Postupci ostvarivanja sjene

Orlandini, Roberto

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski studij računarstva

Završni rad

Postupci ostvarivanja sjene

Rijeka, rujan 2022.

Roberto Orlandini
0069082968

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski studij računarstva

Završni rad

Postupci ostvarivanja sjene

Mentor: izv. prof. dr. sc. Jerko Škifić

Rijeka, rujan 2022.

Roberto Orlandini
0069082968

Umjesto ove stranice umetnuti zadatak za završni ili diplomski rad

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2022.

_____
Roberto Orlandini

Zahvala

Zahvaljujem izv. prof. dr. sc. Jerko Škifiću na podršci tijekom pisanja ovoga rada.
Zahvaljujem kolegama na podršci tijekom studiranja.

Sadržaj

Popis slika **viii**

Popis tablica **ix**

1 Uvod **1**

2 Analiza problem **2**

2.1 Analiza tehnika 3

2.1.1 Mape sjena 3

2.1.2 Volumen sjena 3

2.1.3 Praćenje zraka 4

2.1.4 Izvori svjetla 4

3 Rješavanje zadatka **6**

3.1 Mape sjena 6

3.1.1 Izrada mape sjena 6

3.1.2 Crtanje pomoću mape sjena 7

3.1.3 Varijacije ovisno o vrsti svjetla 7

3.1.4 Nedostaci 8

3.2 Volumen sjena 9

3.2.1 Dubina scene i stencil operacije 10

3.2.2 Izrada volumena sjena 10
3.2.3 Crtanje scene 12
3.3 Praćenje zraka 13
3.3.1 Nedostaci 14

4 Implementacija rješenja **15**
4.1 Knjižice 15
4.1.1 GLFW i Glad 15
4.2 Renderer klase 15
4.2.1 Bazna klasa Renderer 16
4.2.2 Mape sjena 16
4.2.3 Volumne sjene 20
4.2.4 Praćenje zraka 23
4.3 Analiza rješenja 30
4.3.1 Brzina izvođenja 31

5 Zaključak **32**

Bibliografija **33**

Popis slika

2.1 Ilustracija različitih izvora svjetla. 5

3.1 Akne sjena, greska tehnike mape sjena. 8
3.2 Tehnika mapiranje sjena. 9
3.3 Određivanje siluete na objektu. 11
3.4 Tehnika volumnih sjena. 13
3.5 Tehnika pracenje zraka. 14

Popis tablica

4.1 opis tabele 31

Poglavlje 1

Uvod

U prirodi sjene su područja gdje je svjetlost potpuno ili djelomično blokirana nekim objektom. Sjena je trodimenzionalni volumen iza tog objekta gdje nema svjetlosti i mi vidimo samo dvodimenzionalnu siluetu projektiranu na prostor ili objekte u sjeni. Pomocu samog oblika sjene mozemo dobiti informacije kao sto su relativan smjer i udaljenost izvora svjetla, ali i pomaže nam odrediti relaciju između objekta u prostoru.

U računalnoj grafici postupci ostvarivanja sjena podrazumijevaju tehnike koje se koriste za ostvarivanje sjena u sceni, osim sto doprinose većem djelu realizma moze nam dati dodatne informacije, koje su jako korisne u kontekstu interaktivnih medija kao što su video igre, kao što su dubina i međusobni odnos objekta u sceni.

Poglavlje 2

Analiza problem

Svrha ovih tehnika jest da u sceni možemo odrediti koji dijelovi te scene su u sjeni i koji nisu. Tu informaciju ćemo poslije koristiti kada osvjetljavamo scenu. Proučit ćemo vise tehnika koje ovise o metodi ostvarivanja slike. Za metodu rasterizacije ćemo koristiti tehniku mapiranja sjena pomoću tekstura dubina (engl. Depth texture) i tehniku volumnih sjena pomoću šablona tekstura (engl. Stencil texture), a za metodu praćenja zrake (engl. Ray Tracing) umjesto "zraka sjena", zrake će bit više sklone ići prema svjetlu i tako će nastati prostor sa manje ili bez svjetla.

2.1 Analiza tehnika

2.1.1 Mape sjena

Mapiranje sjena je jedna od najpopularnijih tehnika u području interaktivne 3d grafike zbog jednostavnosti implementacije, efikasnosti i mnogo varijacija na tehniku koji daju razne prednosti kao što su kvaliteta sjena, efikasnost tehnike za udaljenije sjene i mogućnosti mekih (engl. Soft shadows) sjena.
U ovom radu ćemo obraditi jednostavne mape sjena (engl. Simple Shadow Mapping).

Baza tehnike je spremnik dubine (engl. Depth Buffer). Pomoću tog spremnika, odredimo koji djeljivi scene su osvijetljeni i koji su dijelovi u sjeni.

Tehniku možemo podijeliti na dva koraka:

I) izrada mape sjena u spremnik dubine

II) normalno iscrtavanje scene pomoću spremnika dubine

U prvom koraku za stvaranje mape scenu ćemo iscrtati u spremnik dubine iz perspektive svjetla. Stime dobivamo informaciju o najbližim točkama u sceni svijetlu tj vidimo koje točke su osvijetljene i sve točke koje su vise udaljene su u sjeni. Onda se spremnik dubine se pretvori u teksturu koja će se u sljedećem koraku koristiti za određivanje sjena. U drugom koraku scenu normalno ocrtavamo iz perspektive kamere ali u koraku određivanja boje točke koristimo informacije iz teksture mape sjena da odredimo da li je točka u sjeni ili ne.

2.1.2 Volumen sjena

Za razliku od tehnike mapiranje sjena, volumne sjene se znatno manje koristi i to zahvaljujući mnogo nedostatka gdje je glavni faktor je da tehnika zahtjeva vise resursa za izvršavanje. Kompleksnost izvršavanja brže raste sa kompleksnošću scene nego tehnika mapiranja. Tehnika se izvršava tako da pomoću položaja izvora svjetla i položaja svih objekta u sceni, sjenu *bacimo* iza objekta tako da izradimo volumen sjene pomoću siluete objekta koje je osvijetljen. To ponovimo za svaki objekt u sceni. Nakon toga scenu ocrtavamo tako da ako je neka točka unutar volumena onda je u

sjeni inače iscrtamo točku kao da je osvijetljena.

2.1.3 Praćenje zraka

Kao prije spomenuto tehnika ostvarivanja sjena pomoću praćenja zraka obuhvaća drugu metodu iscrtavanje scene, isto imenu metodu praćenje zraka. Za razliku od metode rasterizacije kod praćenja zrake mi simuliramo putanje svjetla slično kako bi bilo u stvarnom svijetu. To zahtijeva znatno vise resursa za izvršavanje nego prijašnje metoda, ali zato dobivamo realističnu sliku. Tehnika radi tako da za svaki piksel ekrana pratimo zraku koja izlazi iz kamera i kad zraka pogodi neki objekt u sceni, ovisno o materijalu objekta, zraka se odbije i nastavi do sljedećeg objekta ili izlaska iz scene. Za stvaranje sjene ima više mogućnosti. Jedna mogućnost jest da uz obične zrake koristimo još i zrake *sjena* tako da na pogotku zraku usmjerimo prema svijetlu i ako pogodi neki objekt prije izvora svijeta onda je ta točka u sjeni. Druga mogućnost, i tehnika koju ćemo koristiti u ovom radu, je da pri normalnom praćenju zraka svjetlosti, zraku na pogotku samo usmjerimo prema izvoru svjetla uz dodatno raspršivanje pomoću funkcije gustoće vjerojatnosti (engl. Probability density function (PDF)) i nastavimo dalje pratiti zraku. Time su zrake sklone ići prema izvoru i prirodno stvoriti sjene. Ova tehnika ima prednost lakse implementacije jer nije potrebno stvarati nove zrake za određivanje sjena.

