SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij računarstva

Završni rad

POSTUPCI OSTVARIVANJA SJENE

Mentor: doc. dr. sc. Jerko Škifić

Rijeka, svibanj 2017.

Tibor Jaklin

0069064214

Umjesto ove stranice umetnuti zadatak završnog rada.

Izjava o samostalnoj izradi rada

Temeljem stavka 1. članka 13. Pravilnika o završnog radu, završnom ispitu i završetku preddiplomskih studija Tehničkog fakulteta Sveučilišta u Rijeci izjavljujem da sam rad izradio samostalno sukladno ćlanku 9. istog pravilnika, a temeljem zadanog zadatka (602-04/16-04/46).

U Rijeci, 16. svibnja

_____

Tibor Jaklin

Zahvala

Zahvaljujem mentoru doc. dr. sc. Jerku Škifiću na dodjeli zanimljive teme te na korisnim sugestijama i usmjeravanjima za vrijeme pisanja ovog rada.

Sadržaj

POGLAVLJE 1

1. Uvod

Sjene, u prirodi, nastaju pod utjecajem nekog izvora svjetlosti. Kad izvor svjetlosti obasija neko područje, osvjetli ga. Sjena na nekom području zapravo nastaje kada zrake svjetla tamo nisu prisutne, drugim riječima, sjena je neprisutnost svjetla. Po izgledu sjene može se zaključiti nekoliko bitnih stvari: Može se odrediti relativnan smjer i udaljenost izvora svjetlosti. Isto tako, iz sjene se može odrediti i odnos više prisutnih predmeta koji bacaju sjene. Ova druga pogodnost je jako korisna u području računalne grafike.

Postupci ostvarivanja sjene podrazumjevaju tehnike koje se u području računalne grafike, a posebno u kontekstu 3D grafike, koriste kako bi se ostvario prikaz sjena u scenama radi postizanja realnosti. Sjene se najčešće koriste u izradi videoigara, filmova i animacija gdje je naglasak na realnom izgledu scene velik.

POGLAVLJE 2

2. Analiza problema

Cilj tehnika za ostvarivanje sjena je odrediti koja će područja u prikazanoj sceni biti osvjetljena, a koja ne. Kako je već spomenuto u uvodu, u ovom radu će biti opisane tehnike mapa sjena i volumnih sjena. Obje tehnike se oslanjaju na funkcionalnost OpenGL-a koja dopušta korištenje ugrađenih tekstura dubine (eng. depth texture) i matrice (eng. stencil texture) pored, automatski aktivne teksture, boje (eng. color texture).

2.1 Opis tehnika

2.1.1 Mape sjena

Tehnika mapa sjena temelji se na korištenju OpenGL-ovih tekstura za određivanje položaja sjena. Njezin rad se logički djeli na dva osnovna koraka [1].
“U tehnici mapa sjena razlikuju se dvije osnovne faze: izrada mape sjena i crtanje uz pomoć mape sjena” [2].

Ideja je nacrtati scenu iz perspektive izvora svjetla kako bi se saznalo koji dijelovi scene su vidljivi svjetlu. Sve što nije vidljivo iz perspektive svjetla biti će u sjeni, zato što do tih područja ne mogu doprijeti zrake svjetlosti. Informacije o tom području spremaju se u izrađenu teksturu, zvanu mapa sjena, koja se zatim koristi prilikom crtanja scene u slijedećem koraku.

Slijedi crtanje scene iz perspektive oka, pritom koristeći informacije iz izrađene mape sjena kako bi se odredilo je li piksel koji se trenutno crta vidljiv iz perspektive svjetla. Ukoliko nije, piksel se crta zatamnjen.

2.1.2 Varirajuće mape sjena

U uvodu rada spomenuto je kako je tehnika mapa sjena popularno rješenje za postizanje sjena koja je prisutna u grafičkoj industriji dugi niz godina. Sukladno napretcima u svijetu grafike rasli su i zahtjevi za unapređenje standardne tehnike mapa sjena. Tako su s ciljem minimiziranja određenih nedostataka standardne tehnike izrađene različite implementacije. Jedna od takvih unapređenih tehnika su varirajuće mape sjena (*eng. variance shadow mapping*, skraćeno VSM), čiji je cilj ostvariti “mekani” izgled sjena koristeći proizvoljne tehnike filtriranja. Takav “mekani” izgled pridodaje na realizmu programa u kojem se koristi.

Ideja VSM-a je spremiti informacije u mapu sjena na način da se omogući linearno filtriranje te teksture što se sa standardnom tehnikom mapa sjena ne može. Kada je linearno filtriranje omogućeno javlja se niz novih opcija; opcija kako filtrirati mapu sjene. Filtriranjem mapa sjena se sadržane vrijednosti "zamagljuju" te je prikaz same sjene nakon crtanja puno “mekaniji”.

2.1.3 Volumne sjene

Tehnika volumnih sjena se baš kao standardna tehnika mapa sjena koristi dugi niz godina. Volumne sjene se ne koriste toliko često, a razlog tome je veći broja nedostataka, a najbitnije je spomenuti da je zahtjevnija za resurse. Kompleksnost implementacije ove tehnike je veća čim je veća kompleksnost modela koji bacaju scenu. Veći broj poligona u modelu zahtjeva veću snage računala.

Ideja tehnike je koristeći informacije o položaju izvora svjetlosti i položaju svih predmeta u sceni na koje to svjetlo djeluje izraditi volumen sjena uz čiju će se pomoć određivati koja područja scene će se crtati u sjeni. Volumen sjena nije ništa drugo nego dodatan predmet u sceni čiji se oblik određuje iz predmeta na koje svjetlo djeluje, a izgled odgovara izgledu sjene koje bi taj predmet bacao. Nakon što se volumen sjene nacrta provjerava se koji se predmeti iz scene nalaze u tom volumenu. Oni predmeti koji se nalaze u volumenu biti će u sjeni. Nakon određivanja koji će se predmeti zatamniti vrši se proces ponovog crtanja scene u kojem se zasebno crtaju osvjetljeni djelovi scene i djelovi u sjeni.

2.2 Izrada tekstura

Teksture je nužno razumijeti da bi se sve opisane tehnike izrade sjena mogle ispravno koristiti. OpenGL razlikuje teksture s obzirom na *format* i *vrstu* teksture. Prema formatu se dijele na teksture boje, dubine i matrice (eng. color, depth, stencil textures).

Framebuffer (FBO) je OpenGL-ova struktura koja sadrži teksture. Funkcija FBO strukture je određivanje u koje će se teksture pisati prilikom poziva funkcije za crtanje scene. Jedan FBO objekt može sadržavati najviše jednu dubinsku, jednu matričnu, i sedam tekstura boja. Svaka OpenGL aplikacija nakon svoje inicijalizacije automatski stvara tri teksture, po jednu za svaki format, koje su pohranjene u zadani FBO. Međutim, od spomenute tri teksture program automatski dozvoljava pisanje samo u teksturu boje. Kako bi se pisalo u ostale dvije teksture potrebno je pozvati odgovarajuće funkcije koje će aktivirati izmjenu njihovih vrijednosti.

Programeru OpenGL aplikacije je omogućeno stvaranje i korištenje velikog broja tekstura koje se mogu koristiti za postizanje raznih efekata, a isto tako mu je omogućeno i stvaranje FBO objekata. Prilikom izrade teksture moguće je postaviti razne parametre koji će odrediti njen način

rada. Rezolucija teksture također se određuje pri njenom stvaranju. Bitno je napomenuti da rezolucija zadanog FBO-a odgovara veličini prozora.

Teksture se prilikom izrade formatiraju ovisno o željenoj funkciji koju će obnašati u programu. Različiti formati razlikuju se u broju i tipu varijabli koje sadrže. Tekstura boje sadrži ukupno četiri float vrijednosti, od kojih prve tri opisuju udjele crvene, plave i zelene boje u elementu teksture (eng. texel – texture element), a četvrta predstavlja prozirnost tog elementa. Dubinska tekstura sadrži jednu float vrijednost koja predstavlja informaciju o dubini tog elementa teksture. Matrična tekstura sadrži jednu osam bitnu vrijednost stoga element takve teksture može sadržavati 256 različitih vrijednosti.

2.2.1 Dubinska tekstura

Kao što je ranije spomenuto, dubinsku teksturu u zadanom FBO-u je potrebno aktivirati kako bi se dozvolilo pisanje u nju. Slično kako tekstura boje sprema informacije o boji nekog piksela, tako dubinska tekstura čuva vrijednost dubine, odnosno, udaljenosti piksela od kamere.

Dubinska tekstura koristi se u dubinskom testiranju (eng. depth testing). Cilj takvog testiranja je smanjiti broj nepotrebnog crtanja u isti piksel.

Pri crtanju scene bez dubinskog testiranja potrebno je obratiti pozornost na redoslijed crtanja predmeta – svako crtanje upisuje nove vrijednosti u teksture nekog piksela bez obzira na njihove postojeće vrijednosti. Ukoliko se koristi dubinsko testiranje program će voditi brigu o trenutnoj vrijednosti u dubinskoj teksturi piksela te će pisati u aktivne teksture samo ako je određeni uvjet zadovoljen. Uvjet koji mora biti zadovoljen ovisi o željenom načinu rada, no u ovom će se radu koristiti zadani način rada koji određuje da se pisanje u teksture vrši ako je nova vrijednost dubine fragmenta manja od trenutne. To znači da će se predmet u sceni crtati samo ako je bliži od trenutno nacrtanog predmeta na istom položaju. Time redoslijed crtanja predmeta postaje nebitan što ide u korist programeru.

2.2.2 Matrična tekstura

Pisanje u matričnu teksturu se u zadanom FBO-u također treba aktivirati prije korištenja. Ovaj tip teksture se koristi u tehnici matričnog testiranja (eng. stencil testing) čiji je cilj podjeliti scenu na nekoliko skupina, obično su to samo dvije. Takvom podjelom scene na skupine postaje moguće kontrolirati koji će se djelovi scene crtati, a koji neće.

