Závěrečná zpráva - Raytracer s akcelerační strukturou

Jakub Votrubec
ČVUT FIT, NI-PG1 2024/25

1 Úvod

Tento dokument popisuje závěrečný výstup, v podobě semestrální práce, z předmětu NI-PG1. Náplní semestrální práce bylo vytvoření vlastního raytracering renderu a implementace akcelerační datové struktury pro zrychlení traverzace scény. Cílem bylo prokázat schopnost optimalizovat výkon raytraceru pomocí vhodné datové struktury. Semestrální práce byla zpracována v jazyce C++, standard C++17, a využívá externí knihovny pouze pro načítání scén a konfigurace ve formátu `json`.

2 Struktura raytraceru

Implementoval jsem základní raytracer pro vykreslování scén složených z trojúhelníků. Raytracer má následující funkce, mezi kterými lze přepínat pomocí konfigurace:

- Stíny pomocí stínových paprsků.
- Podpora plošného světla skrze náhodné vzorkování trojúhelníků.
- Vykreslení scény s různými osvětlovacími modely (distance shading, čistě difuzní osvětlení, Phongův osvětlovací model, Blinn-Phongův osvětlovací model).
- Vykreslení scény s různými stánovacími modely (flat shading, smooth shading - Phongův stínovací model).
- Podpora odrazu a lomu světla.
- Backface culling.
- Vykreslení scény s více vzorky na pixel (fuzzysampling), viz *Použité metody nad rámec cvičení*.

Raytracer je navržen tak, aby byl modulární a snadno rozšiřitelný. Hlavní třída `Raytracer` obsahuje metody pro vykreslování a ukládání výsledného obrazu. Scéna je reprezentována jako seznam trojúhelníků, které jsou načteny z `.obj` souboru.

Raytracer načítá scénu pomocí knihovny `tinyobjloader` a používá knihovnu `nlohmann/json` pro načítání konfigurace. Scéna se vykreslí do formátu `.ppm`.

2.1 Použité metody nad rámec cvičení

Při renderování scény jsem zjistil problém s velkým množstvím šumu ve výsledném obrazu. Rozhodl jsem se implementovat několikanásobné vzorkování. Metodu jsem nazval *fuzzysampling*.

Fuzzysampling spočívá v tom, že pro každý pixel se vygeneruje několik paprsků, ale každý s lehce pozměněným směrem paprsku. Jejich výsledky se poté průměrují. Tím se dosáhne hladšího obrazu a sníží se šum. Zmeny směru paprsk jsou generovány náhodně v rámci pixelu, což umožňuje zachovat detaily a zároveň eliminovat šum.

3 Akcelerační datová struktura

Implemetoval jsem dvě varianty akcelerační datové struktury octree:

- základní varianta octree,
- parametrická varianta podle článku *An Efficient Parametric Algorithm for Octree Traversal* od Revelles et al..

3.1 Stavba a traverzace

Octree je struktura, která dělí prostor na menší buňky, což umožňuje rychlejší vyhledávání kolizí mezi paprsky a trojúhelníky. Každý uzel octree obsahuje bounding box, který definuje prostor, který uzel pokrývá, a seznam trojúhelníků, které se v tomto prostoru nacházejí.

3.1.1 Základní octree

Základní octree je implementován jako rekurzivní struktura, která dělí prostor rekurzivně na osm dalších uzlů (oktetů), dokud není dosaženo maximální hloubky nebo počtu trojúhelníků v uzlu.

Traverzace probíhá tak, že pro každý paprsek se nejprve zjistí, zda protíná bounding box aktuálního uzlu. Pokud ano, poračuje se dál na jeho oktety, pokud ne, traverzace končí. Je-li oktet již list stromu (neobsahuje žádné vlastní oktety), traverzace končí a jeho trojúhelníky jsou přidány k výsledku. Traverzace pokračuje rekurzivně, dokud se neprohledá celý strom. Prochází se vždy všechny oktety uzlu.

3.1.2 Parametrický octree

Parametrický octree je vylepšená verze, která využívá parametrickou traverzaci. Tato metoda umožňuje efektivnější prohledávání prostoru tím, že využívá parametrické rovnice pro určení průsečíků paprsku s hranicemi uzlů octree (tzv. "slab"metoda).

