovny jsou použity pro vykreslení výsledného obrazu a správu systémových služeb, a to z důvodu jejich přenositelnosti mezi systémy. Dále pak knihovny pro načítání 3D modelů ve formátu OBJ a knihovna pro práci s texturamy ve formátu TGA. Tyto formáty byly vybrány pro svoji jednoduchost.

Jako základní prvky, ze kterých se skládá scéna budou použity trojuhelníky a koule. Trojuhelníky proto, že jsou základními prvky všech grafických celků a také proto, že je využívá použitý formát pro modely OBJ. Použití koulí neni vynuceno, ale budou použity pro svou jednoduchost. Použití dalších geometrických primitiv je možné v případném rozšíření programu, díky objektově orientovanému návrhu. Bylo uvažováno i o použití tvaru reprezentující neomezenou plochu. Zde však nastal problém s jejím zapracováním do prostoru KD stromu, proto nebyla imlementována.

Pro osvětlení budou použita pouze bodová světla, která jsou v běžných scénách dostačují. Pro vytvoření měkkých stínu může být použít trik s použitím více bodových světel vedle sebe. Další možnosti rozšíření osvětlení jsou však možné, díky objektově orientovanému návrhu.

Program bude vyžadovat přiřazení materiálu ke každému objektu. Materiál bude reprezentovat jako objekt, což umožní jeho jednoduché rozšiřování a změnu vlastností. V programu budou implementovány tyto dílčí vlastnosti materiálů:

- jednotná barva nebo použití barevné textury
- difuzní vlastnost materiálu
- spekulární vlastnost materiálu
- odrazivost materiálu
- průsvitnost materiálu a s tím související index lomu materialu

Jako textury mohou být použity:

- procedurální textury
- textury načtené ze souboru ve formátu TGA

Pro nanášení textur bude použito UV mapování se základním bilineárním filtrováním textur. UV mapování bylo použito proto, že je využívá i použitý formát modelů OBJ.

4.3 Použití KD stromu

Optimalizace pomocí dělení prostoru KD stromem by mělo přinést značné zvýšení výkonu programu. V této části budu využívat techniky popsané v kapitole 3.

4.3.1 Výstavba KD stromu

Pro výstavbu KD stromu bude použit rekurzivní algoritmus. Pro určení pozice dělící roviny bude na výběr z metod využívajících:

- prostorového mediánu
- objektového mediánu
- cenového modelu založeného na SAH

Výběr metody bude možný přes parametry při překladu. Zároveň bude možné měnit vlastnosti KD stromu,jako je maximální hloubka stromu nebo minimální počet objektů v listu, při kterém se ukončí rekurzivní výstavba stromu.

Další nastavitelnou vlastností bude struktura KD uzlu. Na výběr jsou možné dvě varianty:

- základní neoptimalizovaný uzel, obsahující pozici dělící roviny, ukazatel na dva synovské uzly, identifikaci listu a ukazatel na seznam objektu.
- optimalizovaný uzel, který obsahuje pouze pozici dělící, identifikaci uzlu a jeden ukazatel, který se mění dle potřeby na ukazatel na seznam objektů nebo na ukazatel na levý synovský uzel, kde platí, že pozici pravého získáme aritmetickou operací s uzlem.

Při výstavbě stromu se často používá funkce pro určení hranic objektu (AABB obálky). Výpočet hranic objektů sice neni příliš náročný, ale je volán během výstavby mnohokrát, a to v součtu může ovlivnit dobu výstavby KD stromu viz [3]. Proto si rozsahy objektů předpočítáme a uložíme je do třídy popisující objekt. Tato vlastnost bude také nastavitelná.

Nastavitelné vlastnosti budou použity při testování a budou sloužit k porovnání jednotlivých metod a k určení jejich vlivu na výkon programu.

4.3.2 Průchod paprsku KD stromem

Pro průchod paprsku KD stromem bude použit iterativní algoritmus nazvaný TA(rec-B) specifikovaný v [4]. Tento algoritmus je použit, protože je velmi robustní a efektivní. Pro jeho implementaci bude potřeba vytvořit zásobník. Zásobník bude implementován jednoduše jako pole s indexem na vrchol zásobníku.

