SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5001

Preslikavanje sjene

Anteo Ivankov

Zagreb, lipanj 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 5. ožujka 2017.

ZAVRŠNI ZADATAK br. 5001

Pristupnik: **Anteo Ivankov (0036485518)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Preslikavanje sjene**

Opis zadatka:

Proučiti postupke za ostvarivanje sjene u virtualnoj sceni. Razraditi nekoliko algoritama kojima se ostvaruje učinak sjene. Realizirati programsku implementaciju ostvarivanja sjene te načiniti usporedbu implementiranih postupaka. Načiniti ocjenu rezultata i implementiranih algoritama.
Izraditi odgovarajući programski proizvod. Po potrebi koristiti grafičke programske alate. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 9. lipnja 2017.

Mentor:
Prof. dr. sc. Željka Mihajlović

Djelovođa:
Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Siniša Srbljić

SADRŽAJ

1. Uvod

Jedan od glavnih ciljeva računalne grafike je stvoriti realističan i detaljan prikaz objekata koji se nalaze u virtualnoj sceni. Postoji mnogo tehnika koje se koriste za generiranje virtualne scene od kojih svaka ima svoje prednosti i nedostatke. Najosnovniju podjelu možemo napraviti na postupke koji se koriste za iscrtavanje u stvarnom vremenu (engl. Real time rendering) i postupke za iscrtavanje kod kojih brzina iscrtavanja nije toliko važna već je važnije dobiti što je moguće kvalitetniju sliku (engl. Offline rendering).

Neovisno o načinu iscrtavanja, jedan od važnijih učinaka u virtualnoj sceni je prikaz sjena. Sjene su bitne jer izuzetno doprinose ugođaju scene i osjećaju za položaj predmeta u 3D prostoru. Sjene su u osnovi jednostavan učinak. Nastaju tako što neki objekt zaklanja izvor svjetla pa svjetlost ne dopire do površine zaklonjenog objekta. Unatoč tome što su sjene relativno jednostavan učinak, njihov izračun je u općenitom slučaju skupa operacija. Izračun utjecaja sjene usko je vezan s izvorima svjetla u sceni jer njihov položaj, orijentacija i vrsta određuju koji dijelovi scene su u sjeni. Također, ukoliko postoji više izvora svjetlosti potrebno je za svakog od njih utvrditi kakav je njihov utjecaj u sceni. Zbog toga čitav postupak izračuna sjene nije nimalo jednostavan te postoji mnoštvo algoritama kojima se pokušava aproksimirati njihov utjecaj.
Cilj ovog rada je upoznavanje s osnovnim modelom osvjetljenja te osnovnim metodama za iscrtavanje sjene.

2. Korištene tehnologije

2.1 Programsko sučelje OpenGL

Za OpenGL možemo reći da je višeplatformska specifikacija s podrškom za niz programskih jezika, a čiji je cilj omogućiti pisanje aplikacija koje rade s 2D i 3D grafikom [1]. OpenGL definira niz primitiva koji se koriste za izgradnju složenih scena kao što su točke, linije i poligoni. Također, OpenGL nudi mogućnost primjene transformacija nad objektima u sceni, odbacivanje dijelova objekata koji su promatraču nevidljivi, primjenu tekstura i još niz drugih mogućnosti. OpenGL je interno dizajniran kao stroj stanja kojim se upravlja pozivima OpenGL funkcija. U novijim inačicama OpenGL-a omogućeno je i pisanje programa za sjenčanje(engl. Shaders), to jest određeni dijelovi grafičkog protočnog sustava postali su programabilni. To je donijelo veće mogućnosti za programere te omogućilo izradu složenijih programa.

2.2 Programski jezik C++

C++ je programski jezik nastao kao proširenje programskog jezika C. To je jezik opće namjene i srednje razine, s podrškom za objektno orijentirano programiranje. Neki od važnijih koncepata koje podražava su polimorfizam, virtualne funkcije i predlošci (engl. Templates). Neizostavan dio jezika C++ je i standardna biblioteka STL (engl. Standard Template Library) koja pruža programeru mnoge korisne klase napravljene u obliku predložaka.