Tehnika se dijeli na dva koraka:

1) Pisanje u matričnu teksturu čime se scena djeli na više logičkih skupina

2) Korištenje nastale podjele kako bi se kontroliralo što crtati

U ovom će se radu tehnika matričnog testiranja koristiti prilikom implementacije volumena sjena gdje će se scena podjeliti na dvije logičke cjeline.

U prvom se koraku isključuje pisanje u testuru dubine i boje te se vrši crtanje proizvoljnog predmeta u matričnu teksturu. Prilikom crtanja se vrši usporedba trenutnih vrijednosti u matričnoj teksturi i proizvoljne referentne vrijednosti. Ovisno o rezultatu navedene usporedbe omogućuje se izmjena trenutne vrijednosti u matričnoj teksturi. Također, ukoliko je aktivirano dubinsko testiranje, moguće su dvije dodatne izmijene vrijednosti u matričnoj teksturi ovisno o rezultatu dubinskog testiranja. U ovoj fazi prvog koraka se pisajući proizvoljne vrijednosti u teksturu scena dijeli na logičke skupine: ona koja sadrža tu vrijednost i ona koja sadrži zadanu vrijednost.

Nakon izmjena vrijednosti u prvom koraku onemogućuje se pisanje u matričnu teksturu, a zatim slijedi drugi korak koji podrazumijeva uobičajeno crtanje scene. Prije samog crtanja nužno je odrediti vrijednost koju matrična tekstura piksela mora sadržavati da bi se u njih crtalo.

Odabir tih vrijednosti operacije usporedbe vrši se koristeći ugrađene funkcije *glStencilOp()*, a *glStencilFunc()*.

2.3 Tipovi izvora svjetlosti u aplikaciji

U OpenGL aplikaciju moguće je implementirati nekoliko tipova izvora svjetlosti koji se međusobno razlikuju po svom načinu rada, no u ovom će se radu opisati samo dva korištena tipa koji su direkcijsko svjetlo i pozicijsko svjetlo. Oba tipa moraju imati zadane određene informacije kako bi njihova implementacija radila ispravno. Izvor svjetlosti treba imati određeni položaj u sceni, smjer djelovanja te vrijednosti koje opisuju boju i intenzitet kojima svjetlo utječe na scenu.

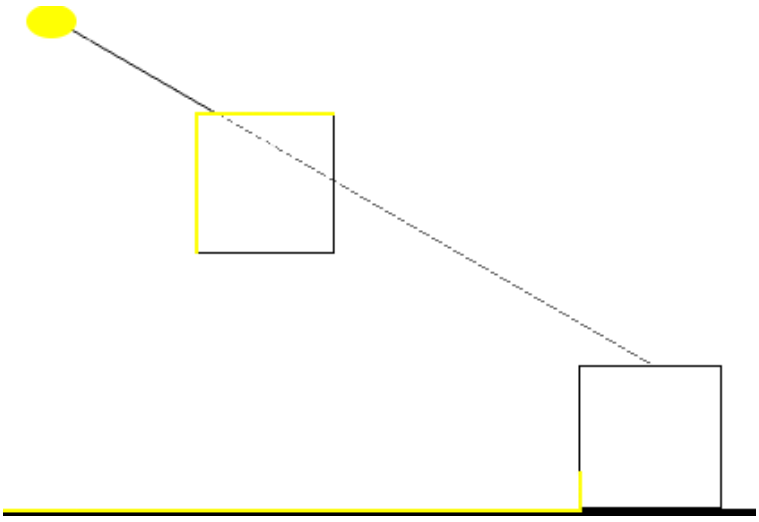
Direkcijsko svjetlo je tip svjetla koje imitira rad sunca. Određuje se samo zadavanjem smjera djelovanja, a za njegov položaj se podrazumjeva da je beskonačno udaljen od scene u zadanom smjeru. Kako je položaj takvog svjetla beskonačno daleko, njegove zrake koje padaju na scenu biti će paralelne.

Pozicijsko svjetlo, za razliku od direkcijskog, ima određen položaj u sceni stoga njegove zrake neće biti međusobno paralelne. Cilj pozicijskog svjetla je imitirati rad žarulje, stoga

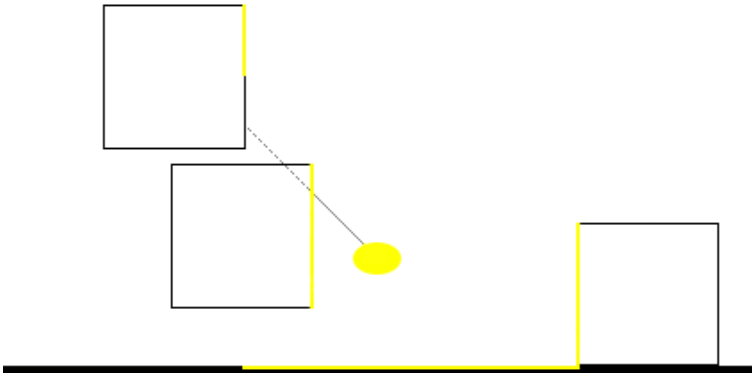
njegove zrake utječu na sve predmete u sceni, a kut pod kojim padaju ovisi o položaju predmeta u pitanju i položaju izvora svjetlosti. Na slici 2.1 prikazane su različiti utjecaji spomenutih svjetla.

2.3.1 Ovisnost tekstura o izvoru svjetlosti

U radu je već objašnjeno formatiranje tekstura ovisno o željenoj funkciji. S obzirom na vrstu teksture OpenGL razlikuje nekoliko različitih tipova. U ovom radu koristiti će se samo dva od nekoliko ponuđenih tipova, a to su dvodimenzionalne (2D) teksture i *cube maps*. C*ube maps* je vrsta teksture koja se sastoji od šest 2D tekstura logički povezanih tako da čine lica modela kocke. U ovome radu će biti potrebno koristiti šest 2D tekstura prilikom implementacije tehnike mapa sjena, no umjesto izrađivanja šest nepovezanih 2D tekstura, biti će korištena funkcionalnost *cube mapa* zbog svojih prednosti. Prednost korištenja *cube mapa* proizlazi iz stvaranja logičkog rasporeda sadržanih tekstura koji omogućuje olakšano čitanje spremljenih vrijednosti. Spremljene vrijednosti dohvaćaju se korištenjem trodimenzionalnih vektora, za razliku od korištenja dvodimenzionalnih vektora kod 2D tekstura. Trodimenzionalni vektor određuje smjer, a ne položaj, u kojem se nalazi željena vrijednost. Korištenje smjera umjesto točnog položaja je moguće jer se u nekom smjeru nalazi točno jedna vrijednost.



a) Direkcijsko svjetlo



b) Pozicijsko svjetlo

Slika 2.1 Utjecaji izvora svjetlosti na scenu

Žuto označeni rubovi predmeta predstavljaju osvijetljeno područje, a crni područje u sjeni.

POGLAVLJE 3

3. Opis rješavanja zadatka

U ovom će poglavlju biti opisana implementacija opisanih tehnika te neki od nedostataka te ponuđeno minimizirane istih.

3.1 Standardne mape sjena

Kako je ranije spomenuto, mapa sjena izrađuje se koristeći teksture stoga je prije prvog koraka potrebno izraditi novu teksturu, najčešće dubinskog tipa, koja će se koristi pri crtanju scene. U standardnoj implementaciji mapa sjena običajeno je koristiti dubinsku teksturu zato što je informacija koja se sprema u nju dubina piksela. Izrađenu teksturu sprema se u novi FBO objekt kako bi se omogućilo pisanje u nju.

Pri izradi teksture postoje razlike između različitih implementacija tehnike. Neke nove tehnike koje nadograđuju standardnu mapu sjena zahtjevaju korištenje teksture boje kako bi postigle željene rezultate.

U prethodnom poglavlju spomenuta su dva osnovna koraka tehnike mape sjena: izrada mape sjena i crtanje uz mapu sjena. U ovom potpoglavlju biti će opširnije opisane obje tehnike.

3.1.1 Korak izrade mape sjena

U koraku izrade mape sjena se prije crtanja scene aktivira novo izrađeni FBO objekt koji sadrži dubinsku teksturu, te se koristeći funkciju *glViewport()* određuje u koji dio prozora će se vršiti crtanje. Veličina određena *glViewport()* funkcijom mora odgovarati rezoluciji dubinske teksture. Nakon što je postavljen aktivni FBO te je određeno područje pisanja funkcijom *glClear()* brišu se trenutne vrijednosti iz aktivnih tekstura i aktivira se odgovarajući shader program koji će se koristiti za crtanje scene.

Da bi shader program radio ispravno potrebno mu je prosljediti sve informacije koje će koristiti. Kako je cilj prvog koraka tehnike izraditi mapu sjena, informacije koje će se slati shader-u biti će transformacijske matrice za odgovarajuće predmete koji se crtaju.

3.1.2 Korak crtanja uz mapu sjena

U slijedećem koraku crta se scena uz pomoć izrađene mape sjena. Crtanje se vrši iz perspektive oka što zahtjeva korištenje novih transformacijskih matrica. Nove transformacijske matrice će određivati kako će scena biti prikazana korisniku programa. Kako bi se postiglo uobičajeno crtanje, u prozor programa, potrebno je aktivirati zadani FBO objekt te postaviti *glViewport()* na vrijednost veličine prozora. Prije samog crtanja potrebno je aktivirati korištenje teksture mape sjena, aktivirati shader program te u njega proslijediti odgovarajuće matrice korištene za transformaciju scene u perspektivu svjetla.