Kapitola 5

Implementace

V této kapitole bude popsána samotná implementace, která vychází z analýzy a návrhu provedeného v kapitole 4.

5.1 Implementační prostředí

Program byl implementován v programovacím jazyce C++. Tento jazyk byl vybrán z důvodu objektově orientovaného přístupu návrhu aplikace. Zároveň tento jazyk splňuje požadavky na výkonnost a přenositelnost a obsahuje sadu standardních šablon, které jsou velice efektivni a velmi urychlují vývoj aplikace. Důvodem byla také možnost použití některých externích knihoven, a to především knihovny OpenGL s nástavbou GLUT a také knihoven pro načítání textur[7] a 3D modelů[2].

Vývoj byl veden převážně na linuxové platformě s dodržením zásad pro přenositelnost programu, jako vývojový prostředek byl použit stadartní linuxový textový editor GEDIT. Použití sofistikovanějšího prostředí nebylo při vývoji potřeba. Pro překlad byl použit GNU překladač GCC.

Samotný vývoj spočíval ve vytvoření první velmi jednoduché verze programu. Do této verze byly postupně přiimplementovávány další požadované vlastnosti, jako jsou optimalizace sledování paprsku pomocí KD stromu, použití rozličných druhů materiálu, práce s kamerou a scénou a také použití antialiasingu pro zlepšení výsledku syntézy obrazu. V průběhu přidávání vlastností byl program několikrát testován statistickým profilerem SYSPROF, pomocí kterého byly vybrány funkce, které jsou kritické pro časovou náročnost programu. Tyto funkce byly optimalizovány úpravami v jejich kódu.

Program byl testován na školním serveru merlin, kde pracoval bez problénů.

5.2 Popis tříd

Samotný program lze rozdělit do několika tříd, které lze dále rozřadit do několika skupin.

Třídy pro orientaci v prostoru
Pro orientaci v trojrozměrném prostoru se používají body a vektory.

Bod je definován pomocí tří neceločíselných hodnot, které udávají jeho pozici v každé ose trojrozměrného prostoru. Je implementován ve třídě `Point`. Tato třída umožňuje základní aritmetické operace s bodem jako jsou posun bodu, sčítání dvou bodu a další. Ve třídě `Point2D` je definován dvojrozměrný bod, který se používá pro UV mapování textur.

Vektor je definován pomocí tří neceločíselných hodnot, které udávají jeho směr od pozice počátku souřadnicových os. Je implementován ve třídě `Vector`, která dědí vlastnosti z třídy `Point` a rozšiřuje je o další, které by v případě bodů neměly smysl. Tyto vlastnosti jsou například násobení vektorů, skalární součet vektorů nebo normalizace vektoru.

Důležitý je i paprsek, který je popsán v třídě `Ray`. Paprsek je definován výchozím bodem a směrovým vektorem.

Třídy pro popis barev

Barva je implementována ve třídě `Color`. Dědí některé vlastnosti z třídy `Point`. Třída `Color` využívá reprezentaci barvy pomocí modelu RGB, s ohledem na kompatibilitu s prostředím OpenGL, které je použito pro vykreslení syntetizované scény. Proto je každá hodnota uložena pomocí neceločíselného typu s rozsahem nula až jedna.

Třídy pro popis materiálu a jejich vlastnosti

Popis materiálu je definován v třídě `Material`. Tato třída obsahuje údaje o optických vlastnostech materiálu jako je třeba průsvitnost, odrazivost a vyzařování světla. Je zde uložen také odkaz na použitou texturu.

Textury jsou popsané v třídách poděděných z třídy `Texture`. Jsou to třídy:

- `OneColor` definující jednobarevnou texturu
- `RandColor` definující texturu, jež je tvořena body náhodných barev.
- `Chess` je textura složená z dvou barev, která jsou zobrazeny jako šachovnice
- `TextureFile` je textura načtená ze souboru typu TGA. Pokud je v nastavení povoleno je na texturu použito bilineární filtrování

Třídy pro popis objektů

Tvar objektů je popsán v třídách odvozených od třídy `Shape`. Je to třída `Triangle` pro popis trojúhelníku a třída `Sphere` pro popis koule. Důležité metody těchto tříd jsou metoda pro výpočet průsečíku s paprskem, metoda pro určení rozsahu objektu(AABB obálky) a metoda vracející normálu v bodě průsečíku.