2.3 OpenGL Shading Language (GLSL)

GLSL je jezik koji OpenGL koristi za pisanje programa za sjenčanje. Sintaksa jezika bazirana je na programskom jeziku C. Napravljen je kako bi se programerima dala mogućnost veće kontrole nad grafičkim protočnim sustavom. Programi napisani u GLSL jeziku nisu neovisne aplikacije već ovise o aplikaciji koja koristi OpenGL.

3. Model osvjetljenja

S obzirom na to da je u stvarnosti interakcija svjetlosti s predmetima u prostoru vrlo složena, u računalnoj grafici postoje različiti postupci kojima se pokušava aproksimirati takva interakcija.

3.1 Phongov model osvjetljenja

U ovom poglavlju spomenut ćemo kako se u virtualnoj sceni aproksimiraju utjecaji globalnog i lokalnog osvjetljenja na temelju Phongovog modela refleksije svjetlosti.
Phongov model refleksije svjetlosti je jedan od najpoznatijih i najčešće korištenih modela odbijanja svjetlosti za grafiku u stvarnom vremenu. Dobio je ime po kineskom znanstveniku Bui-Tuong Phongu, koji je ovaj model postavio 1975.godine. Ovaj model osvjetljenja je jednostavan za računanje, a daje vrlo dobru aproksimaciju. Phongov model odbijanja svjetlosti sastoji se od tri komponente: ambijentne, difuzne i zrcalne.

3.1.1 Ambijentna komponenta

Ambijentna komponenta na jednostavan način simulira utjecaj globalnog osvjetljenja, dakle učinak neizravnog svjetla. Ovo se svjetlo po definiciji vidi svugdje u sceni. Koristi se kako dijelovi scene na koje svjetlost izravno ne pada ne bi bili potpuno crni. Najčešće se koristi u vrlo malim količinama jer daje jednoličnu boju čime se gubi privid trodimenzionalnosti. Formula (1) prikazuje izračun ambijentne komponente gdje je I_a trodimenzionalni vektor koji opisuje intenzitet ambijentne komponente svjetla i on je konstantan za cijelu scenu, a k_a trodimenzionalni vektor koji opisuje ambijentni koeficijent materijala

$$ambijentnaKomponenta = I_a * k_a \quad (1)$$

3.1.2 Difuzna komponenta

Difuznom komponentom simulira se Lambertov zakon koji opisuje odnos osvjetljenosti neke površine u ovisnosti o upadnom kutu svjetla. Intenzitet svjetla je najveći kada zraka upada okomito na površinu. Formula (2) prikazuje izračun difuzne komponente gdje je I_i trodimenzionalni vektor koji opisuje intenzitet izvora svjetlosti, k_d je trodimenzionalni vektor koji opisuje difuzni koeficijent materijala, $\vec{L}$ je jedinični vektor koji ima smjer od promatrane točke prema izvoru svjetla dok je $\vec{N}$ jedinični vektor normale u toj točki.

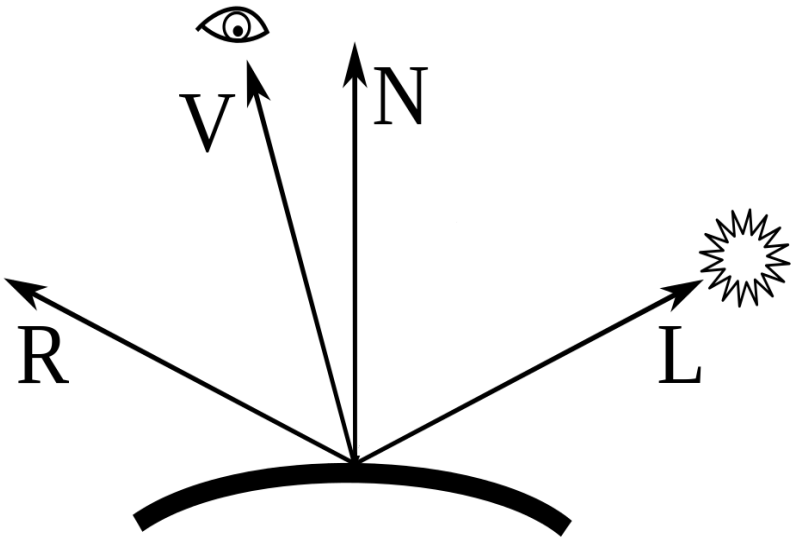
$$difuznaKomponenta = I_i * k_d * (\vec{L} \cdot \vec{N}) \qquad (2)$$