Nejprve se vypočítájí průsečíky paprsku s hranicemi uzlu, z nich se určtí "čas", kdy paprsek vstoupí a opustí daný uzel na jednotlivých osách. Pokud paprsek opustil uzel na nějaké ose dříve, nežli vstoupil na všech osách - minul bounding box uzlu a tato větev traverzace se ozavírá.

Pokud paprsek vstoupil, tak na základě průsečíku vstupu se vypočítá první oktet, do kterého paprsek vstoupí. Ten se následně přidá do fronty pro další zpracování. Následně se postupně se projdou a přidají všechny oktety, do kterých paprsek vstoupí. Procházení všech dalších oktetů probíhá na základě konečného automatu, který určuje, do kterého oktetu paprsek vstoupí na základě jeho směru a aktuálního oktetu.

Optimalizace oproti základnímu octree spočívá v tom, že se neprocházejí všechny oktety, ale pouze ty, do kterých paprsek skutečně vstoupí. Zároveň výpočet velikostí oktetů je založen jen na půlení velikosti rodičovského uzlu, což umožňuje efektivní a výpočetně nenáročné dělení prostoru.

Parametrickou traverzaci se mi bohužel nepodařilo implementovat, vice viz *Problémy při implementaci*.

3.2 Problémy při implementaci

Při implementaci stavby octree jsem narazil na problémy s přesností výpočtů velikosti oktetů. Kvůli chybě u datových typů s plovoucí desteinou čárkou se ne vždy přiřadily všechny trojúhelníky rodičovského uzlu do jeho oktetů, což vedlo k chybám při traverzaci. Tento problém jsem vyřešil nafouknutím bounding boxů o toleranci *epsilon*.

Bohužel se mi nepodařilo správně implementovat a zprovoznit parametrický octree. Narazil na problémy při traverzaci. Traverzace nenalezne všechny trojúhelníky a scéna je poté vyrenderována s dírami, jak švýcarský sýr. Zkoušel jsem, jestli je problém opět v přesnosti výpočtů s čísly s plovoucí desetinnou čárkou, ale nepodařilo se mi ho vyřešit. Proto jsem se rozhodl parametrický octree nevyužít a v měření výkonu použít pouze základní octree.

4 Měření výkonu

4.1 Nastavení měření

Pro měření výkonu byla použita scéna `CornellBox-Sphere.obj` s následující konfigurací:

Tabulka 1: Použité nastavení rendereru

Parametr	Hodnota
Výsledné rozlišení	800×800
Max. zanoření paprsku	10
Vzorků plošného světla na trojúhelník	50
Osvětlovací model	Blinn-Phong
Stínovací model	Smooth
Backface culling	Ano
Fuzzysampling	Ne

Tabulka 2: Použité nastavení struktury octree

Parametr	Hodnota
Max. počet trojúhelníků na uzel	16
Max. hloubka uzlů	10

4.2 Výsledky měření

Porovnání výkonu bez akcelerační struktury a s akcelerační strukturou octree:

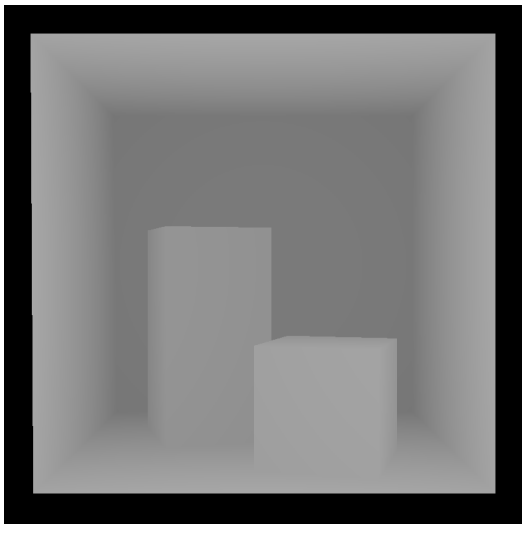
Tabulka 3: Porovnání výkonu raytraceru

Metrika	Bez ADS	Octree
Čas renderování [s]	3h 24m 12.4s	13m 53.7s
Počet kolizí paprsek-trojúhelník	2.46×10^{11}	2.81×10^{9}
Průměrný počet kolizí na paprsek	383 665	4 395
Čas výpočtu kolizí [s]	8 032.8	121.5
Průměrná doba výpočtu kolizí na paprsek [ms]	12.55	0.19