Obecný objekt je popsán třídou `Object`. Každému objektu je přiřazen tvar pomocí třídy `Shape` a materiál třídou `Material`.

Světelný zdroj je popsán ve třídě `Light`. Je definován bodem, ze kterého svítí a intenzitou. Jsou použita pouze bílá světla.

Třídy pro popis optimalizačních metod

KD strom je popsán třídou `KDnode`, která popisuje uzel tohoto stromu. Její nejdůležitější metody jsou metoda pro rozdělení prostoru uzlu do dvou synovských uzlů. Tato metoda se používá rekurzivně při vytváření KD stromu, je implementována tak, aby bylo možno měnit algoritmus výpočtu pozice dělící roviny. Dále pak metody pro určení prvního průsečíku scény definované KD stromem s paprskem a metoda pro určení stínících objektů, která se používá při výpočtu osvětlení.

Třídy pro popis scény

Scéna je popsána ve třídě `Scene`. Třída ukládá seznam objektů a světel ve scéně, kořenový uzel (root) KD stromu a pozici kamery a vykreslovací roviny. Mezi nejdůležitější metody patří metoda pro inicializaci scény, ve které se provádí mimo jiné i výstavba KD stromu a metoda pro syntézu obrazu.

Třídy pro sběr statistických údajů

Sběr statistických údajů o provádění programu je implementován v globální třídě `Stat`. Tato třída shromažďuje informace o počtu primárních, odražených, lomených a stínících

paprsků. Rovněž uchovává údaje o počtu výpočtu průsečíku s jednotlivými druhy geometrických tvarů a počet výpočtu osvětlovacího modelu. Sběr statistik je možno vypnout nastavením parametrů programu. Statistiky jsou vypsány po vykreslení.

5.3 Nastavení programu

Program je nastavován pomocí parametrů při překladu. Nastavení můžeme měnit přímo připsáním vhodného klíčového slova do parametrů překladače nebo lépe zakomentováváním jednotlivých vlastností v hlavičkovém souboru constant.h a následným překladem. V souboru constant.h jsou vysvětleny jednotlivé možnosti nastavení a je ukázán jejich příklad.