3.1.3 Pristup korištenju teksture mapa sjena

Prilikom crtanja u prvom koraku svi se predmeti u sceni transformiraju koristeći *view* i *projection* matrice za izvor svijetla (*lightV*, *lightP*). Time se ostvaruje prikaz scene iz perspektive izvora svijetla, a informacije o dubini svakog piksela spremaju se u dubinsku teksturu izvršavajući shader program. Zatim se izrađena mapa sjena zajedno s *lightV* i *lightP* šalje u shader program drugog koraka. U drugom se koraku koristi drugi par *view* i *projection* matrica (*eyeV*, *eyeP*) koji transformira scenu u perspektivu oka. Kako bi se pri crtanju piksela, u tom drugom koraku, moglo odrediti je li određeni fragment u svjetlu ili sjeni taj se fragment mora transformirati dva puta. Prvi puta se transformira koristeći par *eyeV* i *eyeP* matrica, a drugi puta koristeći *lightV* i *lightP* matrice. Transformacijom prvim parom dobije se željeni prikaz scene u prozoru. Transformacijom drugim parom matrica se fragment smješta na odgovarajući položaj u perspektivi svjetla. Nakon te transformacije koristi se fragmentova x i y vrijednost kako bi se dohvatio sadržaj teksture na poziciji fragmenta. Dohvaćena vrijednost je dubina najbližeg predmeta svjetlu na položaju fragmenta. Ukoliko je vrijednost dohvaćena iz teksture jednaka ili veća od dubine fragmenta, fragment je taj najbliži vidljivi predmet svjetlu stoga će biti osvjetljen. Ukoliko je vrijednost manja od dubine fragmenta, fragment će biti u sijeni jer se nalazi iza najbližeg predmeta.

Obje transformacije fragmenta i provjera dubine vrši se u shader programu drugog crtanja.

3.1.4 Razlike ovisne o tipu svjetla

Prilikom crtanja u prvom koraku bitno je obratiti pozornost na pravilno postavljanje zakonitosti rada svjetla. Položaj izvora svjetla te širina njegovog *frustuma* mora biti pažljivo određena kako bi ono imalo željeni utjecaj na scenu.

Izgled shader programa za dubinski prolaz tehnike bitno ovisi o tipu svjetla koje djeluje na scenu. Za izradu mape sjena pod utjecajem direkcijskog svjetla koristi se jednostavna dubinska 2D tekstura. Razlog tome je što je položaj takvog tipa svjetla beskonačno udaljen od scene stoga je iz njegove perspektive vidljivo cijelokupno područje na koje svjetlo utječe, to jest, vidljiva je cijela scena.

Za izradu mape sjena u sceni pod utjecajem pozicijskog svjetla nije dovoljno koristiti samo jednu 2D teksturu. Razlog tome je što se iz perspektive pozicijskog svjetla ne vidi cijelokupno područje na koje ono djeluje, za razliku od direkcijskog svjetla. Za rješavanje tog problema moguće je izraditi mapu sjena koristeći šest tekstura, gdje će svaka gledati u određenom smjeru kako bi se zabilježile sve informacije o pordučju pod utjecajem svjetla. Umjesto izrade zasebnih šest tekstura, u ovom radu će se koristiti tip teksture mapa kocke zbog svih, ranije opisanih, pogodnosti.

Dakle, korištenjem *cube mape*, upisuju se informacije o vidljivom dijelu scene u šest 2D tekstura. Cilj je prikazati utjecaj svjetla na scenu, a svaka od tekstura će sadržavati informacije o dubini scene u jednom od šest odgovarajućih smjerova.
Kada se crtanje u teksture završi dobije se gotova tekstura tipa *cube mape* koja logičkim raspoređivanjem svojih šest dijelova omogućuje intuitivno dohvaćivanje vrijednosti.

Korištenje teksture *cube mape* traži ispunjavanje svih šest tekstura pritom koristeći različite transformacijske matrice, što znači da je scenu u prvom koraku potrebno crtati šest puta, svaki puta koristeći različite transformacijske matrice. No, koristeći OpenGL-ovu tehnologiju geometrijskih shadera moguće je ispuniti sve teksture u samo jednom crtanju.

3.1.5 Nedostaci tehnike i njihovi ispravci

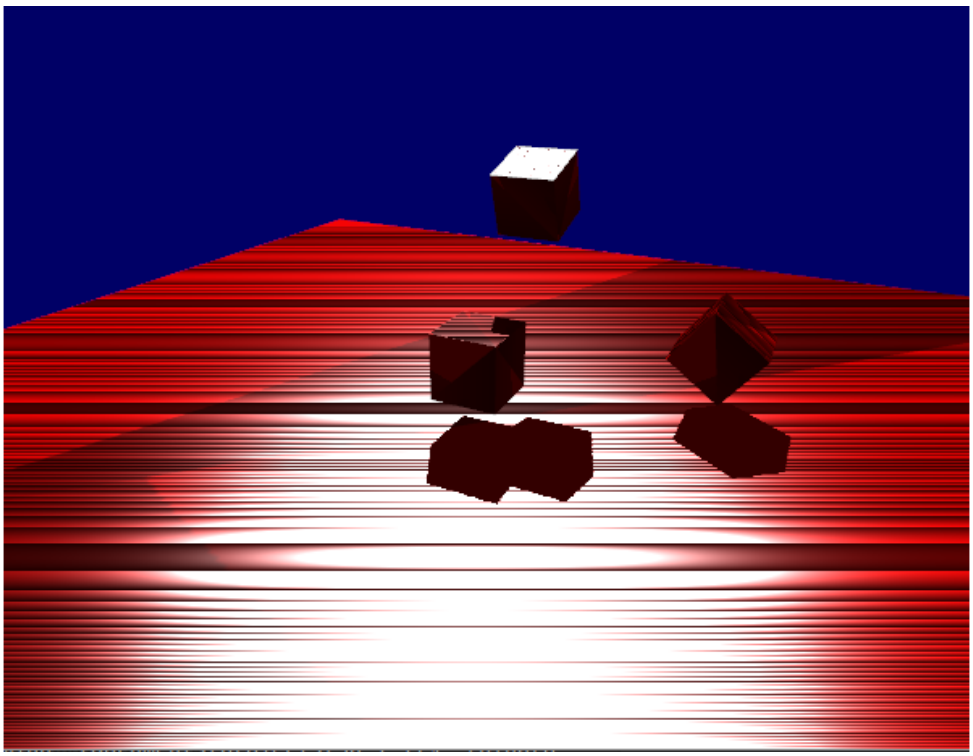
Opisana je osnovna tehnika mapa sjena koja omogućuje crtanje sjena u odgovarajuće dijelove scene, ovisno o utjecaju izvora svjetlosti. No, osnovna tehnika mapa sjena podliježe uočljivim greškama u prikazu.

Velik broj grešaka u mapama sjena nastaje pri odabiru rezolucije dubinske teksture. Ovisno o rezoluciji teksture, udaljenosti svjetla i kutu pod kojim svjetlo baca zrake na scenu javljaju se određene greške. Najuočljivija greška koja se javlja naziva se akne sjena.

Rezolucija teksture koja se koristi za izradu mape sjena je u pravilu vrlo ograničena, s ciljem uštede resura dostupnih aplikaciji, zbog čega se prilikom crtanja u jedan njezin teksel obuhvati vrijednost nekoliko stvarnih piksela. “Kada bi rezolucija mogla biti beskonačno velika svaki teksel odgovarao bi najviše jednom stvarnom pikselu u sceni”[1]. Čitanjem vrijednosti iz teksture, pri izvršavanju drugog koraka, se za više stvarnih piksela u sceni uzima jedna vrijednost. Ukoliko je kut pod kojim zrake svjetlosti padaju na predmet velik greška postaje uočljivija. Prugasti uzorak vidljiv na nacrtanoj sceni posljedica je netočnog “podudaranja” dubina piksela u sceni i teksela u teksturi, zbog koje jedna grupa piksela učita vrijednosti koje se prema teksturi nalaze ispod podloge, dok drugi učitaju vrijednost iznad podloge. Izgled problema prikazan je na slici 3.1.

Ovaj problem rješava se koristeći male float vrijednosti zvane *bias* koja se oduzme od vrijednosti dubine fragmenta u obradi kako bi se promatrao bliži fragment. Odgovarajuća vrijednost *bias* varijable zadaje se uzimajući u obzir rezoluciju teksture, ali i kut pod kojim zrake svjetlosti padaju na predmet u pitanju. Ukoliko se za *bias* uzme pre velika vrijednost javiti će se novi problem, nazvan *peter panning*, koji će prouzročiti “odvajanje” sjene od predmeta koji je baca. Za rješavanje takvog problema u prvom se koraku koristi tehnika *Front Face Culling* koja ne crta prednja lica predmeta, već samo stražnja lica. Time se problem miče s prednjih lica na stražnja, koja su ionako već zatamnjena jer su okrenuta od svjetla.

Pretjerano kvadratičast izgled sjena posljedica je korištenja niske rezolucije za teksturu mape sjena. Koristeći teksturu više rezolucije moguće je ispraviti ovakav problem, no viša rezolucija zahtjeva više resura od računala tako da je bitno voditi računa o željenom omjeru kvalitete i brzine. Takav oštar izgled sjene moguće je dodatno popraviti korištenjem filtrirajućih tehnika u shader programu. U ovom radu koristi se PCF filtrirajuća tehnika.



Slika 3.1 Akne sjena, greška u implementaciji mapa sjena

3.1.6 PCF filtriranje

Percentage-closer filtering (eng. PCF) je jedna od filtrirajućih tehnika koje se pretežito koriste za unaprijeđenje tehnike mapa sjena. Tehniku se implementira u shader program koji se izvršava u drugom koraku mapa sjena. Ideja tehnike je za svaki fragment koji se obrađuje čitati više vrijednosti iz mape sjena. Svako čitanje vrši se koristeći različite koordinate relativno udaljene od obrađivanog fragmenta. Za svaku se pročitanu vrijednost provjeri je li u sjeni. Zatim se računa prosječna vrijednost rezultata svih provjera koja se koristi kao faktor zatamnjivanja područja. Time se postiže sjena koja prema rubovima postaje postepeno svjetlija.

"Moguće je postići glađi izgled sjena čitajući veću regiju piksela, a time i veći broj piksela iz teksture" [4]. No, za čitanje veće regije potrebno je koristiti teksturu veće rezolucije kako bi izgled sjene zadržao realan izgled.

Nedostatak ove tehnike je da je iznimno spora jer, da bi rezultat izgledao bolje, potrebno je koristiti veći broj dodatnih čitanja iz teksture. Primjerice, za implementaciju PCF tehnike koja za svaki piksel koji obrađuje čita 9 dodatnih vrijednosti iz teksture. Također, koristeći ovu tehniku svi nedostaci iz standardne tehnike (akne sjena) postaju uočljiviji. Izgled sjena izrađenih standardnom tehnikom sa i bez PCF filtriranja prikazan je na slici 3.2.