2.1.4 Izvori svjetla

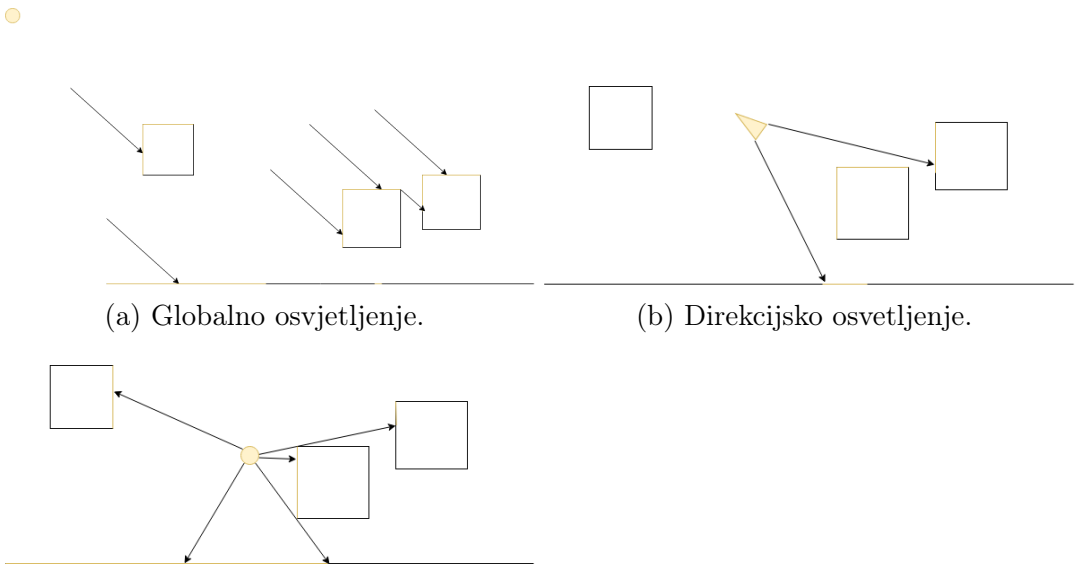
U 3D grafici možemo razlikovati vise vrsta izvora svjetlosti koji variraju po funkciji i *obliku* svjetla koji stvaraju. Neki od izvora svjetla su:

I) Globalno: je svjetlo koje možemo opisati kao sunce. Definira se samo se smjerom svjetla a pozicija je beskonačno udaljena. Zbog toga zrake koje dolaze od tog izvora su paralelne.

II) Pozicijsko: ili točkasto definira se sa pozicijom u sceni i prigušenjem svjetla, a zrake izlaze u svim smjerovima.

III) Direkcijsko: radi ako reflektor (engl. Spotlight) tako da stvara svjetlo samo u određenom smjeru. Definira se sa pozicijom u sceni i smjerom zraka svjetla.

Važno je razlikovati različite vrste svjetla jer neke tehnike ostvarivanje sjena su pogodnije za određene vrste izvora. U ovom radu samo ćemo direkcijsko i pozicijsko svjetlo koristiti. Tehnika mapiranje svjetla zbog načina stavanja map, tako da scenu iscrtamo iz perspektive svjetla, najbolje funkcionira kao direkcijsko, a da bi radio kao pozicijsko trebamo napraviti mapu sjena za svaki ostali smjer u sceni. A ostale tehnike su pogodne i sa direkcijskim i sa pozicijskim izvorom svjetlosti. Različiti izvori svjetla su ilustrirani na slici 2.1



(a) Globalno osvjetljenje.

(b) Direkcijsko osvetljenje.

(c) Pozicijsko osvijetljene.

Slika 2.1 Ilustracija različitih izvora svjetla.

Poglavlje 3

Rješavanje zadatka

3.1 Mape sjena

Kao prije spomenuto baza tehnike mape sjena je spremnik dubine stog prvo je potrebno izraditi spremnik. Ovisno o varijaciji metode moguće je da se koristi tekstura umjesto spremnika dubine jer je potrebno spremiti više informacija nego što stane u spremnik dubine. U jednostavnim mapama sjena koristi se jedino spremnik dubina jer je samo potrebno spremiti dubinu. Da bi koristio spremnik prvo moramo napraviti FBO (Framebuffer object). FBO nam služi za rukovanje spremnikom i za korištenje kao cilj iscrtavanja.

3.1.1 Izrada mape sjena

Prvi je korak da stvorimo novi FBO objekt koji sadrži spremnik dubine. Nakon što smo stvorili FBO moramo postaviti prozor za prikaz (engl. Viewport) da bude iste veličine kao što je spremnik dubine, ta veličina može se razlikovati od krajne veličine prozora na koji će se sve iscrtati. Kad smo postavili FBO i veličinu prozora onda ispraznimo spremnik i pripremimo scenu i shader-e. Prije nego što iscrtamo scenu potrebno je shader aktivirati i proslijediti podatke za transformaciju scene kao što su `projection`, `model` i `view` matrice i pozicija svjetla. Scenu sada crtamo iz perspektive svjetla u spremnik dubine tako da spremimo samo najbliže točke.

3.1.2 Crtanje pomoću mape sjena

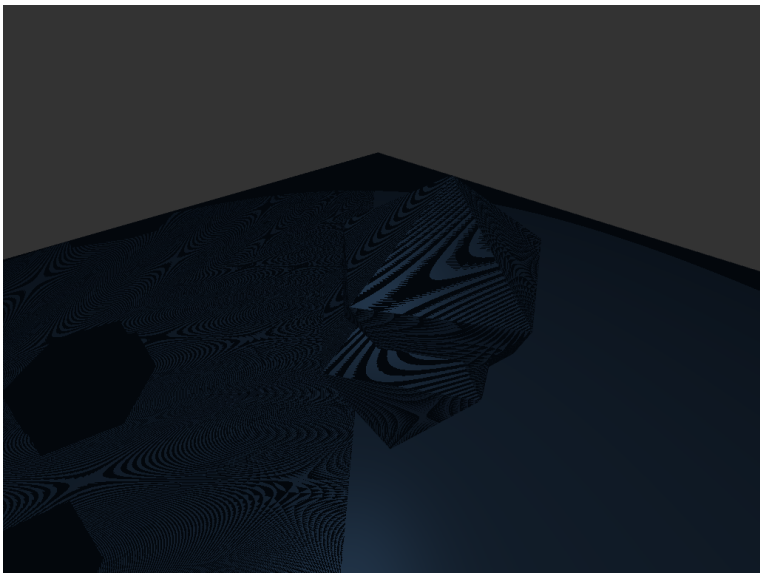
Drugi korak obuhvaća crtanje scene pomoću izrađene mape sjena. Prvo moramo deaktivirati FBO i postaviti prozor za prikaz na prvobitno stanje. Nakon toga aktiviramo shader, proslijedimo podatke i spojimo spremnik dubine na shader. Sada se crtanje vrsi iz perspektive oka. Tijekom crtanja da bi odredili da li je tocka u sjeni ili ne, prvo moramo u vertex dijelu shadera transformirati točke u svjetski prostor s matricom `model` i to proslijediti u fragment dio shadera prije nego što točke dalje transformiramo sa `view` i `projection` matricom u perspektivu oka. Sada u fragment dijelu shadera, određujemo da li je točka u sjeni tako da točku prebacimo iz perspektive oka u perspektivu svjetla na način da izračunamo vektor od točke do izvora i pomoću tog vektora možemo dobit uzorak iz mape sjena. Onda pomoću uzorka mape sjena možemo provjeriti da li je točka koju želimo nacrtati, iz perspektive svjetla, dublja od uzorka. Ako je točka dublja znaci da je u sjeni i tako ju i nacrtamo, ako je točka iste dubine ili bliža znaci da je u svijetlu.