3.1.3 Zrcalna komponenta

Zrcalna komponenta aproksimira zrcalni odsjaj na predmetu. Zrcalna komponenta ovisi o kutu između reflektirane zrake i položaja kamere. Što je kut između kamere i reflektirane zrake manji to je zrcalni odsjaj veći te se naglo smanjuje povećanjem tog kuta. Formula (3) prikazuje izračun zrcalne komponente gdje je I_i trodimenzionalni vektor koji opisuje intenzitet izvora svjetlosti, k_s je trodimenzionalni vektor koji opisuje zrcalni koeficijent materijala, $\vec{R}$ je jedinični vektor smjera koji se dobije kao reflektirani vektor vektora $\vec{L}$ s obzirom na $\vec{N}$, $\vec{V}$ je jedinični vektor od točke prema očištu, dok n predstavlja faktor koji opisuje sjajnost materijala. Što je n veći to je materijal sjajniji.

$$zrcalnaKomponenta = I_i * k_s * (\vec{R} \cdot \vec{V})^n \qquad (3)$$

Slika 1 prikazuje sve vektore koji se koriste pri izračunu Phongovog modela osvjetljenja.



Slika 1. Phongov model refleksije svjetlosti

3.1.4 Ukupno osvjetljenje

Nakon izračuna ambijentne, difuzne i zrcalne komponente, ukupnu boju u točki računamo formulom (4). U formuli (4) kao rezultat dobivamo trodimenzionalni vektor čije komponente predstavljaju u kojoj mjeri su zastupljene crvena, zelena i plava komponenta boje.

$$boja = ambijentnaKomponenta + difuznaKomponenta + zrcalnaKomponenta \quad (4)$$

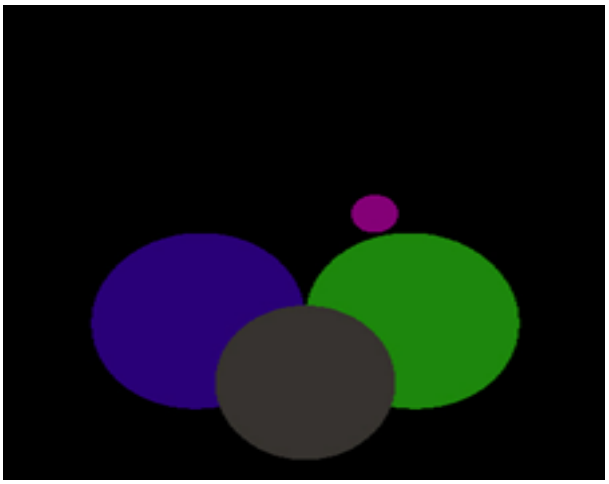
3.1.5 Implementacija Phongovog sjenčanja

Phongovo sjenčanje računa osvjetljenje u svakoj točki virtualne scene pa ga iz tog razloga implementiramo u sjenčaru fragmenata (engl. Fragment shader). Isječak koda 1 prikazuje implementaciju Phongovog sjenčanja u jeziku za sjenčanje GLSL.