Slika 3.2 Usporedba standarne mape sjena i mape sjena s PCF-om

3.2 Varirajuće mape sjena

Razlika između standardne i varirajuće tehnike je u pristupu izrade teksture mape sjena u prvom koraku te dodatni korak koji slijedi izradu teksture, a u kojem se vrši filtriranje te teksture. Tehnika VSM se dijeli u tri logičke cjeline:

1) Dubinski prolaz (eng. depth pass)

2) Filtriranje mape sjena (eng. blur pass)

3) Prolaz crtanja (eng. render pass)

3.2.1 Dubinski prolaz

Prvi korak ove tehnike sličan je prvom koraku standarne tehnike mapa sjena. Potrebno je izraditi novu teksturu boje te FBO objekt kojemu će tekstura pripadati.

Jedan od zahtjeva ove tehnike je da se tekstura sastoji od barem dvije komponente. Iz tog razloga je odabrana tekstura formata boje, koja nudi četiri komponente, no kako se zadnje dvije neće koristiti moguće je od OpenGL-a zatražiti format sa samo dvije komponente.

Prije početka pisanja potrebno je aktivirati izrađeni FBO objekt, postaviti veličinu područja u koje će se crtati, koristeći *glViewport()***,** tako da odgovara rezoluciji korištene teksture te je potrebno ograničiti pisanje u prve dvije komponente teksture boje koristeći ugrađenu funkciju *glColorMask()*. Nakon postavljanja svih uvjeta izvodi se crtanje scene iz perspektive izvora svjetlosti, što se postiže koristeći transformacijskih *view* i *projection* matrica.

Izvršavanjem samog crtanja u shader programu transformira se scena u perspektivu izvora svjetlosti te se u teksturu upisuju dvije informacije. U prvu komponentu teksture se piše vrijednost dubine, a u drugu kvadrirana vrijednost dubine. Takvim pristupom izradi mape sjena omogućuje se linearno filtriranje mape.

3.2.2 Filtriranje mape sjena

U drugom koraku se izvršava filtriranje mape sjena. Cilj filtriranja je, ovisno o odabranoj tehnici filtriranja, relativno izmijeniti vrijednosti u mapi. U implementaciji tehnike VSM mapa sjena se filtrira koristeći *gaussian blur* funkciju.

3.2.3 Gaussovo zamagljivanje

Gaussian blur postiže zamutnjavanje obrađivane teksture. Intenzitet kojim blur funkcija zamagljuje teksturu ovisi o odabranoj dimenziji *kernela*. Pojam *kernel* označava broj susjednih piksela koji će se uzimati u obzir pri određivanju nove vrijednosti trenutno obrađivanog piksela. Tehnika *gaussian blur* radi na principu da prolazi kroz svaki piksel te mijenja vrijednosti svakog kanala njegove teksture boje, gledajući svaki kanal zasebno. Vrijednost kojom će se zamjeniti postojeća u određenom kanalu teksture računa se na temelju vrijednosti sadržanih u istom kanalu svih susjednih piksela određenih u kernelu.

"Mapa sjena se, umjesto jednom, filtrira dva puta iz razloga da se kompleksnost algoritma svede s $O(n^2)$ na $O(n) + O(n)$" [5]. Prvi puta filtrira se horizontalno, a drugi puta vertikalno. Kako u OpenGL-u nije moguće koristiti istu teksturu čiji se podaci filtriraju za spremanje tih filtriranih podataka morati će se izraditi dodatna tekstura (filtarna tekstura) za tu svrhu. Ta filtarna tekstura biti će jednaka teksturi mape sjena, ali će biti različite rezolucije ovisno o željenoj kompleksnosti filtriranja. U slučaju ovog rada, filtarna tekstura biti će jednaka teksturi mape sjena.

Prilikom dva filtriranja teksture koriste se malo drugačiji podaci. Kernel i njegove vrijednosti su u oba slučaja jednaki, no kernel vrijednosti (položaji susjednih piksela) se množe s varijablom koja je različita u svakom od dva koraka. Ta varijabla predstavlja veličinu jednog teksela u teksturi u koju će se spremiti filtrirani podaci.

3.2.4 Prolaz crtanja

Nakon što je mapa sjena filtrirana poželjnom tehnikom, spremna je za korištenje. Prije crtanja scene aktiviraju se odgovarajući FBO i teksture za korištenje, odabire se shader program te se u njega šalju transformacijske matrice. Shader program sadrži funkciju za izračun sjena specifičnu za ovu tehniku.

3.2.5 Čebiševljeva nejednakost u izračunu sjena

Funkcija se temelji na teoremu Čebiševljeve nejednakosti. Ukratko, teorem opisuje koliki se postotak brojeva, u nekom skupu brojeva s poznatom srednjom vrijednosti i varijancom, nalazi iznad određene vrijednosti te koliko ih se nalazi ispod te vrijednosti.

$$P(x \geq t) \leq p_{max}(t) = \frac{\sigma^2}{\sigma^2 + (t-\mu)^2} \tag{3.1}$$

Varijabla μ iz funkcije označava srednju vrijednost, a σ^2 označava vrijednost kvadrirane varijance [4].

U shader programu se iz shadow map teksture dohvaćaju vrijednosti iz prvog i drugog kanala za trenutno obrađivani fragment. Vrijednost iz prvog kanala, filtrirana vrijednost dubine, sprema se u varijablu μ, a varijablu σ^2 razlika vrijednosti iz drugog kanala i kvadrirane vrijednosti iz prvog kanala. Zatim se dvije varijable iskoriste za izračun opisane čebiševljeve formule. Dobivenu vrijednost potrebno je zbiti između vrijednosti 0 i 1, kako bi se izbjegle moguće greške, a zatim se koristi kao faktor zatamnjenja. Ta vrijednost opisuje kojim intenzitetom je potrebno zatamniti fragment.

3.3 Volume sjene

Za ispravan rad tehnika se oslanja na korištenje funkcionalnosti *depth testing*-a i *stencil testing*-a. Za ovu tehniku nije obavezna izrada novog FBO već su sve teksture koje se koriste one zadane. “Algoritam volumnih sjena izrađuje volumen za pronalazak osjenčanih područja iz geometrije zbog čega ne dijeli nedostatke s tehnikom mapa sjena, već postiže točne “tvrde” sjene. Međutim, izračun takvih točnih sjena zahtjeva veću snagu računalnog sklopovlja“ [3].

3.3.1 Crtanje scene i postavke za izradu volumena

U prvom koraku se cijela scena crta na uobičajeni način, iz perspektive oka, a zatim se isključuje pisanje u dubinsku teksturu kako bi njen sadržaj ostao neizmijenjen prilikom novog crtanja. Također se aktivira korištenje stencil teksture te se preko odgovarajućih funkcija postavlja način rada *stencil testiranja*. Način rada na kojem se temelji ova tehnika je: izmjena vrijednosti u stencil teksturi prilikom zadovoljenog *depth fail* uvjeta. Do *Depth fail*-a će doći prilikom pokušaja crtanja fragmenta na mjesto gdje je već upisan fragment s manjom dubinom.

Nakon što je način rada *stencil testiranja* definiran slijedi poziv operaciji crtanja. Međutim, ne crta se cijela scena već se crtaju modeli (volumeni sjena) koji predstavljaju izgled sjene koju baca određeni predmet kao posljedicu utjecaja svjetla.

3.3.2 Izrada volumena

Izgled volumena sjena određuje se u geometry shaderu prilikom crtanja u stencil buffer. Najprije je bitno znati da se u tom, drugom, koraku crta predmet koji će bacati sjenu, a iz tog se predmeta, u geometry shaderu, izradi volumen sjene. Volumen sjene je model koji imitira izgled sjene nastale zbog utjecaja svjetla na predmet u pitanju.

Kako bi se volumen nacrtao ispravno, shader programu mora biti dostupna informacija o položaju izvora svjetla. Rad *geometry shadera* u pitanju može se podijeliti u dvije logičke cjeline.

U prvom koraku shadera otkriva se obris predmeta. Pojam obris predmeta podrazumijeva skup svih rubova predmeta koji spajaju jedan dva poligona od kojih je jedan okrenut u smjeru

svjetla a drugi okrenut u smjeru suprotnom svjetlu. Obris se identificira tako da se redom prolaze sva lica predmeta u pitanju. Prvo se ispituje da li trenutno lice gleda u svjetlo ili ne; ako ne gleda algoritam ga odbacuje. Ukoliko lice gleda u svjetlo prolazi se preko svakog njegovog ruba te se provjerava da li se radi o rubu koji odgovara opisu obrisnog ruba lica.

U drugom koraku se iz dviju točaka koje čine rub određuje četverokut. Četverokute se izrađuje koristeći spomenute točke i dvije točke koje se dobiju produživanjem prvog para točaka u beskonačnost. Ovaj proces izrade četverokuta se ponavlja za svaki identificirani obrisni rub. Nakon što se nacrta novi volumen potrebno je nacrtati dva dodatna četverokuta koja će zatvoriti volumen s ciljem izbjegavanja mogućih grešaka u daljnjem radu programa. Jedan se crta tako da se spoje svi rubovi koji čine obris, a drugi četverokut se izrađuje iz prvog kojeg se translatira u beskonačnost. Pritom je važno da se pri crtanju tih četverokuta odrede odgovarajuće normale. Bliži od dva četverokuta imati će normale usmjerene prema svjetlu, a drugi normale usmjerene od svjetla.

Kako će tehnika volumena sjena mijenjati vrijednosti u stencil teksturi pri depth fail-u moguće je odrediti korištenjem ugrađenih funkcija *glStencilOpSeparate()*. Koristeći tu funkciju određuje se da prilikom crtanja stražnjih lica (eng. *back face*) volumena vrijednost u stencil teksturi inkrementirati, a prilikom crtanja prednjih lica (eng. front face) dekrementirati.