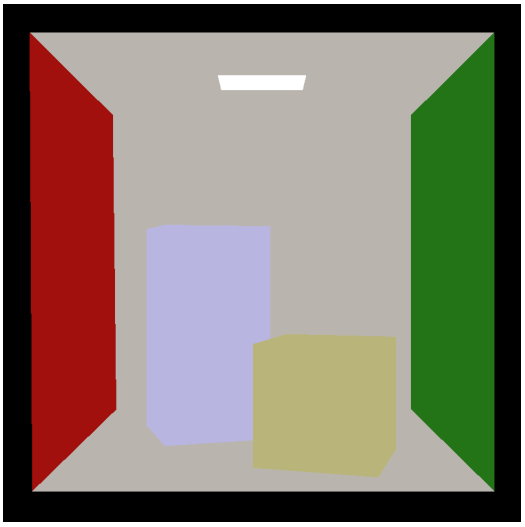
Tabulka 4: Statistiky struktury octree

Statistika	Hodnota
Počet uzlů	1 584
Počet listů	1 300
Průměrná hloubka listů	5.22
Max. trojúhelníků v listu	34
Průměr trojúhelníků v listu	7.03
Čas stavby octree [s]	0.04
Počet prohledaných uzlů	1 138 825 603
Čas strávený traverzací	4m 43.943s
Průměrný počet trojúhelníků vrácených z traverzace	33.7

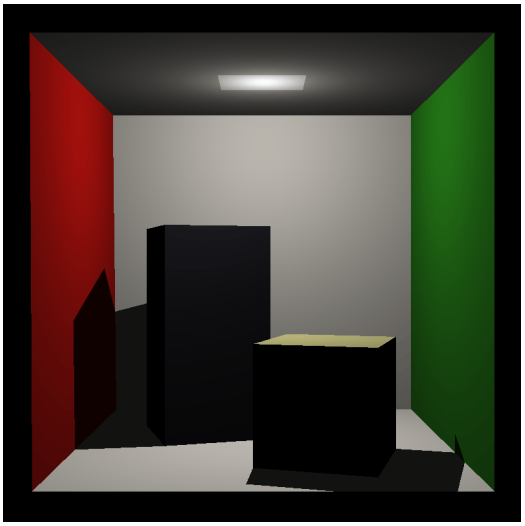
5 Ukázky výstupů



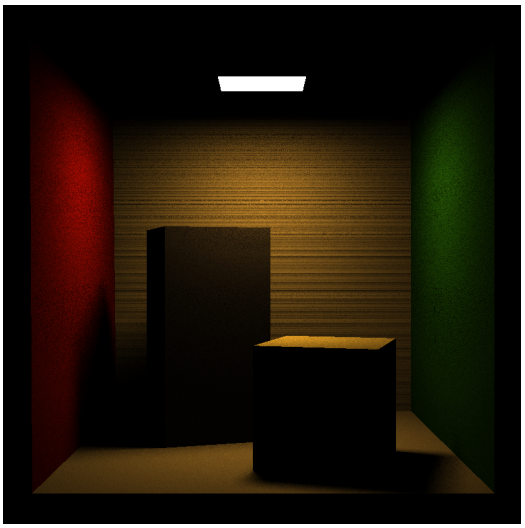
Obrázek 1: Ukázka vykreslení scény s distance shadingem.



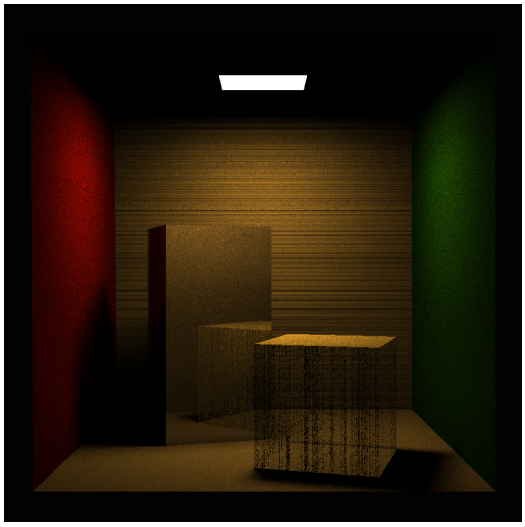
Obrázek 2: Ukázka vykreslení scény s difúzním osvětlením.