3.1.3 Varijacije ovisno o vrsti svjetla

Ovisno o vrsti izvora svjetla implementacija se tehnike se mijenja. Najveća je razlika tipu spremnika dubine. U najjednostavnijim slučaju kad imamo direkcijsko svijetlo, spremnik može biti jednostavna dubinska 2D tekstura. Razlog tome je način kako ispunjavamo taj spremnik. Spremnik se puni tako da scenu iscrtavamo kao da smo postavili kameru na poziciju svjetla i usmjerili je prema zrakama svjetla. Ali problem nastane kad želimo koristiti pozicijski izvor svjetla, jer sa jednom teksturom možemo spremiti samo jedan smjer zraka svjetla. Da bi dobili mapu sjena za pet ostala smjera trebali bi napraviti jos pet tekstura i za svaku teksturu zasebno iscrtavati scenu u određeno smjeru. Umjesto korištenja posebnih šest tekstura pogodno je koristiti tip teksture mapa kocke jer obuhvaća sve smjerove potrebne za pozicijsko svjetlo. Mapa kocke isto olakšava dobivanje uzorka dubine gdje možemo dobiti pomoću vektora između točke i izvora svjetla umjesto da moramo prebacivati točke iz perspektive oka u perspektivu svjetla pomoću matrica transformacija i projekcija. U ovom radu za implementaciju mape sjena koristiti će se mape kocka za spremnik zbog prethodno spomenutih pogodnosti.

3.1.4 Nedostaci

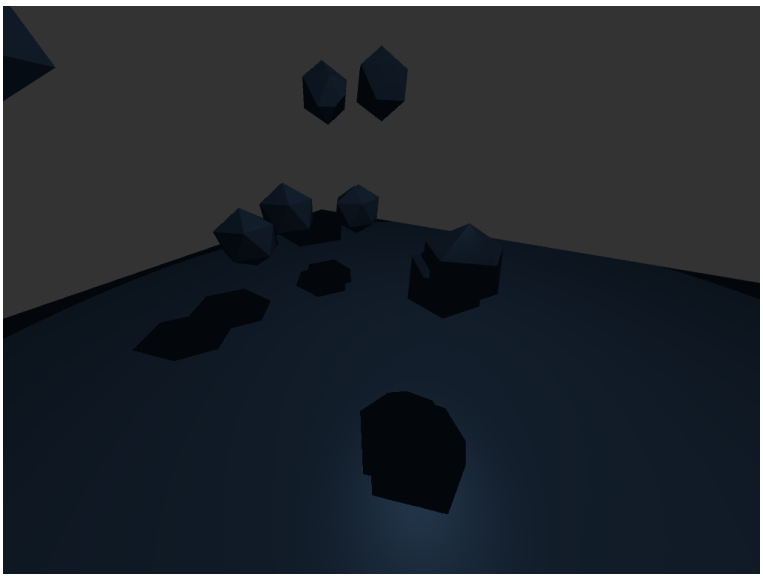
Jednostavne mape sjena sklone su grafički greškama zbog neispravnog postavljana tekstura i shadera. Greške u metodi mape sjena najčešće nastaju zbog neispravnog odabira rezolucije dubinske teksture. Ovisno o rezoluciji, poziciji i vrsti izvora svjetla grafičke greške mogu nastati prilikom iscrtavanja scene. Jedna od čestih vidljivih greška jest `akne` sjena. Greška akne sjena se vidi na slici 3.1



Slika 3.1 Akne sjena, greska tehnike mape sjena.

Zbog limitiranih resursa dostupnih aplikaciji, rezolucija teksture dubina je vrlo ograničena. Zbog toga prilikom crtanja scene jedna tocka u sceni može obuhvatiti vrijednost nekoliko piksela. U drugom koraku tijekom crtanja scene kad uzmemo uzorak dubine ta vrijednost može predstavljati više piksela na prozoru. Greška znatno ovisi o kutu pod kojim zrake svjetlosti padaju na objekte. Kad je dovoljno velik kut svjetlosti greška postaje vidljiva. Uzorak nastaje zbog netočnog određivanja da li je tocka u sjeni ili ne kada zbog male rezolucije teksture dobimo krivi uzorak dubine. Tako nastaju vidljive pruge gdje se formiraju grupe tocka koji su u sjeni zbog lošeg uzorka i obrnuto.
Akne se rješavaju koristeći varijable `bias` koja se oduzme od dubine točke u tijekom

crtanja scene tako da dobijemo. Pomoću te varijable tijekom određivanja točke `bias` oduzmemo od uzorka dubine koji izracunali kako bi dobili blizi vrijednost dubine. Vrijednost varijable treba pažljivo odabrati, uzimajuci u obzir rezoluciju teksture, kut svjetla i tipa izvora svjetla. U slučaju da je bias pre velik nastat ce novi problem zvan `Peter Panning` gdje sjena se pocne odvajati od objekta koji stvara sjenu. Zbog male rezolucije teksture mape sjena, sjene mogu dobiti kockasti izgled. Problem se može riješiti povećanjem rezolucije, ali kao prije spomenuto dostupni resursi mogu biti limitirani pa povecanje je nemoguce. Može isto riješiti tako da dobivene *kockaste* sjene popravimo pomoću filtrirajućih tehnika kao što su postotno bliže filtriranje (engl. percentage-closer filtering (PCF)), eksponencijalno filtriranje i slično.



Slika 3.2 Tehnika mapiranje sjena.

3.2 Volumen sjena

Za tehniku volumen sjena koriste se spremnik dubine i spremnik šablone. Za razliku od tehnike mape sjena, za tehniku volumen sjena nije potrebno izraditi FBO objekt, već možemo koristiti spremnike koji su već zadani u aplikaciji. Ideja iza metoda volumnih sjena je proširiti siluetu objekta koji nastaje kada svjetlost padne na

njega u volumen i zatim prikazati taj volumen u spremniku šablona pomoću nekoliko šablonskih operacija.

3.2.1 Dubina scene i stencil operacije

Slično kao u metodi mapa sjena, prvo scenu nacrtamo u spremnik dubine, ali iz perspektive oka. Zatim onemogucimo dodatno pisanje u spremnik dubina kako u sljedećim koracima dubina bi ostala netaknuta. Nakon što smo nacrtali scenu aktivira se `stencil` tekstura i pomoću funkcije `glStencilFunc()` i funkcije `glStencilOpSeperate()` postavimo odgovarajuci nacin rada `stencil testiranja`. Postavit ćemo operaciju `stencil` spremnika prema metodi `Depth Fail`. Za razliku od jednostavnije metode koja se zove `Depth Pass`, `Depth Fail` nema problem kada je kamera unutar volumena sjene gdje bi inace došlo do krivog izračuna i mogućnosti preokretanja sjena. Kad je `stencil` spremnik postavljen pocinjemo sa sljedećim korakom gdje stvaramo volumen sjena.