3.3.3 Tri stanja stencil teksture

Dakle, ukoliko se neki hipotetski predmet iz scene nalazi između volumena sjene i oka prilikom crtanja volumena vrijednosti u stencil teksturi će se inkrementirati i dekrementirati te će na kraju crtanja iznositi nulu. Ako je neki drugi predmet postavljen tako da se između njega i oka nalazi volumen sjene vrijednosti u teksturi se neće mijenjati te će ostati na nuli. No, međutim, ako se predmet nalazi unutar volumena vrijednost teksture nakon crtanja biti će inkrementirana jer će do *depth fail*-a doći samo prilikom crtanja stražnjih lica.

3.3.4 Završno crtanje scene

Cilj prethodnog koraka bio je odrediti koji se predmeti iz scene nalaze unutar volumena sjene koristeći stencil teksturu. Nakon uspješnog izmjena vrijednosti u *stencil teksturi* isključuje se daljnje pisanje u nju, kako se vrijednosti ne bi izgubile prilikom novog crtanja, a ponovo se uključuje pisanje u depth teksturu. Prije trećeg koraka bitno je razumijeti da je rezultat prethodnog koraka logička podjela scene na dva dijela, s obzirom na vrijednost u stencil teksturi. Dijelovi scene koji se nalaze u sjeni u stencil teksturi sadrže vrijednost jedan. Ovakav način podjele omogućuje da se kontrolira područje scene u koje će se crtati prilikom izvršavanja slijedećeg koraka.

U zadnjem koraku ove tehnike scena se crta dva puta. Prvi puta se crta dio scene koji je osvijetljen, a u drugi puta dio u sjeni. Određivanje djela scene u koji će se crtati vrši se pozivom "određenim stencil funkcijama". Osjenčani dio scene crta se koristeći samo "ambijentalno" svjetlo kako bi se postigla vidljiva razlika između "dva dijela".

3.3.5 Prednosti i nedostaci tehnike

Tehniku stencil volumen sjena moguće je implementirati na dva načina. Prvi način je način koji je upravo opisan i implementiran je u računalnom programu, a zove se *depth fail*. Drugi način je *depth pass*.

Obje implementacije imaju svoje prednosti i nedostatke

Prednosti korištenja *detph pass* tehnike su što je potrebno crtati manje dodatnih poligona/geometrije, brža je i jednostavnija za implementaciju. Nedostatak ove tehnike je da nije dozvoljen ulazak kamere u volumen sjene. Ukoliko se kamera translatira unutar volumena, doći će do greške zbog koje se sjene neće crtati.

Depth fail pristup zahtjeva korištenje mnogo poligona kako bi se osiguralo da je volumen sjene optimalno zatvoren, zahtjeva crtanje samog volumena do beskonačnosti. Time je sporija i teža za implementirati.

Uzevši u obzir sve prednosti i nedostatke zaključuje se da je *depth pass* tehnika idealna za korištenje u programima koji ne dozvoljavaju ulazak kamere u volumen. U osatlim računalnim programima moguće je koristiti *depth fail* tehniku.

POGLAVLJE 4

4. Implementacija programa

Za izradu računalne aplikacije koja popraćuje ovaj rad korišteno je softversko sučelje Open Graphics Library (OpenGL) te alati GLFW, GLEW, GLM. Uz spomenute alate korištena je funkcija za punjenje shader programa, koja je povučena iz literature [6].

Program je pisan u C++ jeziku u linux operacijskom sustavu. Sadržaj programa je podjeljen u četiri cjeline. Svaka cijelina prikazuje implementaciju jedne od opisanih tehnika. Prva cjelina predstavlja izgled standardne tehnike mape sjena pod utjecajem direkcijskog svjetla koja koristi PCF tehniku filtriranja. Druga cjelina prikazuje implementaciju VSM tehnike pod utjecajem istog direkcijskog svjetla. Treća prikazuje standardnu tehniku mape sjena pod utjecajem pozicijskog svjetla, a tehnika također koristi PCF filtriranje. Četvrta cjelina je posljednja te prikazuje tehniku volumnih sjena, pod utjecajem pozicijskog svjetla.

4.1 Knjižnica GLM

U programu je ekstenzivno korištena knjižnica GLM koja omogućuje pojednostavljeni rad s vektorima i matricama. Knjižnica također opisuje funkcije koje se koriste za izradu željenih transformacijskih matrica. Sadrži funkcije za izradu matrica translacije, rotacije, skaliranja koje programeru aplikacije omogućavaju jednostavnu izradu model transformacijske matrice. Funkcije za izradu spomenutih matrica su redom *glm::translate()*, *glm::rotate()*, *glm::scale()*. Također su podržane funkcije za izradu matrica pogleda i projekcijskih matrica. Funkcija *glm::lookAt()* izrađuje matricu pogleda koristeći informacije koje prima kao tri parametra. Ta tri parametra opisuju položaj kamere, smjer gledanja kamere te vektor *Up* koji predstavlja vrh pogleda kamere. Koristeći *Up* vektor moguće je rotirati kameru. Za izradu projekcijkih matrica koriste se dvije funkcije, ovisno o željenom tipu matrice. Funkcija *glm::ortho()* prima šest parametara koji određuju širinu *frustuma* ortogonalne projekcije. Za izradu perspektivne projekcijske matrice koristi se *glm::perspective()* čiji su parametri redom: širina vidnog polja, omjer sadržane slike te položaj bliže *Z-ravnine* i položaj udaljenije *Z-ravnine*.

4.2 Klase Model i Camera

Klase *model.h* i *camera.h* su izrađene specifično za korištenje u priloženom računalnom programu te su njihove funkcije olakšati organizaciju programa.

Model.h klasa opisuje i drži transformacijske model matrice različitih predmeta koji se mogu crtati u sceni programa. Klasa tako dozvoljava izradu objekta čiji se položaj, rotacija te veličina, korištenjem ugrađenih metoda, kroz tok programa mogu manipulirati. Mogu se koristi metode koje vraćaju model matricu predmeta u obliku tipa podataka *glm::mat4* ili *float*.

Camera.h klasa također sadrži klasu sa pripadajućim varijablama i metodama koje omogućuju postavljanje *view* i *projection* matrica.

4.3 Klase opisanih tehnika

Sve tehnike ostvarivanja sjena opisane u ovom radu implementirane su u priloženi računalni program. Svaka tehnika sastoji se od nekoliko osnovnih koraka, a ovisno o željenom cilju tehnike moguće je na osnovne korake nadodati nove. Kako je i sama implementacija osnovnih koraka dovoljno kompleksna sve su tehnike enkapsulirane u zasebne klase, da bi se postigla veća preglednost pisanog koda.

4.3.1 Zajedničke značajke klasa

Klase su organizirane na način da sadrže sve bitne informacije za ispravan rad. Klasa svake tehnike čuva informacije o vlastitim teksturama i shader programima koje koristi te sadrži funkcije, zvane *handler funkcije,* koje šalju informacije u odgovarajuće shader programe. Objekti svih tehnika inicijaliziraju se eksplicitno, koristeći metodu *Init()* čiji su parametri informacije o veličini prozora. Sve klase sadrže metodu *fillFBO()* čiji se poziv vrši neposredno nakon inicijalizacije objekta, a cilj te metode je izraditi odgovarajuće teksture potrebne za daljnje korištenje tehnika. Ovisno o tehnici, za izradu teksture se koriste različiti, unaprijed određeni, parametri a rezolucija teksture izračunata je na temelju primljenih parametara u funkciju *Init()*. Funkcija *fillFBO()* bool je tipa kako bi se na temelju vraćene vrijednosti moglo utvrditi da je

postignut željeni rezultat. Nakon uspješne inicijalizacije i izrade tekstura, objekt je spreman za korištenje.

U glavnoj *do* petlji programa vrši se crtanje sene s tehnikama izrade sjena. Ovisno o željenoj tehnici scena se crta na drugačiji način kako bi se najbolje prikazale prednosti i nedostaci određenih tehnika. Program korisniku omogućuje da aktivira ponuđene tehnike koristeći tipkovnicu.

4.3.2 Klasa mape sjena

```
class ShadowMap_dir{
private:
	int WINDOW_WIDTH = 0;
	int WINDOW_HEIGHT = 0;
	float SHADOW_COEF = 1.0f;

	GLuint mFBO;
	GLuint mDepthTexture;

	GLuint shader_depthID;
	GLuint shader_lightingID;

	bool loadShaders();
	bool fillFBO();

public:
	bool Init( int shadow_W, int shadow_H,
		int window_W, int window_H );
// postavljanje ovisnosti za prvi korak
	void firstPassSetup();
// postavljanje ovisnosti za drugi korak
	void secondPassSetup();
// handler funkcije za prvi korak
	void setLightMVP( glm::mat4 lMVP);
```

```
// handler funkcije za drugi korak
      void setLightDir( glm::vec3 lightDir);
      void setModel   ( glm::mat4 Model);
      void setView    ( glm::mat4 View );
      void setProj    ( glm::mat4 Proj );
      void setBiasedLightMVP( glm::mat4 biasedlMVP);
      void setIsPCF   ( bool isPCF );
      void setDepthTexture( uint offset = 1);
};
```

Metoda *loadShaders()* poziva se implicitno kroz metodu *Init()*, a postiže punjenje varijabli objekta predviđenih za čuvanje shader programa. Samo učitavanje sadržaja vertex, geometry i fragment shadera te njihovo prevađanje i povezivanje u jedinstveni shader program izvršava se koristeći eksternu *loadShader()* funkciju koja je povezana s klasom u prikazanoj definiciji. *LoadShader()* funkcija preuzeta je iz literature [6] te je izmijenjena kako bi bolje odgovarala za korištenje u ovom programu.