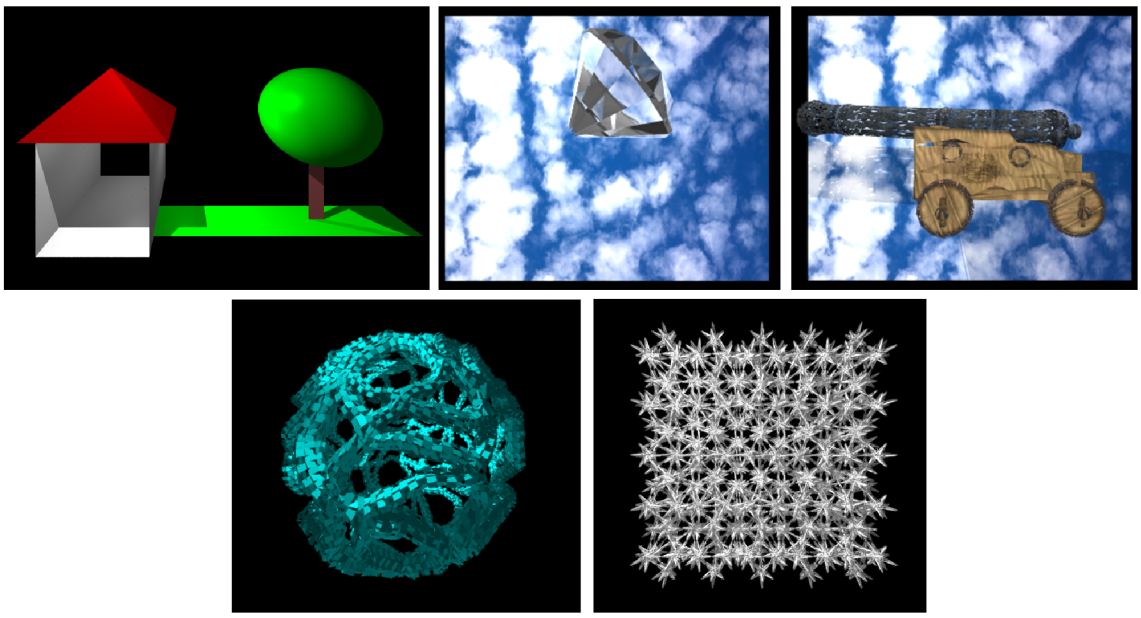
Kapitola 6

Testování

Cílem této kapitoly je představení testovacích scén a jejich použití pro testování vlastností určených metod optimalizace sledování paprsku, které byly představeny v kapitole 4. Testování bude probíhat podle toho v jakém pořadí byly jednotlivé metody implementovány.

6.1 Popis testovacích scén

Bylo naimplementováno celkem pět scén s různými vlastnostmi, jako je geometrické rozložení v prostoru a počet polygonů. Aby bylo možné scény ve výsledku srovnávat, obsahuje každá tři zdroje světla a skládá se převážně z trojúhelníků. Scény jsou seřazeny podle počtu trojúhelníků, které obsahují.



Obrázek 6.1: Ukázka použitých scén

- Scéna 1

  Je to nejjednoduší scéna. Obsahuje celkem 15 objektů, z toho 14 trojúhelníků a jednu kouli.

- Scéna 2

  Scéna obsahuje model diamantu na texturovaném pozadí. Model se snaží napodobit vlastnosti reálného diamantu, jako jsou lom a odraz světla. Model je převzat z [1]. Scéna obsahuje 112 trojúhelníků.

- Scéna 3

  Jedná se o model středověkého děla na texturovaném pozadí. Model je převzat z [1]. Scéna obsahuje 5984 trojúhelníků.

- Scéna 4

  Model koule, je převzat z [1]. Scéna obsahuje 25632 trojúhelníků.

- Scéna 5

  Model krajky, je převzat z [1]. Scéna obsahuje 209203 trojúhelníků.

6.2 Nastavení testů

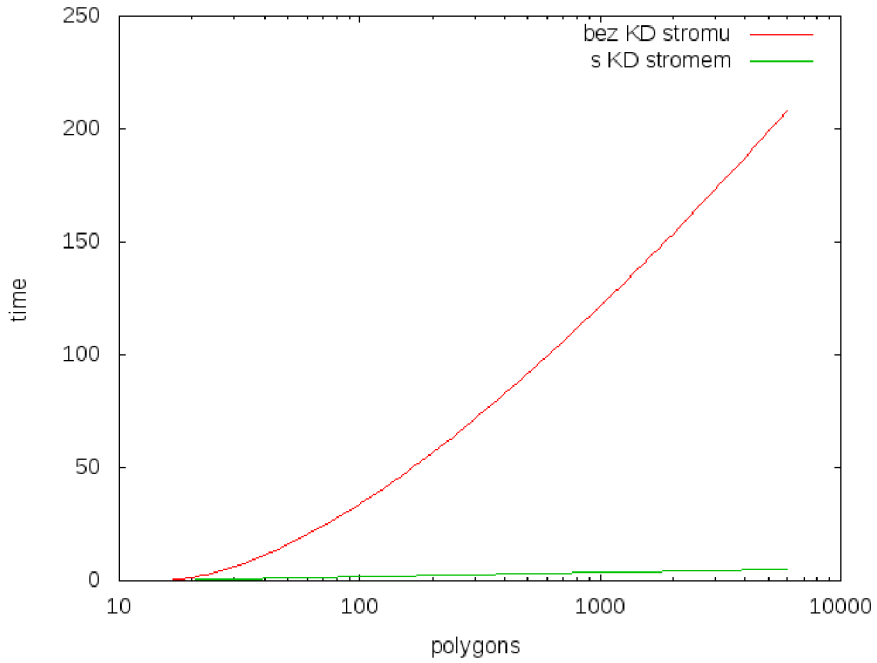
Testy budou prováděny na notebokovém počítači s základní konfigurací Intel Core 2 Duo T7300 2GHz, 2GB RAM 667 MHz na operačním systému Ubuntu 10.4. Překlad aplikace byl proveden nástrojem GCC s optimalizacemi při překladu nastavenými na -03. Nastavení programu bude následující:

- velikost zobrazovací plochy 640x640 obrazových bodů, bez antialiasingu
- perspektivní projekce
- maximální hloubka rekurze sledování paprsku bude 3
- na textury bude použito základní bilineární filtrování
- počet objektů, při kterém se již nebude dělit uzel u prostorového a objektového mediánu, bude 2
- nastavení konstant Ki a Kt pro použití cenového modelu, bylo experimentálně určeno na Ki=0,3 a Kt=1

6.3 Použití KD stromu

Budeme testovat, jaký vliv má použití KD stromu na celkový čas syntézy scény. Porovnáme časy bez použití a s použitím KD stromu. Pro KD strom byla použita metoda dělení prostoru pomocí objektového mediánu a byla vybrána hloubka stromu s nejlepším výsledkem. Testovány jsou scény 1 až 4. Scéna 5 neni testována, protože by její testování programem bez KD stromu trvalo velice dlouho.

		Scéna 1	Scéna 2	Scéna 3	Scéna 4
	trojúhelníků	15	112	5984	25632
bez KD stromu	čas [s]	0,57	3,9	208,24	825,49
	průsečíků	8,37	80,44	3577,27	13603,22
s KD stromem	čas [s]	0,36	2,03	5,87	3,9
	průsečíků [milionů]	2,57	32,5	78,12	44,93
bez / s	čas	1,58	1,92	35,47	211,66
	průsečíků	3,25	2,47	45,79	302,76



Obrázek 6.2: Vyhodnocení testu použití KD stromu

Z výsledku testu jasně vyplývá, že optimalizace použitím KD stromu výrazně zvyšuje rychlost syntézy scény pomocí sledování paprsku. Čím je scéna složitější, obsahuje více objektů, tím je i vyšší poměrné zrychlení.

Pokud porovnáme počet výpočtů průsečíku je zřejmé, že jejich poměrný pokles je větší než samotné zrychlení, což je způsobeno algoritmem průchodu paprsku přes KD strom. To potvrzuje i statistický profiler použitý na program syntetizující třetí scénu za pomocí KD stromu, kdy zjistíme, že algoritmus průchodu stromem spotřebovává 15% času.

6.4 Použití metod pro určení pozice dělící roviny

Porovnáme tři metody dělení prostoru použité při výstavbě stromu a jejich vliv na výkon syntézy obrazu. Pro každou metodu a scénu je použita vždy taková hloubka stromu, která podává nejlepší výsledky.

Vliv použité metody dělení prostoru na dobu syntézy obrazu:

	Scéna 1	Scéna 2	Scéna 3	Scéna 4
prostorový medián	0,38	1,16	8,82	7,31
objektový medián	0,36	2,03	5,87	3,9
cenový model SAH	0,36	1,96	9,23	7,86

Výsledek testu se neshoduje s očekávanými vlastnostmi. Cenový model SAH, který by měl podávat nejlepší výsledky, se ukazuje jako nejhorší metoda. To může být způsobeno špatným nastavením konstant nákladů na výpočet průsečíku(Ki) a nákladů na zanoření do hloubky(Kt) nebo také malým počtem testovacích scén, které nemusí poskytovat přesný statistický obraz.