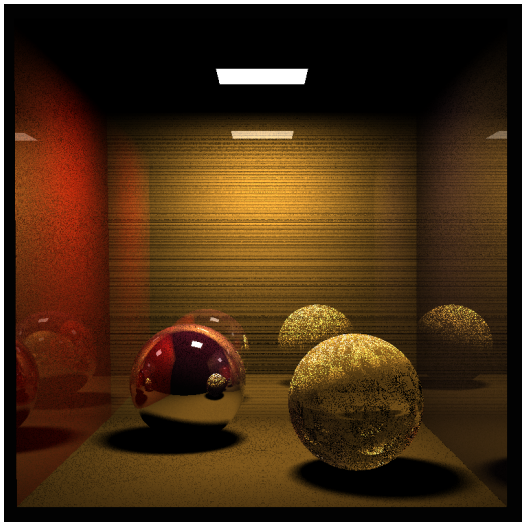
Obrázek 3: Ukázka vykreslení scény s Blinn-Phong osvětlením a stíny.



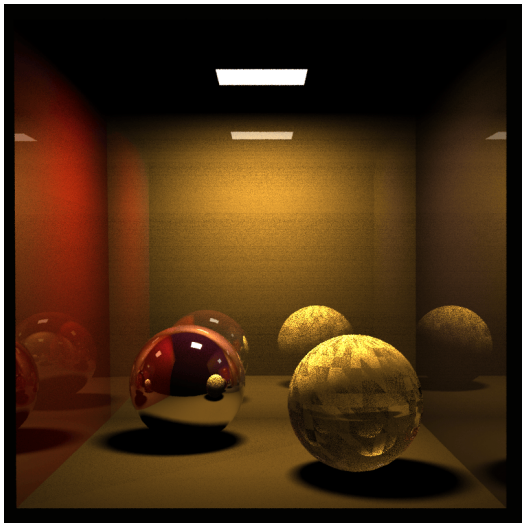
Obrázek 4: Ukázka vykreslení scény s plošným světlem.



Obrázek 5: Ukázka vykreslení scény s odrazem a lomem světla.



Obrázek 6: Ukázka vykreslení scény s odrazem a lomem světla na zrcadlových stěnách.



Obrázek 7: Ukázka vykreslení scény s fuzzysamplingem (10 vzorků na pixel).

6 Implementace

Veškerá implementace je dostupná na fakultním GitLabu a mém GitHubu.

Více informací o kompilaci, konfiguraci a spuštění naleznete v souboru README samotného repozitáře.

7 Závěr

Tato semestrální práce prokázala schopnost optimalizovat výkon raytraceru pomocí akcelerační datové struktury octree. Implementace základního octree vedla k významnému zrychlení traverzace scény a snížení počtu kolizí mezi paprsky a trojúhelníky. Parametrickou variantu octree se mi bohužel nepodařilo implementovat, což je škoda, protože by mohla přinést další zlepšení výkonu.

Další rozšíření raytraceru by mohlo zahrnovat implementaci dalších akceleračních struktur, jako jsou BVH (Bounding Volume Hierarchy) nebo kd-stromy, které by mohly dále zlepšit výkon při renderování složitějších scén.

Rád bych také zkusil implementovat paralelní zpracování raytraceru, což by mohlo výrazně zrychlit renderování na moderních vícejádrových procesorech nebo GPU.

8 Použitá literatura

- Přednášky a materiály z předmětu NI-PG1.
- *An Efficient Parametric Algorithm for Octree Traversal* - Revelles et al.
- `tinyobjloader` - Knihovna pro načítání `.obj` souborů.
- `nlohmann/json` - Knihovna pro práci s JSON v C++.
- Dokumentace k C++ standardu a knihovnám.