4.3.2.1 postavljačke metode

```
void ShadowMap_dir::firstPassSetup(){
      glBindFramebuffer ( GL_FRAMEBUFFER, mFBO );
            glViewport( 0, 0, SHADOW_WIDTH, SHADOW_HEIGHT );
            glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
            glColorMask( GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE );
            glUseProgram(shader_depthID);

      glCullFace( GL_FRONT );
      glEnable( GL_CULL_FACE );
}
void ShadowMap_dir::secondPassSetup(){
      glCullFace( GL_BACK );
      glBindFramebuffer ( GL_FRAMEBUFFER, 0 );
            glColorMask( GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE );
```

```
        glViewport( 0, 0, WINDOW_WIDTH, WINDOW_HEIGHT );
        glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
        glUseProgram(shader_lightingID);
}
```

Prikazane funkcije *firstPassSetup()* i *secondPassSetup()* izvršavaju aktivaciju odgovarajućih FBO-a te određivanje u koje teksture će se vršiti pisanje. Također se brišu vrijednosti iz tekstura prije njihove uporabe, postavlja se *glViewport()* koji odgovara rezoluciji korištenih tekstura te se aktivira odgovarajući shader program.

4.3.2.2 Sadržaj shadera

Kroz rad su objašnjene razlike u implementaciji tehnike mape sjena koje su posljedica korištenja različitih tipova izvora svjetlosti. Razlike u dvije implementacije javljaju se prilikom pisanja shader programa. Spomenuto je da je prilikom korištenja pozicijskog svjetla potrebno ispuniti mapu kocke te da se s ciljem minimizacije broja potrebnih crtanja koristi funkcionalnost geometrijskog shadera.

Shader program korišten u prvom koraku za direkcijsko svjetlo sastoji se od vertex i fragment shadera. U vertex shaderu se vrši transformacija zaprimljene točke predmeta u perspektivu izvora svjetlosti. Za takvu transformaciju koriste se model teksture, specifične za model, te view i projection teksture koje su izrađene prema zahtjevima rada izvora svjetla. Fragment shader tog programa vrši pisanje vrijednosti dubine u teksturu.

Shader program za isti korak prilikom korištenja pozicijskog svjetla je znatno drugačiji. Vertex shader transformira dobivene točke u koordinatni sustav svijeta, koristeći model matricu, međutim vertex shader ne šalje izračunate transformacije u fragment shader, već u geometrijski shader. Dodani geometrijski shader služi izbjegavanju velikom broju crtanja scene.

```
#version 330 core
    layout (triangles) in;
    layout (triangle_strip, max_vertices=18) out;
uniform mat4 lightVP[6];
out vec4 screen_pos; // FragPos from GS (output per emitvertex)

void main()
```

```
{
  for(int face = 0; face < 6; ++face)
  {
    gl_Layer = face; // Ova varijabla određuje u koje lice se crta.
    for(int i = 0; i < 3; ++i) // za svaku točku u poligonu
    {
      screen_pos = gl_in[i].gl_Position;
      gl_Position = lightVP[face] * screen_pos;
      EmitVertex();
    }
    EndPrimitive();
  }
}
```

Main funkcija geometrijskog shadera izvršava dvije ugnježđene for petlje čiji je cilj nacrtati scenu šest puta, svaki puta koristeći drugi par transformacijskih *view* i *projection* matrica kako bi se ispunila odgovarajuća tekstura. Nakon što se scena transformira koristeći odgovarajuće matrice, njene se vrijednosti šalju u fragment shader. U fragment shaderu se za svaki primljeni fragment izračuna vrijednost udaljenosti između njegovog položaja i položaja svjetla. Izračunatu vrijednost potrebno je podjeliti sa udaljenošću *udaljenije Z ravnine* čime se postiže zbijanje vrijednosti dubine između nule i jedan.

4.3.3 Klasa tehnike VSM

4.3.3.1 Postavke za filtriranje mape sjena

Klasa *varirajućih mapa sjena* je po svojoj definiciji gotovo identična klasi standardne tehnike. Razlike se pronalaze u, već spomenutom, različitom pristupu izrade teksture te u dodatnom koraku filtriranja mape sjena. Iz tog razloga sadrži shader više i odgovarajuće varijable za skladištenje podataka kako bi se uspješno izvodio dodatni korak.

Za izvršavanje procesa filtriranja koriste se dva prolaza, svaki filtrirajući sadržaj teksture po jednoj od dviju osi.

```
void ShadowMap_variance::blurXPassSetup(){
	glDisable( GL_CULL_FACE );
	glBindFramebuffer(GL_FRAMEBUFFER, mFBO_blur);
	glViewport(0, 0, WINDOW_WIDTH * SHADOW_COEF * BLUR_COEF,
WINDOW_HEIGHT * SHADOW_COEF * BLUR_COEF);
	glColorMask(GL_TRUE, GL_TRUE, GL_FALSE, GL_FALSE);
		glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

	glUseProgram(shader_blurID);
}
void ShadowMap_variance::blurYPassSetup(){
	glDisable( GL_CULL_FACE );
	glBindFramebuffer(GL_FRAMEBUFFER, mFBO_depth);
		glColorMask( GL_TRUE, GL_TRUE, GL_FALSE, GL_FALSE );
		glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
	glViewport(0, 0, WINDOW_WIDTH * SHADOW_COEF, WINDOW_HEIGHT *
SHADOW_COEF);

	glUseProgram(shader_blurID);
}
```

4.3.3.2 Implementacija čebiševljeve jednadžbe u fragment shaderu

Nakon izrade mape sjena te njezinog filtriranja izvršava se posljednji korak tehnike *varirajućih mapa sjena* u kojem se cijela scena crta na uobičajen način, pritom provjeravajući jesu li područja koja se crtaju u sjeni. U fragment shaderu se prvo poziva funkcija za izračun osvjetljenja fragmenta, a nakon nje se poziva ključna funkcija za određivanje sjena.

Funkcija je već prikazana u trećem poglavlju ovog rada (3.1).

```
float chebyshevUpperBound( float distance, vec2 myPos)
{
	vec2 moments = texture(shadowMap, myPos).rg;

	if (distance <= moments.x)
		return 1.0f;
	float variance = moments.y - ( moments.x*moments.x );
	variance = min( max( variance, 0.00002 ), 1.0f);

	float d = distance - moments.x;
	float p_max = variance / (variance + d*d);
	p_max = clamp(p_max, 0, 1);

return p_max;
}
```

Vrijednosti koje se šalju kao parametri funkciji su *distance*, dubina fragmenta iz perspektive oka, i vektor *myPos* koji sadrži položaj fragmenta iz perspektive svjetla.

4.3.4 Klasa tehnike volumen sjena

Klasa tehnike volumena sjene ne razlikuje se previše od klase standardne mape sjena. Najbitnija razlika je u tome da se u implementaciji volumena sjene koristi tehnika *deferred*

lighting za osvjetljavanje scene. Zbog korištenja spomenute tehnike klasa sadrži nekoliko dodatnih tekstura koje drže vrijednosti o predmetima u sceni bez utjecaja svjetla.

Pristup izrade tehnike volumena sjene je različit od pristupa implementiranju tehnika mapa sjena, što se tiče pisanja shader programa. U prethodnim su se tehnikama, za crtanje u drugom koraku, koristili shader programi koji utemeljuju transformaciju predmeta u odgovarajući prostor u sceni u *vertex shaderu* i proces osvjetljenja scene, ovisno o korištenim tipovima svjetla, u *fragment shaderu*.

4.3.4.1 Deferred lighting

U nadolazećem dijelu programa će se koristiti tethnika u literaturi zvana deferred lighting. “Cilj deferred lighting tehnike je razdvojiti procese osvjetljenja od procesa transformiranja predmeta u shader programima”[7]. Ova tehnika je obično korisna kada se crtaju scene s velikim broj svjetla jer se u tom slučaju postiže znatna ušteda resura.

Prilikom crtanja scene s više izvora svjetlosti, a da se pritom ne koristi tehnika u pitanju, sve je predmete u sceni potrebno nacrtati toliko puta koliko ima različitih izvora svjetlosti. Dakle, za scenu od deset predmeta i deset izvora svjetlosti crtanje će se izvršiti sto puta.
Ako se ista scena prikaže koristeći tehniku *deferred lighting*-a biti će potrebno jedno crtanje za sve predmete u sceni te još jedno dodatno crtanje za svaki izvor svjetlosti. Dakle, scena će se crtati jedanaest puta.

U tehnici volumena sjene posljedica korištenja tehnike *deferred lighting* neće biti znamenita jer će se koristiti samo jedan izvor svjetlosti. No, svi predmeti u sceni će se crtati samo jednom i njeni rezultati će se spremiti u teksturu. U posljednjem koraku tehnike će se sadržaji tekstura, koji predstavljaju sve predmete u sceni, crtati na ravninu koja prekriva cijeli prozor. Pri crtanja na ravninu koristi se drugi shader program u kojem se primjenjuje utjecaj svjetla. Crtanje na ravninu će se odviti dva puta; prvi puta će se crtati osvjetljeno područje, a drugi puta područja u sjeni.

4.3.4.2 Postavke za crtanje volumena u matričnu teksturu

Kako je opisano u prijašnjem poglavlju, prilikom crtanja volumena sjene ispunjava se matrična tekstura na čijem se ispravnom sadržaju temelji cijela tehnika. Prikazana metoda predstavlja kako se postavlja način rada matrične teksture za postizanje željenog rezultata.