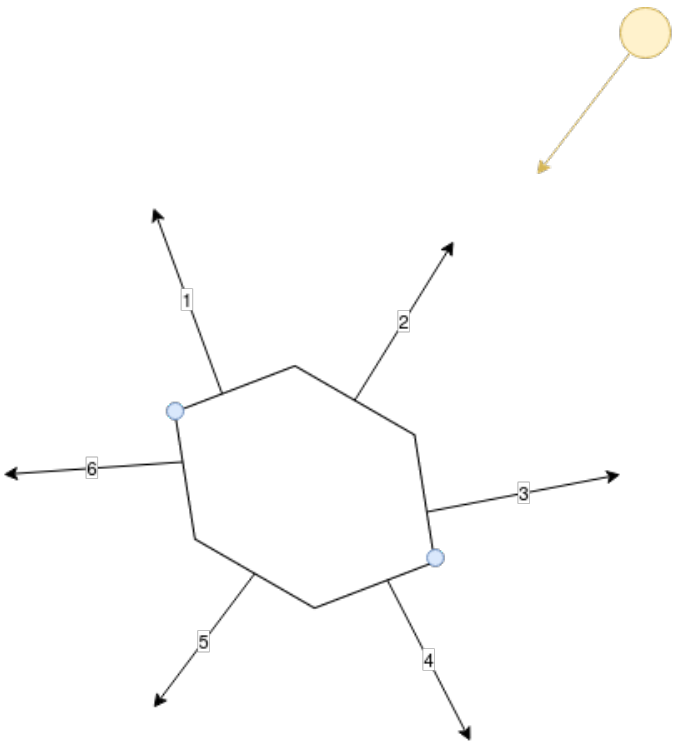
3.2.2 Izrada volumena sjena

Zbog efikasnosti volumen kreiramo u geometry shaderu prilikom crtanja u `stencil` spremnik. Tijekom crtanja scene prvo se nacrta objekt koji stvara sjenu a zatim , u geometry shaderu, stvorimo volumen sjene koju taj objekt stvara. Prije stvaranja volumena potrebno je odrediti siluteu objekta.

Kako bismo odredili da li je rub trokuta dio siluete ili ne, moramo pronaći susjedni trokut koji ima isti rub i izračunati skalarni produkt između smjera svjetlosti i normala i izvornog trokuta i njegovog susjeda. Rub se smatra diom siluete ako je jedan trokut okrenut prema svjetlu, a njegov susjed nije.

Na slici 3.3 crvena strelica predstavlja zrake svjetla koje padaju na objekt čije su normale 1, 2 i 3 (skalarni produkt između ovih normala i obrnutog vektora svjetlosti je veći od nule). Rubovi čije su normale 4, 5 i 6 okrenuti su od svjetla (ovdje bi isti točkasti produkt bio manji ili jednak nuli). Dva plava kruga označavaju siluetu objekta, a razlog je taj što je rub 1 okrenut prema svjetlu, ali njegov susjedni rub 6

nije. Točka između njih je dakle silueta. Isto vrijedi i za drugu točku siluete. Rubovi (ili točke u ovom primjeru) koji su okrenuti prema svjetlu kao i njihovi susjedi nisu silueta (između 1 i 2 i između 2 i 3).



Slika 3.3 Određivanje siluete na objektu.

Algoritam za pronalaženje siluete je jednostavan. Ali zahtijeva dodatnu informaciju što je da znamo susjedne točke svakog trokuta. Ovo je poznato kao susjedstvo trokuta(engl. Triangle adjacencies). U ovom radu koristimo objekte koji već imaju određene susjede pa necemo obraditi algoritam za određivanje susjeda.

Nakon što smo odredili siluetu, volumen sjene nastaje od siluete i njegovim produljenjem u beskonacnost. Taj volumen crtamo u `stencil` spremnik pomoću sljedećih jednostavnih pravila:

I) Ako test dubine ne uspije prilikom crtanja poligona volumena sjene okrenutih unazad, povećavamo vrijednost u `stencil` spremniku.

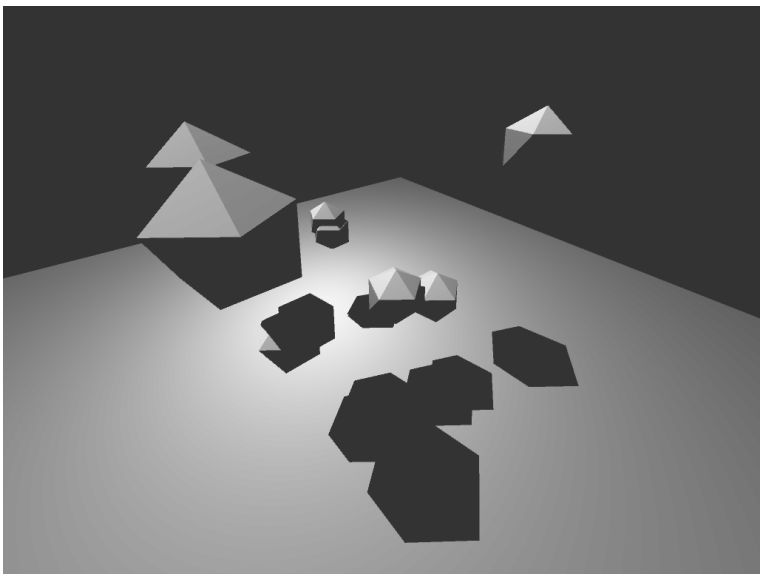
II) Ako test dubine ne uspije prilikom iscrtavanja prednjih poligona volumena sjene, smanjujemo vrijednost u `stencil` spremniku.

III) Ne poduzimamo ništa u sljedećim slučajevima: test dubine prošao, `stencil` test nije uspio.

Pomoću funkcija `glStencilOpSeperate()` možemo odrediti kako se vrijednost ažurira u `stencil` spremniku uzimajuci u obzir ta tri pravila.

3.2.3 Crtanje scene

U prijasnom koraku smo odredili koji se objekti u sceni nalaze unutar volumena sjene koristeci stencil spremnik. Kad smo odredili sve volumene za sve objekte u sceni onemogućujemo dalje ažuriranje stencil spremnika kako bi ostali netaknuti u daljem crtanju a ponovo se omogućuje ažuriranje spremnika dubine. Prijašnji korak je podijelio scenu na dva dijela. Dijelovi koji se nalaze u volumenu sjene, u stencil spremniku imaju vrijednost jedan. U zadnjem koraku pomoću te informacije scenu možemo u dva dijela nacrtati. Na prvom crtanju samo osvijetljeni dio scene nacrtamo, a na drugom crtanju, crtamo samo dio scene koji je u sjeni i to osvijetlimo samo sa ambijentalnim svjetlom.



Slika 3.4 Tehnika volumnih sjena.