6.5 Použití optimalizovaného KD uzlu

Porovnáme výkon programu a spotřebu paměti při použití neoptimalizovaného a optimalizovaného uzlu KD stromu. Pro naši implementaci platí, že neoptimalizovaný uzel má 56 bytů a optimalizovaný má 24 bytů. Test byl proveden na vzorku všech scén při různém nastavení vlastností programu, celkem 70 stejných testů pro každou variantu uzlu.

	neoptimalizovaný	optimalizovaný	rozdíl [%]
čas[s]	1529,22	1392,20	8,96

Vliv optimalizovaného uzlu KD stromu na celkový výkon neni sice velký, ale také není zanedbatelný. Důležitou vlastností je také ušetření paměti, protože optimalizovaný uzel je skoro 2,5 krát menší než původní neoptimalizovaný. To potvrzuje i při měření spotřebované paměti. Byla otestována scéna 4 s metodou dělení uzlu pomocí cenového modelu SAH a hloubkou stromu 25. Test zjistil, že při použití optimalizovaného uzlu bylo potřeba o více než 11MB paměti méně.

6.6 Výstavba stromu

Budeme porovnávat dobu výstavby a parametry stromu v závislosti na použité metodě pro určení pozice dělící roviny.

Scéna		1		2		3		4	
hloubka stromu		5	12	8	15	12	20	12	20
prostorový medián	čas[s]	0	0	0	0	0,02	0,04	0,08	0,15
	uzlů	49	517	39	131	547	2595	2081	11865
objektový medián	čas[s]	0	0	0	0,09	2,03	3,97	75,45	79,15
	uzlů	55	1255	511	52647	8191	1389213	8191	1870695
cenový model SAH	čas[s]	0	0	0,01	0,02	3,27	4,03	81,48	85,75
	uzlů	11	11	333	6633	2119	38369	3907	64181

Z výsledků testu je zřejmé, že výstavba KD stromu pomocí metody dělení prosoru prostorovým mediánem je z použitých metod nejrychlejší. To odpovídá i teoretickým předpokladů. Metody objektového mediánu a cenového modelu mají přibližně srovnatelné časy. Zároveň lze vypozorovat, že doba výstavby stromu u těchto dvou metod je závislá na počtu objektů ve scéně.

Pokud porovnáváme počty vytvořených uzlů, zjistíme, že metoda objektového mediánu vytváří při porovnání s ostatními mnohem více uzlů. Tato vlastnost může být nevýhodou při syntéze složitých scén (řádově statisíce objektů), protože může vzniknout tak rozsáhlý strom, který již nebude možno umístit do systémové paměti a to bude mít za následek rapidní snížení rychlosti programu.

Pokud použijeme statistický profiler zjistíme, že při vystavbě KD stromu je velice často používána funkce pro zjištění hranice objektu (AABB obálky). Proto byla provedena optimalizace programu a hranice objektů byly předpočítány před samotným průchodem paprsku. Otestujeme použití této optimalizace.

Scéna		3		4	
hloubka stromu		12	20	12	20
prostorový medián	původní čas[s]	0,02	0,04	0,08	0,15
	předvypočítané hranice[s]	0,02	0,04	0,08	0,15
objektový medián	původní čas[s]	2,03	3,97	75,45	79,15
	předvypočítané hranice[s]	0,57	2,25	9,36	11,57
cenový model SAH	původní čas[s]	3,27	4,03	81,48	85,75
	předvypočítané hranice[s]	0,96	1,14	10,39	11,56

Z výsledku vyplývá, že předvypočítání obálek má zásadní pozitivní vliv na dobu výstavby KD stromu u metod dělení prostoru pomocí objektového mediánu a cenového modelu SAH.

Kapitola 7

Závěr

Použití KD stromu pro urychlení sledování paprsků je velmi vhodné, protože dochází k rapidnímu snížení výpočtů průsečíků a tím i k zrychlení samotné syntézy obrazu. Tato vlastnost se nejlépe ukázala při testování složitých scén (řádově desetitisice a více objektů), kde by syntéza obrazu pomocí základního algoritmu sledování paprsku trvala řádově hodiny nebo i dny. Použitím KD stromu je možné takové scény syntetizovat v řádu sekund nebo minut. Teoreticky by měla být rychlost syntézy obrazu s použitím KD stromu logaritmicky závislá na počtu objektů ve scéně. Tento předpoklad přibližně potvrzují i výsledky této práce, ale bylo použito jen několik testovacích scén, proto nemusí být statistický obraz zcela přesný.