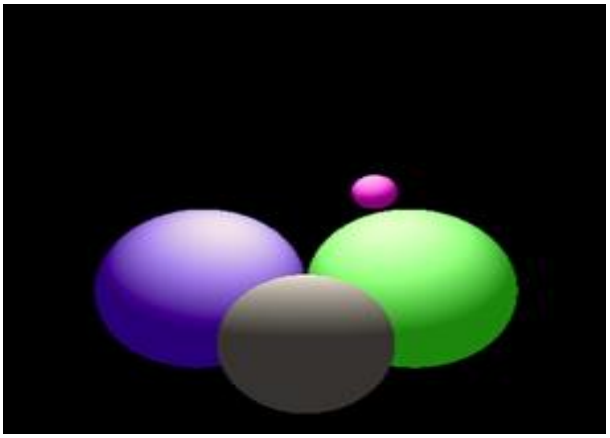
```
#version 330 core
struct Light
{
    vec3 direction;
    vec3 ambientIntensity;
    vec3 color;
};

struct Material
{
    vec3 ka;
    vec3 kd;
    vec3 ks;
    float n;
};
in vec3 normal;
in vec3 fragmentPosition;
uniform Light light;
uniform Material material;
uniform vec3 eyePosition;
out vec4 FragColor;

void main()
{
    // Calculate ambient component.
    vec3 ambientComponent = light.ambientIntensity * material.ka;

    // Calculate diffuse component.
    vec3 L = normalize(-light.direction);
    vec3 N = normalize(normal);
    vec3 diffuseComponent = vec3(0.0, 0.0, 0.0);
    float diffIntensity = dot(L,N);
    if(diffIntensity > 0.0)
    {
        diffuseComponent = (light.color * material.kd) * diffIntensity;
    }

    // Calculate specular component.
    vec3 V = normalize(eyePosition - fragmentPosition);
    vec3 R = normalize(reflect(light.direction, N));
    float specularIntensity = pow(dot(V, R), material.n);
    vec3 specularComponent = vec3(0.0, 0.0, 0.0);
    if(specularIntensity > 0.0)
    {
        specularComponent = (light.color * material.ks) * specularIntensity;
    }

    vec3 color = ambientComponent + diffuseComponent + specularComponent;
    FragColor = vec4(color, 1.0);
}
```

Isječak koda 1. Implementacija Phongovog sjenčanja

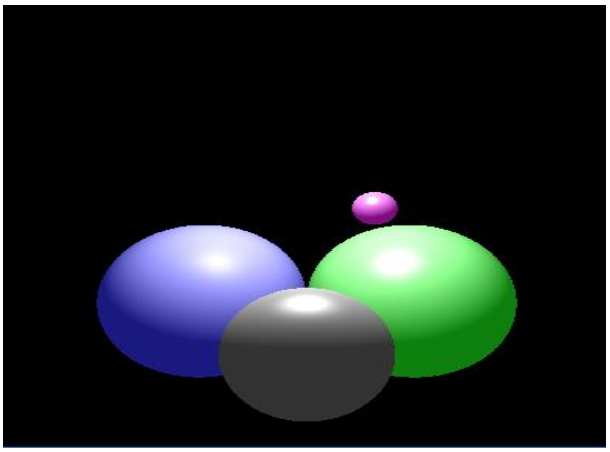
Slika 2, Slika 3 i Slika 4 prikazuju kako različite komponente Phongovog osvjetljenja utječu na osvjetljenje scene.



Slika 2. Ambijentna komponenta Phongovog osvjetljenja



Slika 3. Ambijentna i difuzna komponenta Phongovog osvjetljenja



Slika 4. Ambijentna, difuzna i zrcalna komponenta Phongovog osvjetljenja

4. Modeli izvora svjetlosti

Izvori svjetlosti aproksimirani su s nekoliko modela. Neki od osnovnih modela su: Usmjereno svjetlo i točkasti izvor svjetlosti.