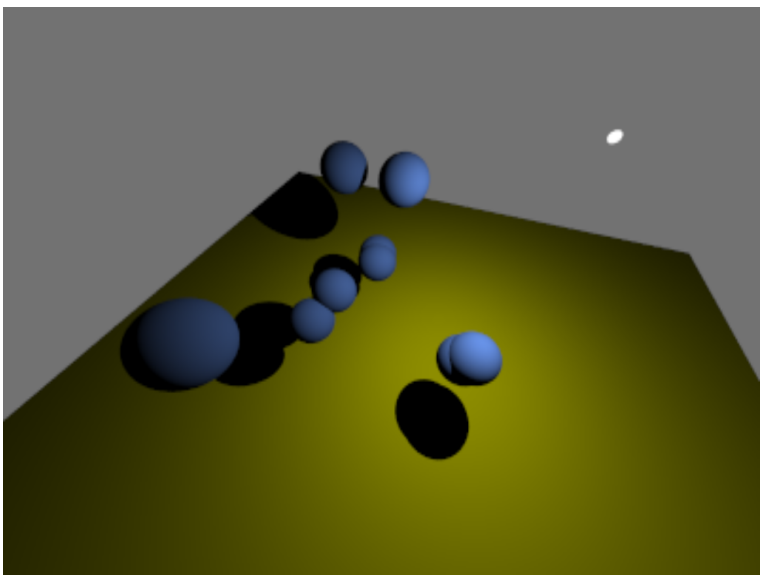
3.3 Praćenje zraka

Kod tehnike praćenja zraka dodati funkcionalnost određivanja sjena je znatno jednostavnije naprema prijašnjim metodama. Metodu crtanja scene ne trebamo znatno promijeniti, već koristimo funkcionalnost koja postoji i samo dodatno usmjerimo zrake. Kod crtanja scene bez usmjeravanja, zrake bi prirodno stvarale mjesta u sceni s manje svjetla gdje bi nastale meke sjene. To znatno ovisi o kompleksnosti scene i broju objekta u sceni. S vecom kompleksnoscu scene postoji veca sansa da mjesta ostanu bez svjetla i budu u sjeni. Problem s time jest da u sceni osim globalnog svjetla ne možemo simulirati druge izvore svjetla. Da bi riješili taj problem i istovremeno stvorili sjene mi zrake na pogotku s objektom usmjerimo prema izvoru svjetla i to pomoću funkcije gustoće vjerojatnosti PDF. PDF funkcija opisuje relativnu vjerojatnost da varijabla, u ovom slučaju zraka, poprimi određenu vrijednost. Sa PDF funkcijom zraku na pogotku usmjerimo prema svjetlu s određenim raspršivanjem u područje izvora svjetla. Da bi nacrtali sliku za svaki piksel prozora "ispucamo" zrako iz kamere prema sceni. Kada zraka dodje u kontakt sa objektom u sceni, zraka se odbije od objekta ovisno o materijalu objekta i normale u tocki kontakta i jos pomoću

PDF funkcije dodatno usmijerimo zraku prema svjetlu u sceni. Ako pri kontaktu vektorski produkt zrake iz kamere i vektora od kontakta do svjetla je manji od 0 onda smatramo da je u tocka u sjeni. Ako nije u sjeni zraku "odbijemo" i nastavimo sa praćenjem zrake do sljedećeg objekta ili izlaska iz scene.

3.3.1 Nedostaci

Kao prije spomenuto metoda praćenja zraka je mnogo zahtjevnija na resurse nego druge metoda i posto je implementirano da se svo računanje izvršava na procesoru (engl. central processing unit (CPU)) brzinu ne možemo usporedit s drugim metodama. Slično kao u metodi mapiranje sjena, kod praćenja zraka može doci do grafičkih greska kao što su akne i šum slike. Isto i nastaju zbog slicnog razlog, rezolucija slike je premalena i kompleksnih scena i naglih kuteva zraka. Oba problema možemo riješiti tako za svaki piksel umjesto jedne zrake "izbacimo" više zraka koji malo odstupaju jedan od drugoga i uzmemo prosjek svih. Time efektivo dodamo `antialiasing` u metodu praćenja zraka.



Slika 3.5 Tehnika pracenje zraka.

Poglavlje 4

Implementacija rješenja

4.1 Knjižice

Za implementaciju aplikacije koja rješava opisane probleme koristi se višeplatformska grafička specifikacija opengl (Open Graphics Library) kao i nekoliko pomoćnih biblioteka (glfw, glm, glad, stb, imgui). Navedena implementacija napisana je u c++ na operativnom sustavu linux. Glavni dijelovi programa odvojeni su u zasebne klase renderera koje sve nasljeđuje od iste apstraktne klase koja pruža sučelje za crtanje scene i korisničkog sučelja. Svaki renderer u konstrukciji postavlja sve što je potrebno za određenu tehniku čak i dijele neke zajedničke stvari renderera.

4.1.1 GLFW i Glad

Glfw je biblioteka koja pruža platformsko neovisni API koji se koristi za kreiranje prozora aplikacije, rukovanje unosom i stvaranje opengl konteksta. I Glad je pomoćna biblioteka koja se koristi za učitavanje opengl proširenja.

4.2 Renderer klase

Svi rendereri nasljeđu od apstraktne osnovne klase Renderer koja pruža virtualnu metodu `render()` i virtualnu metodu `draw_dialog()` koju će rendereri nadjačati.

Pomoću apstraktne klase olakšavamo prebacivanje između renderera tijekom izvođenja aplikacije, jasno definiramo sučelje za crtanje scene i olakšavamo separaciju renderera i njihovih ovisnosti.

4.2.1 Bazna klasa Renderer

Listing 4.1 Bazna klasa Renderer

```
class Renderer {
public:
  struct Transformations {
    int screen_width{};
    int screen_height{};
    float screen_aspect{};
    glm::mat4 projection{};
    std::shared_ptr<Camera> cam{};
  };

  Renderer() = default;
  virtual ~Renderer(){};
  virtual void render(Scene const &, Transformations const &,
      double ticks) = 0;
  virtual void draw_dialog() = 0;
};
```

Metoda render uzima scenu i strukturu transformacija kao argumente. `Scene` sadrži sve objekte i svjetla u sceni i samo pruža lakši način za rukovanjem tim objektima. Struktura `transformations` sadrži sve potrebne argumente potrebne za iscrtavanje scene poput matrice projekcije i globalne kamere.

4.2.2 Mape sjena

Listing 4.2 Klasa za mapiranje sjena

```
class ShadowMappingRenderer final : public Renderer {
public:
  ShadowMappingRenderer(std::filesystem::path root_dir,
                        int const a_shadow_width, int const
                           a_shadow_height);
  ~ShadowMappingRenderer() = default;
  void render(const Scene &, Transformations const &, double
     ticks) override;
  void draw_dialog() override { ImGui::Text("test"); }

private:
  Shader m_solid_shader;
  Shader m_depth_shader;
  Shader m_shadow_shader;

  GLuint m_depth_map_FBO;
  GLuint m_depth_cubemap;

  float near_plane = 1.0f;
  float far_plane = 25.0f;

  int const shadow_width{};
  int const shadow_height{};

  glm::mat4 m_shadow_proj;

  std::vector<glm::mat4> m_shadow_transforms;
};
```

Konstrukcija klase zahtijeva korijensku direktoriju te širinu i visinu teksture sjene. Korijenski direktorij, kao i kod ostalih metoda, koristi se za pronalaženje lokacija shader-a koji se učitavaju tijekom izvođenja programa. Tijekom konstrukcije klasa postavlja shadere i međuspremnik dubine koji će se kasnije koristiti tijekom prvog

koraka iscrtavanja scene. Posto koristimo pozicijsko svijetlo u sceni, koristiti ćemo teksturu kocke (engl. Cubemap) za spremnik dubine umjesto 6 zasebnih tekstura i postavimo sve potrebne parametre teksture.

Tijekom izvođenja aplikacije na aktivnom render ce se zvati funkcija `render()` koji iscrtava scenu s aktivnom metodom. U klasi mapi sjena funkciju `render()` možemo podijeliti na tri koraka. U prvom koraku moramo moramo ažurirati matricu transformacije koja se koristi za projekciju sjene, gdje svaku stranu cubemapa ažuriramo sa novom `view` matricom koja je usmjerena u određeni smjer ovisno o strani kocke.