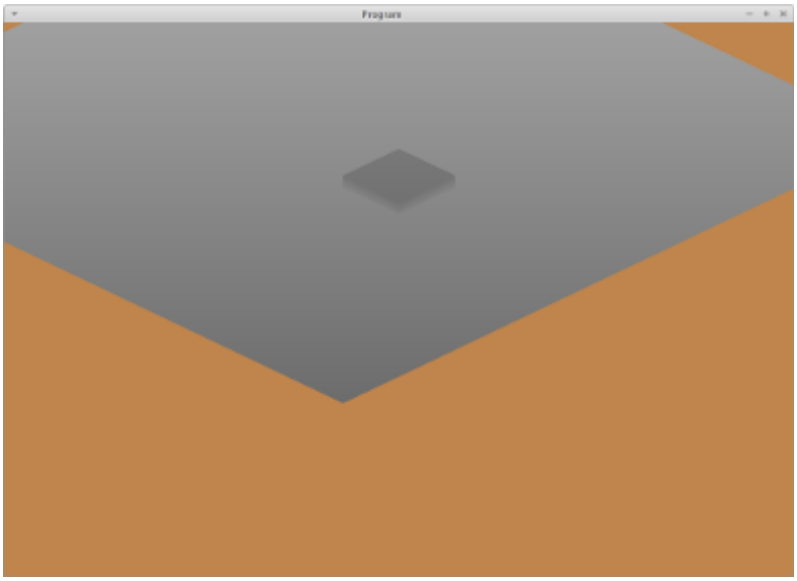
```
void ShadowVolume::volumePassSetup(){
	glDepthMask(GL_FALSE);
	glEnable(GL_DEPTH_CLAMP);
	glColorMask( GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE );
	glDisable(GL_CULL_FACE );

	glUseProgram( shader_volumeID );
	glEnable(GL_STENCIL_TEST);
	glClear ( GL_STENCIL_BUFFER );

	glStencilOpSeparate( GL_BACK, GL_KEEP, GL_INCR_WRAP, GL_KEEP );
	glStencilOpSeparate(GL_FRONT,GL_KEEP,GL_DECR_WRAP, GL_KEEP );
	glStencilFunc( GL_ALWAYS, 0, 0xff );
		glStencilMask( 0xff );
}
```

4.4 Pristup ispravljanju tehnika

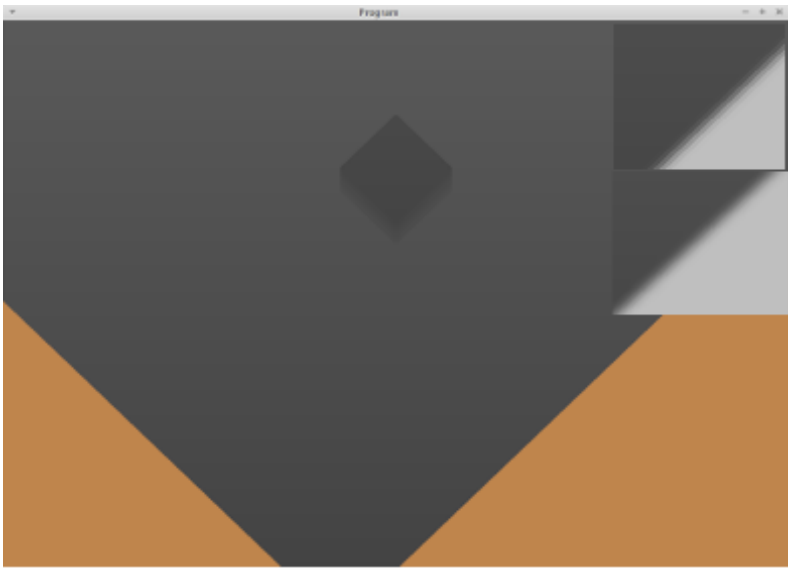
Za implementaciju svake od predloženih tehnika potrebno je prvo dobro razumjeti ideju te tehnike te biti upoznat s funkcionalnostima OpenGL-a nužnim za njihov razvoj.

Prilikom razvoja aplikacije koja implementira tehnike mapa sjena korisno je napisati funkcionalnost koja će crtati sadržaj ispunjene teksture mape sjena u proizvoljan dio prozora. U slučaju da prevodioc (eng. c*ompiler*) ne primjeti grešku prilikom prevođenja i spajanja programa, a sjena u aplikaciji se crta na neispravan način, ili je uopće nema, prikaz sadržaja teksture pomoći će programeru otkriti je li problem nastao pri ispunjavanju mape sjena ili kasnije.

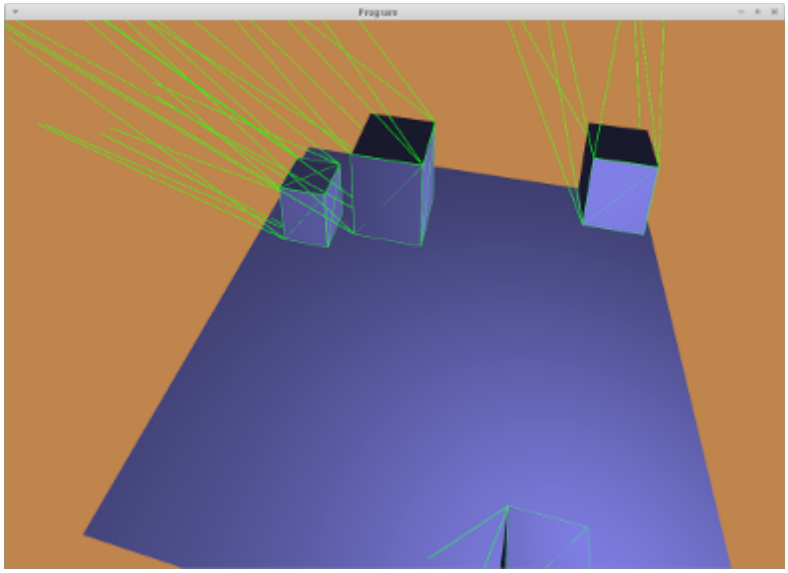
Slična tehnika se koristi u razvoju volumnih sjena. U volumnim sjenama se umjesto mapa sjena u prozor crta sam volumen, kojeg je praktičnije prikazati u *wireframe* izgledu kako bi zakrivao manji dio scene. Na slijedećoj slici, 4.1, prikazani su rezultati korištenja tih funkcionalnosti za svaku tehniku.



a) Standardna mapa sjena



b) Varirajuća mapa sjena



c) Volumne sjene

Slika 4.1. Pristup ispravljanju tehnika. Slika prikazuje rezultate prikazivanja mapa sjena u (a) i (b) primjeru. U gornjem desnom kuta primjera (b) prikazana su oba rezultata procesa filtriranja mape sjena. Zadnji, (c) dio, slike prikazuje scenu s vidljivim volumenima sjena koji sežu u beskonačnost.

4.5 Analiza tehnika

U ovome potpoglavlju biti će rezimirane najbitnije prednosti i nedostaci opisanih tehnika te će se usporediti brzina njihovih izvršavanja.

"Tehnika volumena sjene ima ozbiljne nedostatke: iznimno je zahtjevna za procesor (eng. *central processing unit, CPU*), može crtati samo "tvrde" sjene, koristi velik dio vremena punjenja (eng. *fill rate*) što znači da će program sporije raditi na slabijim grafičkim karticama" [2]. Uz sve navedeno, izrada volumena sjena ovisi o kompleksnosti predmeta u sjeni, crta dodatnu geometriju za svaki predmet koji baca sjenu te od svakog takvog predmeta zahtjeva da sadrži listu rubova svojih poligona. O listi rubova ovisi funkcija za izračun obrisa rubova, i na posljetku, samog volumena. Međutim, velika prednost tehnike je da njena najjednostavnija implementacija stvara sjene koje nisu nazubljene, što se je nemoguće postići s mapama sjena.

Prednost mapa sjena je da je neovisna o složenosti modela čiju sjenu crta. Nedostaci tehnike su: gotovo je nemoguće izraditi sjene visoke kvalitete zbog ograničene veličine teksture, samo-sjenčanje (eng. *self shadowing*) pri kojem predmeti bacacju sjene na područja koja bi

trebala biti osvjetljena. Najjednostavnija implementacija sjena podliježe mnogim greškama, te se gotovo uvijek uz nju koriste tehnike filtriranja.

Varirajuće mape sjena postižu veliki napredak u odnosu na standardnu tehniku, te uspijevaju postići dobar rezultat, realnog izgleda sjena, s manje resursa.

4.5.1 Brzina izvođenja

U programu je mjereno vrijeme crtanja scene za svaku implementaciju tehnike. Kao uobičajena metoda mjerenja koristi se izračun vrijednosti *frames per second* (FPS), no FPS nije dovoljno vjerodostojan način za izračun performansi grafičke aplikacije zato što njegova vrijednost nije linearno ovisna o vremenu [8]. Bolji pristup računanju brzine izvođenja programa je koristeći *frame time*. Vrijednost *frame time* računa se po slijedećoj formuli:

$$frame\,time = \frac{1}{FPS} \tag{4.1}$$

Tehnike su mjerene u uvjetima sličnim onim priloženog programa, u sceni s jednakim brojem predmeta. Rezultati tehnika bili su očekivani, a prikazani su u tablici:

Tablica 4.1 Vrijeme crtanja scene tehnika

Tehnika	Vrijeme u ms
SM	16.6667
SM + PCF	16.9492
VSM	16.6667
SV	20.0000

Iz rezultata se može zaključiti kako je tehnika VSM najbrža u ovisnosti o kvaliteti sjene koju postiže. Izgled sjene VSM približno je jednak sjeni koja se dobije koristeći PCF filtriranje pri standardnim mapama sjena, a crta se brže. Tehnika SV zahtjeva više vremena za crtanje, no izgled sjena koji se postiže ne može se zanemariti. Također, tehnike su testirane koristeći jednostavne modele, kocke, tako da se može predpostaviti kako bi za kompleksnije modele u sceni tehnici SV trebalo više vremena za izvršavanje.

POGLAVLJE 5

5. Zaključak

Razvojem industrije računalnih igrara, animacija i filmova porastao je zahtjev za, ne samo funkcionalnim, već čim realnijim sjenama. Ovaj rad opisuje prednosti i nedostatke između dviju najpopularnijih tipova sjena, mapa sjena i volumnih sjena. Obje tehnike imaju svoje prednosti i nedostatke te ih je moguće nadograđivati kako bi se postigli bolji rezultati. Međutim ideja na kojoj se tehnike temelje predstavlja konačno ograničenje koje se još uvijek nije uspjelo unaprijediti. Kako su obje tehnike aktivne u industriji dugi niz godina, izradila su se brojna rješenja koja minimiziraju nedostatke tehnika u određenim situacijama. Može se sa sigurnošću reći da se postupci izrada sjena s napretkom računalnog sklopovlja, najbitnije, grafičkih kartica, neće prestati razvijati te će se s vremenom pojavljivati bolji načini implementacije.