Bylo otestováno a naimplementováno několik metod pro určení dělíci roviny používajících se při výstavbě KD stromu. Tyto metody se výrazně podílejí na kvalitě výsledného KD stromu a tím i na rychlosti syntézy obrazu. Podle teoretického předpokladu by měly nejlepší výsledky poskytovat metody založené na cenovém modelu (v našem případě použitá metoda SAH). Tento předpoklad se však úplně nepotvrdil, což může být způsobeno nevhodným nastavením konstant tohoto modelu nebo také malým počtem testovacích scén. Velmi se naopak osvědčila metoda pro určení dělíci roviny pomocí prostorového mediánu. Použitím této jednoduché metody je dosaženo bezkonkurenčně nejrychlejší výstavby KD stromu a současně se ukázalo, že i rychlost sledování paprsku je srovnatelná s ostatními metodami.

Při testování se ukázalo, že doba výstavby stromu hraje nezanedbatelný podíl na celkové rychlosti programu. Tato vlastnost se projevuje především u složitých scén, kdy výstavba KD stromu trvá i několikanásobně déle než samotná syntéza obrazu. Bylo zjištěno, že na dlouhé době výstavby KD stromu se z velké části podílí funkce pro určení hranic objektů (takzvaných AABB obálek), proto byla použita optimalizace, která spočívala v předvypočítání těchto hranic objektů. Tato optimalizace znamenala výrazné zkrácení doby výstavby stromu. Použití další metody optimalizace spočívající v minimalizaci datových struktur pro popis KD stromu přineslo jen velmi malé zvýšení výkonu. Na druhou stranu ovšem došlo ke zmenšení nároků na paměť.

V případě pokračování projektu bych se zaměřil na urychlování výstavby KD stromů. Další možností vývoje by také mohla být akcelerace výpočtů pomocí SSE jednotek nebo pomocí grafické karty(GPGPU) s cílem syntézy obrazu v reálném čase.

Literatura

[1] Online databáze OBJ modelů[online]. `http://www.sharecg.com/`.

[2] Wavefront OBJ Specification[online]. `http://netghost.narod.ru/gff/vendspec/waveobj/obj_spec.ps`.

[3] Biker, J.: Raytracing: Theory Implementation Part 7, Kd-Trees and More Speed[online]. `http://www.devmaster.net/articles/raytracing_series/part7.php`.

[4] Havran, V.: *Heuristic Ray Shooting Algorithms, disertační práce.* FEL CVUT Praha, 2001.

[5] Havran, V.; Herzog, R.; Seidel, H.-P.: *On the Fast Construction of Spatial Hierarchies for Ray Tracing.* FEL CVUT Praha, MPI Saarbrucken, 2006.

[6] Krivda, M.: *Zobrazení kulečníku pomocí distribuovaného sledování paprsku, diplomová práce.* FIT VUT Brno, 2009.

[7] Molofee, J.: Výpis OpenGL rozšíření, ořezávací testy a textury z TGA obrázků[online]. `http://nehe.ceske-hry.cz/tut_24.php`.

[8] Wald, I.: *Realtime Ray Tracing and Interactive Global Illumination, disertační práce.* Computer Graphics Group Saarland University, Saarbrücken, Germany, 2004.

[9] Wald, I.; Havran, V.: *On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N).* FEL CVUT Praha, SCI Institute University of Utah, 2006.

[10] Whitted, T.: *An Improved Illumination Model for Shaded Display.* ACM Communication on Graphics, 1980, iSBN 26-342-349-6.

[11] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika.* Computer Press, 2004, iSBN 80-251-0454-0.

Příloha A

Obsah CD

- Elektronická verze technické zprávy bakalářské práce ve formátu PDF spolu se zdrojovými kódy pro Latex
- Zdrojové kódy programu pro syntézu obrazu pomocí KD stromu a soubory nutné k překladu
- Dávkové testovací skripty
- Tabulka výsledků testů
- Vytvořené obrazy testovacích scén