Listing 4.3 Matrica transformacije

```
m_shadow_transforms[0] =
m_shadow_proj * glm::lookAt(light_pos,
                            light_pos + glm::vec3(1.0f, 0.0f,
                                0.0f),
                            glm::vec3(0.0f, -1.0f, 0.0f));
```

Nakon ažuriranja matrica u drugom koraku kao prethodno opisano prvo scenu nacrtamo u spremnik dubine. Prvo postavimo prozor za crtanje sa `glViewport()`, postavimo FBO i ocistimo spremnik sa `glClear()`. Nacrtamo sve objekte u spremnik dubine. I u zadnjem koraku opet resetiramo prozor, od-spojimo FBO i sada crtamo normalno scenu.

Shader

U analizi metode mapiranje sjena opisana je varijacija metode ovisno o vrsti izvora svjetla. Ta razlika se najbolje vidi u implementacija shadera. Prilikom crtanja pozicijskog svjetla potrebna nam je mapa kocke i sa ciljem efikasnijem izvršavanju koristimo geometrijske shadere.
U prvom koraku shader program korišten za direkcijsko svjetlo koristi samo vertex i fragment dio shadera. U vertex dijelu točku koju crtamo prebacimo u svjetski prostor sa matricom transformacije `model` i prije nego sto transformiramo u perspektivu

oka proslijedimo točku u fragment shader. U fragment dijelu shadera prvo prebacimo tocku koju zelimo crtati u perspektivu svjetla, uzmemo uzorak mape dubine, testiramo za sjenu i na kraju nacrtamo tocku.
U slučaju da koristimo pozicijsko svjetlo priliko koristenja shader programa doda se još jedan korak, geometrijski dio programa. Svrha geometrijskog programa jest da scenu nacrta u svim mogućim smjerovima, to postigne pomoću 6 matrica transformacija. Time znatno smanjimo potrebni broj crtanja scene.

Listing 4.4 Geometrijski shader

```
#version 330 core
layout (triangles_adjacency) in;
layout (triangle_strip, max_vertices=18) out;

uniform mat4 shadow_matrices[6];

out vec4 frag_pos; // FragPos from GS (output per emitvertex)

void main()
{
    for(int face = 0; face < 6; ++face)
        {
            gl_Layer = face;
            for(int i = 0; i < 3; ++i)
                {
                    frag_pos = gl_in[i].gl_Position;
                    gl_Position = shadow_matrices[face] *
                        frag_pos;
                    EmitVertex();
                }
            EndPrimitive();
        }
}
```

Na početku postavljamo raspored ulaznih podataka. Koristimo `triangles_adjacency`

jedino zbog jednostavnosti implementacije ostatka programa gdje sve metode koriste iste podatke. `triangles_adjacency` ne utječe na izvršavanje programa. U main funkciji programa iteriramo 6 lica cubemapa gdje svaku stranu navodimo kao izlazno lice pohranjujući redni broj lica u `gl_Layer`. Zatim generiramo izlazne trokute transformirajući svaki ulazni vrh iz svijeta u perspektivu svjetlosnog prostora množenjem `frag_pos` s matricom transformacije koju smo na početku crtanja postavili. Sve izračunate vrijednosti `frag_pos` proslijedimo u fragmet dio kamo ćemo ažurirati mapu sjena.

4.2.3 Volumne sjene

Klasa za metodu volumnih sjena je vrlo slična klasi za metodu mape sjena. Za razliku od mape sjena za konstrukciju nije potrebno dodijeliti rezoluciju teksture mape sjena jer volumne sjene koriste već zadane spremnike aplikacije. I imamo dodatne pomoćne funkcije.

Listing 4.5 Klasa za volumne sjene

```
class ShadowVolumeRenderer final : public Renderer {
public:
  ShadowVolumeRenderer(std::filesystem::path root_dir,
     Transformations const &);
  ~ShadowVolumeRenderer();
  void render(const Scene &, Transformations const &, double
     ticks) override;
  void draw_dialog() override;

private:
  Shader m_shadow_volume;
  Shader m_complete;
  Shader m_solid_shader;
  Shader m_first_pass;
  Shader m_material;
```

```
  void render_into_depth(Scene const &scene, Transformations
     const &trans);
  void render_shadow_volume_into_stencil(Scene const &scene,
                                         Transformations
                                            const &trans);
  void render_shadowed_scene(Scene const &scene,
     Transformations const &trans);
  void render_ambient(Scene const &scene, Transformations
     const &trans);
  void render_lights(Scene const &, Transformations const &);
};
```

Konstrukcija klase je znatno jednostavnija jer je samo potrebno inicijalizirati shadere. Funkcija `render()` je znatno kompliciranija naprema funkciji metode mapiranja sjena.

Listing 4.6 Funkcija za crtanje scene pomoću metode mapiranje sjena

```
void ShadowVolumeRenderer::render(const Scene &scene,
                                  Transformations const &
                                     trans, double ticks) {

  auto &light = scene.lights().at(0);
  auto const view = trans.cam->view();
  auto const screen_width = trans.screen_width;
  auto const screen_height = trans.screen_height;
  glDepthMask(GL_TRUE);

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT |
     GL_STENCIL_BUFFER_BIT);
  glClearStencil(0);
  glDisable(GL_STENCIL_TEST);
  glEnable(GL_DEPTH_TEST);
```

```
    render_into_depth(scene, trans);
    render_shadow_volume_into_stencil(scene, trans);
    render_shadowed_scene(scene, trans);
    render_ambient(scene, trans);

    glDisable(GL_BLEND);

    glDepthMask(GL_TRUE);
    glDepthFunc(GL_LEQUAL);

    glDisable(GL_STENCIL_TEST);
    render_lights(scene, trans);
}
```

Funkciju smo podijelili na vise pomoćnih funkcija koja svaka izvršava jedan korak u metodi. Kao i kod drugih na pocetku resetiramo prozor i sve spremniku ispraznimo. U ovom slučaju trebamo dodatno napomenuti i `GL_STENCIL_BUFFER_BIT` jer koristimo stencil spremnik za stvaranje volumena sjena. Nakon što smo sve postavili pozivamo funkcije za pojedinačne korake u metodi. Tako da kao prije opisano prvo nacrtamo scenu u spremnik dubine, stvorimo volumen sjena, crtamo osvijetljeni dio scene i nacrtamo dijelove u sjeni sa ambijentalnim svjetlom. Na kraju resetiramo postavke spremnika na zadane tako da crtanje radio kao uobičajeno i zadnje nacrtamo svjetla u sceni.

Listing 4.7 Pomoćna funkcija za crtanje osvijetljenog dijela scene

```
void ShadowVolumeRenderer::render_shadowed_scene(Scene const
    &scene,
                                                 Transformations
                                                     const &
                                                 trans) {
    glDrawBuffer(GL_BACK);

    glStencilFunc(GL_EQUAL, 0x0, 0xFF);
```

```
    glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

    m_complete.use();
    m_complete.set<vec3>("lightPos", scene.lights().at(0).
        translation());

    m_complete.set<vec3>("lightColor", vec3(1.f));

    m_complete.set<glm::mat4>("projection", trans.projection);
    m_complete.set<glm::mat4>("view", trans.cam->view());
    for (auto const &obj : scene.objects()) {
      m_complete.set<glm::mat4>("model", obj.model());
      m_complete.set<vec3>("objectColor", vec3(0.4f));
      obj.draw();
    }
}
```

funkcija `render_shadowed_scene` crta osvijetljeni dio scene nakon što smo generirali volumen sjena i ažurirali stencil spremnik i prijasnjem koraku. Pomocu `glStencilFunc` postavljamo da crtanje moze samo uspjeti ako je tocka koju crtamo točno nula sto znaci da nije u nekom volumenu i time nije u sjeni. Sa `glStencilOp` onemogućujemo ažuriranje stencil spremnika. Nakon toga iteriramo po svakom objeku u sceni i nacrtamo ga.