Opisuju se implementacije tehnika i neki od mogućih problema do kojih dolazi te kako ih rješiti. Također se vršila analiza brzine izvođenja obrađenih tehnika. Iz rezultata analize tehnika varirajućih mapa sjena identificirala se kao bolji izbor od standardne tehnike, na temelju brzine njenog izvršavanja i kvalitete sjena koju ostvaruje. No, sve dok se ne izradi univerzalan pristup za rješavanja problema izrade kvalitetnih, realnih sjena, nijedna od dviju osnovnih (mape sjena, volumne sjene) tehnika neće ispasti iz uporabe.

Literatura

[1] Elmar Eisemann, Michael Schwarz, Ulf Assarsson, Michael Wimmer: "Real-Time Shadows", A K Peters, 2011.

[2] Randima Fernando: "GPU Gems", Addison-Wesley, 2004.

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

[4] Huber Nguyen: "GPU Gems 3", Addison-Wesley, 2007.

[5] LambdaCube3d: "Variance Shadow Mapping", s Interneta, **https://lambdacube3d.wordpress.com/2012/10/14/variance-shadow-mapping/**, 14. listopada 2012.

[6] **http://www.opengl-tutorial.org/**, s Interneta, 2012.

[7] Randi J. Rost, Bill Licea-Kane, "OpenGL Shading Language Third Edition", Addison-Wesley, 2009.

[8] "Performance", **https://www.khronos.org/opengl/wiki/Performance**, s Interneta, 13. travnja 2015.

Popis kratica

FBO	- Framebuffer
PCF	- Percentage-Closer Filtering
SM	- Shadow Map
VSM	- Variance Shadow Map
SV	- Shadow Volume
lightV	- view matrica izvora svjetlosti
lightP	- projection matrica izvora svjetlosti
eyeV	- view matrica oka
eyeP	- projection matrica oka
OpenGL	- Open Graphics Library
GLUT	- OpenGL Extension Wrangler Library
GLFW	- OpenGL Framework
GLM	- OpenGL Mathematics

Sažetak

Postupci izrade sjena predstavljaju skup tehnika, iz područja računalne grafike, koje omogućuju crtanje sjena u računalnom programu koristeći za njihovu izradu svoj specifičan pristup. U ovom radu opisane su tehnike mape sjena, varirajuće mape sjena te tehnika volumnih sjena. Objašnjen je njihov pristup rješavanju problema izrade sjene, funkcionalnosti OpenGL-a na kojoj se temelje te prednosti i nedostaci njihovog korištenja. Također je obavljena analiza brzine tehnika te primjer njihove implementacije u priloženom programu.
Na čitatelju je da odluči, na temelju usporedbe predstavljene u ovom radu, koje će metode koristiti u vlastitoj implementaciji.

Ključne riječi – sjena, mapa sjena, volumen sjena, varirajuća mapa sjena

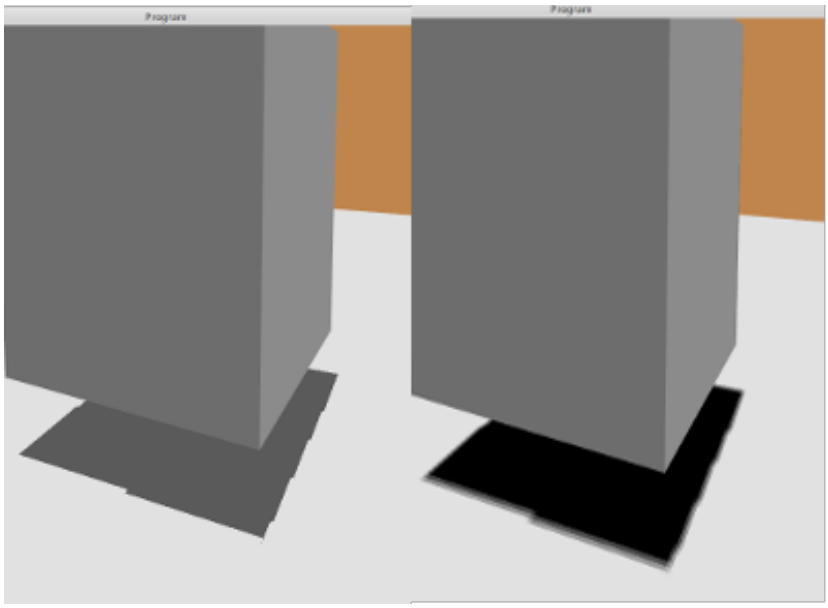
Abstract

Shadow creation methods encompass all techniques, from the domain of computer graphics, that allow drawing shadows in computer applications by using a specific approach for their creation. This thesis describes some of those techniques: shadow maps, variance shadow maps and volume shadows. Each technique's approach to solving the problem is explained, as are all of the OpenGL functionalities they depend on. Pros and cons of all techniques are also pointed out. An analysis is made comparing the performance of techniques and an example program, implementing all of the said techniques, is included.
It is up to the reader to decide, based on the comparisons made in this thesis, to decide which methods are the most suitable for his / her implementation.

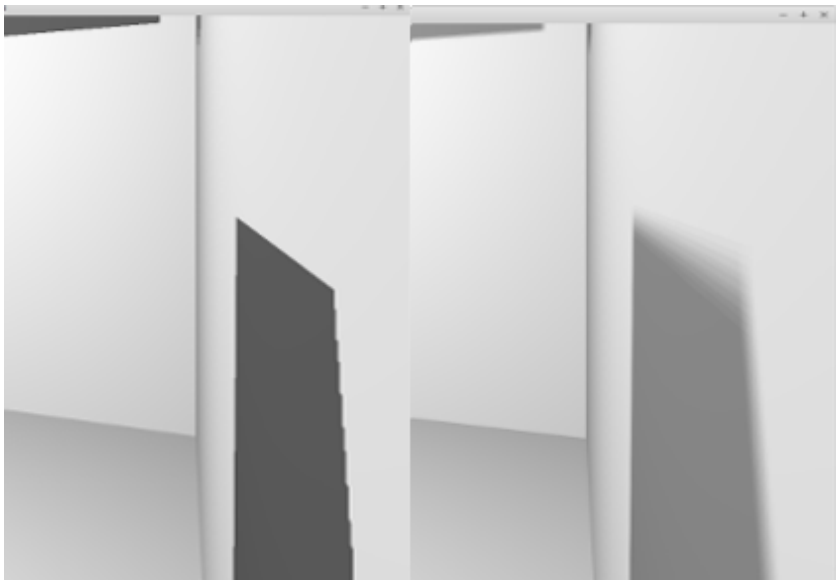
Key words – shadow, shadow map, variance shadow map, shadow volume

Dodatak A

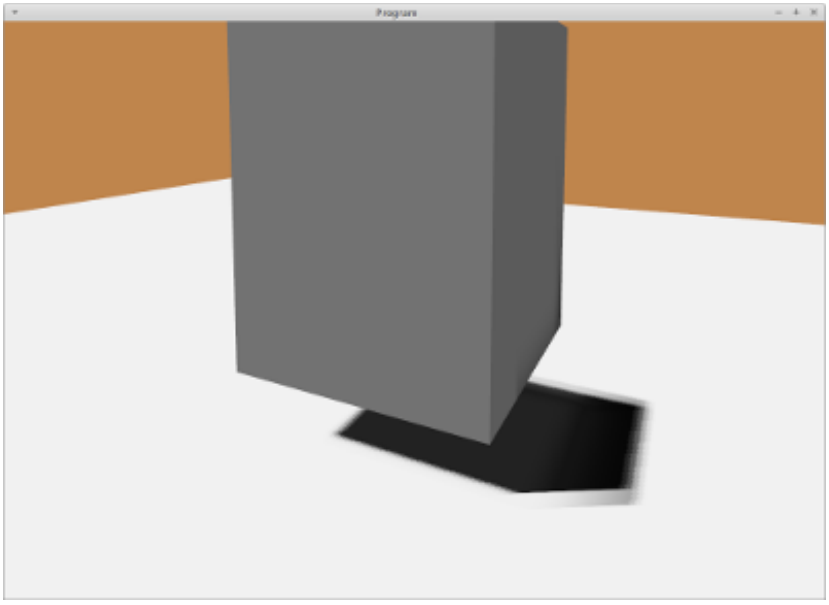
Izgled priloženog programa



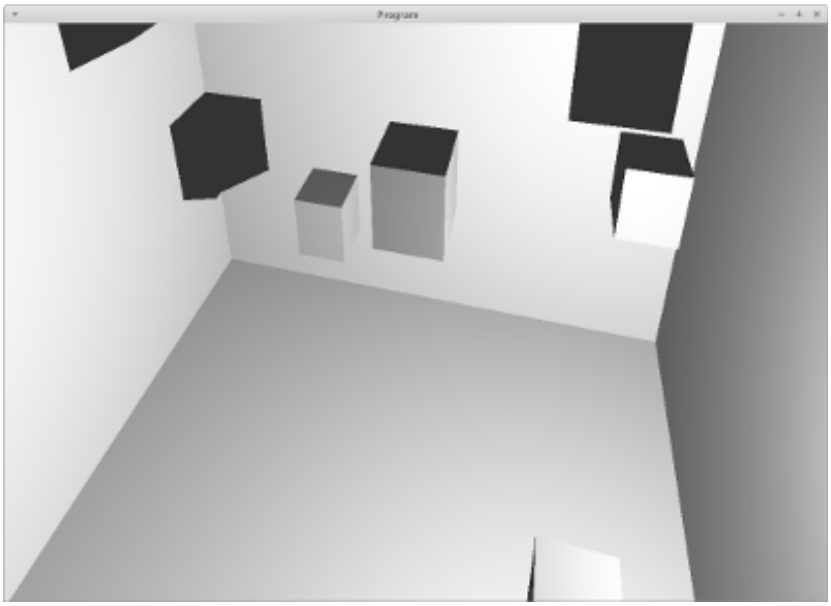
Slika A.1 Standardne mape sjena i mape sjena s PCF-om



Slika A.2 Standardne pozicijske mape sjena i pozicijske mape sjena s PCF-om



Slika A.3 Varirajuće mape sjena



Slika A.4 Volume sjene