4.2.4 Praćenje zraka

Listing 4.8 Klasa metode praćenja zraka

```
class RayTracingRenderer final : public Renderer {
public:
  RayTracingRenderer(std::filesystem::path root_dir);
  ~RayTracingRenderer() {
    thread_cleanup();
```

```
    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);
    glDeleteBuffers(1, &EBO);
    destroy_realtime();
  };
  void render(const Scene &, Transformations const &, double
      ticks) override;
  void draw_dialog() override;
  void destroy() {
    destroy_realtime();
  }

  auto elapsed() {
      return std::chrono::duration_cast<std::chrono::
          microseconds>(elapsed_time);
  }

  struct RenderTaskArgs {
    std::shared_ptr<Camera> cam;
    size_t image_height;
    size_t image_width;
    int samples_per_pixel;
    int max_depth;
    int start;
    int stop;
  };

  enum ThreadTaskType {
    normal,
    realtime,
  };
  struct ThreadTask {
```

```
    ThreadTaskType type;
    Pixels *pixels;
    int index;
    std::atomic<bool> *thread_finished;
    std::atomic<bool> *should_finish;
    std::atomic<bool> *should_pause;
  };
...
```

Klasa za metodu praćenja zraka je znatno kompliciranija od ostalih jer osim potrebe za stvaranje sjena implementiramo i samu metodu crtanja slike. U ovom slučaju ne koristimo opengl direktno za crtanje nego već sami implementiramo ray tracing a opengl samo koristimo za prikaz krajnje slike.
U klasi dodatno definiramo pomoćne strukture koje koristimo za pozivanje ostalih funkcija. Sa strukturama grupiramo zajednicke argumente.
Konstrukcija ove klase relativno jednostavna, samo inicijaliziramo sve što je potrebno za prikazati sliku sa strane opengla. Crtanje scene se izvršava asinkrono na jednoj ili vise dretva.
Kad zelimo nacrtati sliku pozivamo funkciju `render_frame` koja kao argument uzima scenu koju želimo nacrtati.

Listing 4.9 Funkcija za crtanje slike

```
void RayTracingRenderer::render_frame(Scene const *scene) {
  Log::the().add_log("Starting Render\n");
  Log::the().add_log("Width=%zu, Height=%zu\n", m_image_width
      , m_image_height);
  Log::the().add_log("Threads=%d\n", m_n_threads);

  assert(a_camera);
  a_camera->update_rt_vectors();

  m_pixels.clear();
  m_pixels.resize(m_len);
```

```
    setup_threads({ThreadTaskType::normal, &m_pixels, 0,
        nullptr, nullptr},
                  scene);

    m_rendering = true;
}
```

Funkcija prvo ažurira sve vektore od kamere i isprazi spremnike za sliku, nakon toga poziva pomoćnu funkciju `setup_threads` koja kao što ime sugerira postavlja dretve koje u pozadini crtati slike. Pomocu dretva osiguravamo interaktivnost ostatka aplikacije. `setup_threads` uzima kao argument `ThreadTask` i scenu. U aplikaciji razlikujemo dvije vrste crtanja scene. Prva vrsta je kada zelimo samo jednom nacrtati scenu, to bi bio tip `normal`, i druga vrsta je kad želimo da se slika konstanto ažurira, tip `realtime`. Drugu vrstu mozemo interaktivno koristiti da naprimjer postavimo scenu a onda pozovemo prvi tip sa postavkama za veću kvalitetu da samo jednom nacrta. Tip odredujemo pomocu strukture ThreadTask gdje je prva varijabla tip crtanja, druga gdje spremin sliku i ostale varijable helper varijable u slucaju ako postavljamo vise dretva pa na pocetku budu postavljene na nula i nullptr.
U slučaju kada želimo samo jednom nacrtati sliku `setup_threads`pozovemo sa normalnim tipom crtanja. Onda ovisno o broju dretva koje smo postavili `setup_threads` podijeli sliku na jednake dijelove i stvori dretve sa granicama u kojima treba crtati. Svaka dretva poziva funkciju `ren_task;`

Listing 4.10 Funkcija praćenje zrake

```
void ren_task(RayTracingRenderer::ThreadTask task,
              RayTracingRenderer::RenderTaskArgs ra, Scene
                  const *scene) {
  auto image_height = ra.image_height;
  auto image_width = ra.image_width;
  int const len = image_height * image_width * 3;
  int samples_per_pixel = ra.samples_per_pixel;
  int max_depth = ra.max_depth;
```

```
int starting_index = ra.start * image_width * 3;
int index = starting_index;

assert(task.pixels);
Pixels *pixels = task.pixels;

auto type = task.type;
do {
  if (type == RayTracingRenderer::ThreadTaskType::realtime)
      {
    ra.cam->update_rt_vectors();
    if (*(task.should_finish)) {
      return;
    }
    if (*task.thread_finished || *task.should_pause) {
      std::this_thread::yield();
      continue;
    }
  }
  for (int j = ra.start; j <= ra.stop; ++j) {
    for (int i = 0; i < image_width; ++i) {
      if (type == RayTracingRenderer::ThreadTaskType::
          realtime) {
        if (*task.should_finish) {
          return;
        }
        if (*task.should_pause) {
          do {
            std::this_thread::yield();
          } while (*task.should_pause);
        }
      }
```

```
          color pixel_color(0, 0, 0);
          for (int s = 0; s < samples_per_pixel; ++s) {
            auto u = (i + random_float()) / (image_width - 1);
            auto v = (j + random_float()) / (image_height - 1);
            ray r = ra.cam->get_ray(u, v);
            pixel_color += ren_ray_color(r, scene, max_depth);
          }
          auto [x, y, z] = get_pixel_tuple(pixel_color,
              samples_per_pixel);
          (*pixels).at(index++) = x;
          (*pixels).at(index++) = y;
          (*pixels).at(index++) = z;
        }
      }
      index = starting_index;
      *task.thread_finished = true;
    } while (task.type == RayTracingRenderer::ThreadTaskType::
      realtime);
};
```

Ovo je glavni dio klase i obuhvaca crtanje slike. Na pocetku postavimo pomocne varijable i odredimo granice ako postoje vise dretva. Promatrajmo funkciju u slučaju da koristimo samo jednu dretvu. U tom slučaju većinu funkcije, dijelom koji se bavi sa održavanje asinkronih dretva, možemo zanemariti. Pomoću ugniježđenih for petlji prođemo svaki piksel i za svaki piksel bacimo zraku u scenu, u slučaju da `samples_per_pixel` je veci od jedan od vise zraka bacimo sa malim odstupanjem i na kraju zbrojimo pojedinačne zrake i dobijemo boju piksela. Boju rastavimo na komponente i zapišemo u spremnik slike. Da bi odredili boju koju dobijemo od zrake moramo pozvati funkciju `ren_ray_color`

Listing 4.11 Funkcija određivanja boje zrake

```
static color ren_ray_color(ray const &r, Scene const *world,
    int depth) {
```

```
hit_record rec;

if (depth <= 0)
  return color(0, 0, 0);

if (!hit_scene(r, world, depth, rec)) {
  return color(0.2f, 0.2f, 0.2f);
}

ray scattered;
color albedo;
color emitted = rec.mat_ptr->scatter->emitted();
float pdf;

if (!rec.mat_ptr->scatter->scatter(r, rec, albedo,
   scattered, pdf))
  return emitted;

auto &light = world->lights().at(0);
auto on_light = light.translation();
auto to_light = on_light - rec.p;
auto distance_squared = glm::length2(to_light);
to_light = glm::normalize(to_light);

if (glm::dot(to_light, rec.normal) < 0)
  return emitted;

float light_area = 2048.f * light.scale().x;
auto light_cosine = fabs(to_light.y);
if (light_cosine < 0.000001)
  return emitted;
```

```
    pdf = distance_squared / (light_cosine * light_area);
    scattered = ray(rec.p, to_light);
    return emitted + albedo *
                        rec.mat_ptr->scatter->scattering_pdf(r
                            , rec, scattered) *
                        ren_ray_color(scattered, world, depth
                            - 1) / pdf;
}
```

Pomoću funkcije `ren_ray_color` pratimo zraku u sceni i postavljena je kao rekurzivna funkcije. Na pocetku provjeravamo da li smo došli do maksimalne dubine odbijanja zrake i ako jesmo vratimo `color(0,0,0)` jer zraka je jos u sceni ali nije pogodila svijetlo pa jedino ostaje da je u sjeni. Nakon toga provjerimo da li je zraka izasla iz scene ako je vratimo boju pozadine. Dalje postavljamo pomoćne varijable za određivanje boje zrake i zrake koja ce se odbiti. Provjerimo da li je zraka u sjeni pomoću produkta. I na kraju odredimo smjer odbijanja zrake pomocu PDF funkcije i materijala pogodenog objekta i rekurzivno ponavljamo sve dok ne prođe limit dubine ili izađemo iz scene.

4.3 Analiza rješenja

Promatrajuci sve metoda, metoda mapiranje sjena je najjednostavnija za implementirati, efikasnija je od drugih. Efikasnost isto ne ovisi o kompleksnosti objekta u sceni. Nedostaci su: manja kvaliteta sjena zbog ograničene veličine tekstura, samosjenčanje gdje predmeti stvaraju sjenu na dijelovima scene gdje bi trebala biti osvijetljena. Treba voditi računa o vrsti izvora svjetla i najčešće je potrebno filtrirati sjene da bi se poboljšala kvaliteta.
Metoda volumnih sjena je zahtjevnija na resurse, kompleksnost raste brzi sa kompleksnoscu objekta jer je potrebno stvarati volumen ovisno o geometriji objekta. Moguce je samo crtati tvrde sjene. Potrebno je odrediti susjede svakom trokutu od objekta da bi mogli koristiti geometrijski shader. Prednosti volumnih sjena su bolja kvaliteta sjena, nije potrebno paziti na vrstu izvora svjetla i najjednostavnija

je implementacija za stvaranje sjena koje ne treba filtrirati. Pracenje zraka je teze usporediti sa ostalim metodama, gdje druge metode su znatno efikasnije i bojlje primjenjene u interaktivnim aplikacijama, metoda pracenje zraka ja red velicine manje efikasnija. To je zahvaljujuci što svo racunanje se izvršava na procesoru. Ali zato dobivamo veliki skok u realizmu sjena i slike. I dobivamo neke stvari "besplatno" kao sto su prozirni materijali, zamućenje dijela slike i slično.

4.3.1 Brzina izvođenja

U aplikaciji mjerimo vrijeme crtanja scene za svaku tehniku. U računalnoj grafici uobičajena mjera koja se koristi je `frames␣per␣seconds` (FPS), ali problem sa FPS nije dovoljno precizan pogotovo ako je u aplikaciji uključen `Vertical Synchronization` (vsync) gdje aplikacija čeka između crtanja da sinkronizira sliku sa ekranom. Bolja mjera u računanju brzine programa je koristeci `frame time`. `Frame time` se računa po sljedećoj formuli

$$frametime = \frac{1}{FPS} \tag{4.1}$$

Sve tehnike su mjerene sa istim scenama i istim tipom svjetla. Rezultati tehnika su prikazani u tablici:

Tablica 4.1 opis tabele

Tehnika	Vrijeme (us)
SM	248
SV	287
RT	700ms

Iz rezultata se vidi kako su tehnike mape sjena i volumnih sjena za istu jednostavnu scenu jednako efikasni dok pracenje zraka za isto treba, par reda veličine vise, 700ms da nacrta scenu. Ali to je samo ako koristimo jednu dretvu, kad koristimo 16 dretva treba samo 120ms. I dalje znatno vise nego druge metode.

Poglavlje 5

Zaključak

Razvojem industrije interaktivnih aplikacija, animacija i filmova porastao je zahtjev za, ne samo efikasnijim, nego i znatno realnijim sjenama. U ovom radu smo analizirali i implementirali dvije od najpopularnijih metoda za interaktivne aplikacije, mape sjena i volumne sjene. Usporedili smo im prednosti i nedostatke. Kako možemo nadograditi metodu da dobijemo bolje rezultate. Isto smo analizirali i implementirali metodu praćenja sjena koja je puno popularnija u industriji animacija i filmova.
Opisuju se implementacije metoda, moguci problemi koji mogu nastati i kako ih riješiti. Usporedili smo prednosti i mane između metoda i analizirali brzinu izvođenja. Iz rezultata zakljucujemo da metoda mapa sjena je jedan od najboljih izbora za interaktivne aplikacije zbog efikasnosti izvođenja i lakoce implementacij. Dok kad zelimo sto veci realizam onda jedan od kandidata je metoda praćenja sjena. Svaka od metoda ima jedno područje u kojem najbolje rade i zato nece ispasti iz uporabe.

Bibliografija

[1] Matt Pharr: "GPU Gems 2", Addison-Wesley, 2005

[2] https://learnopengl.com/, s interneta 2022

[3] https://ogldev.org/, s interneta 2022

[4] “Ray Tracing: The Rest of Your Life.” raytracing.github.io/books/RayTracingInOneWeekend.html, s interneta 2022

[5] “Ray Tracing in One Weekend.” raytracing.github.io/books/RayTracingInOneWeekend.html, s interneta 2022

Sažetak

Postupci ostvarivanja sjena obuhvaca skup tehnika pomocu kojih mozemo crtati sjene u aplikaciji uzimajuci u obzir njihove specificnosti. U ovom radu analizirane i implementirane su tehnike mape sjena, volumne sjene i pracenje zraka. Obraden je pristup izrade sjena pomocu tih tehnika pomocu graficke specifikacije opengl i implementacije tehnike pracenje zraka. Analizirani su prednosti i nedostaci tehnika u specificnim slucajevima, i analizirana je brzine izvodenja tehnike u usporedbi s drugim tehnikama.

Ključne riječi — **sjena, mape sjena, volmen sjena, pracenje zrake**

Abstract

Shadow creation procedures include a set of techniques with which we can draw shadows in an application, taking into account their implementation details. In this work, shadow maps, volume shadows and ray tracing techniques were analyzed and implemented. The approach of creating shadows using these techniques was done by using the opengl graphic specification and the implementation of the ray tracing technique. The advantages and disadvantages of the techniques in specific cases were analyzed, and the execution speed of the technique was analyzed in comparison with other techniques.

Keywords — **shadow, shadow map, shadow volume, ray tracing**