4.1 Usmjereno svjetlo

Usmjereno svjetlo definira se smjerom i intenzitetom. Iz toga proizlazi da su sve zrake koje dolaze iz usmjerenog svjetla međusobno paralelne. Dobar primjer usmjerenog svjetla je sunce koje zbog svoje udaljenosti možemo aproksimirati usmjerenim svjetlom. Također, za usmjereno svjetlo je karakteristično to što intenzitet ne ovisi o udaljenosti objekta do kojeg dolaze zrake kao što je to u slučaju s točkastim izvorom svjetlosti.

4.2 Točkasti izvor svjetlosti

Točkasti izvor svjetlosti definira se intenzitetom koji ovisi o udaljenosti svjetla i objekta te pozicijom. Kod realnih svjetala, intenzitet svjetla obrnuto je proporcionalan kvadratu udaljenosti. Ovisnost intenziteta o udaljenosti prikazana je formulom (5). U formuli (5) L_1 je intenzitet svjetla na udaljenosti 1, a D predstavlja udaljenost izvora do promatrane točke.

$$L = \frac{L_1}{D^2} \tag{5}$$

Međutim, formula (5) ne daje dobre rezultate u računalnoj grafici. Primjerice, kako udaljenost postaje sve manja i manja intenzitet se povećava u beskonačnost. Osim toga, formula ima samo jedan parametar koji se može postaviti što je dosta ograničavajuće. Zbog toga se u praksi koristi formula (6) koja daje veću slobodu pri opisivanju ovisnosti intenziteta o udaljenosti. U formuli (6) A, B, i C su početni parametri koji opisuju kako se mijenja intenzitet s udaljenošću.

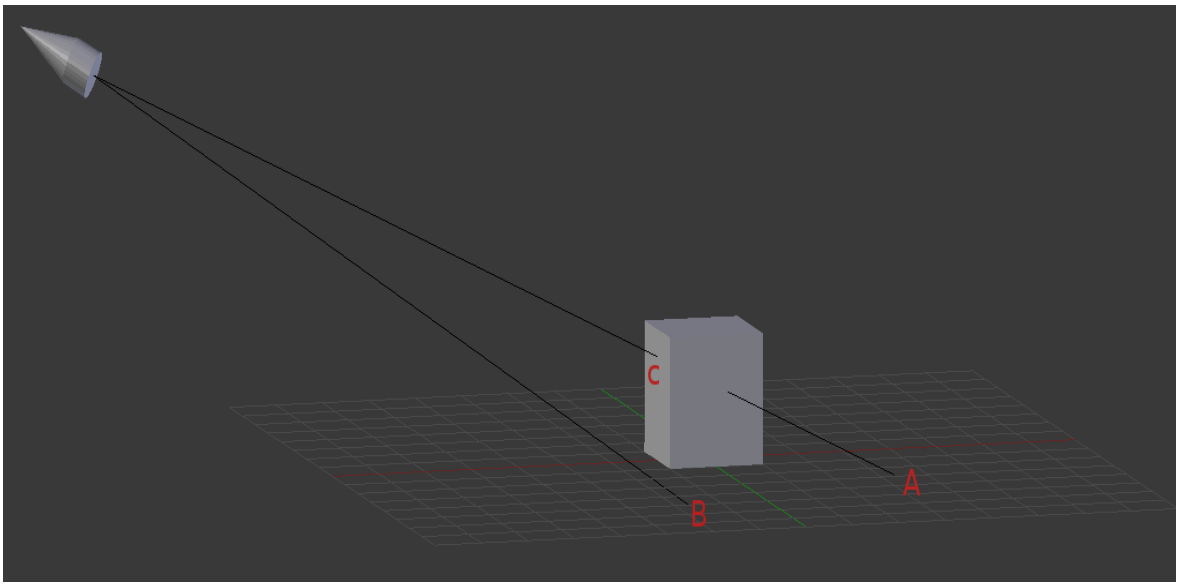
$$L = \frac{L_1}{A + B * D + C * D^2} \tag{6}$$

5. Metoda teksture sjena

Metoda teksture sjena jedna je od najčešće korištenih metoda za iscrtavanje sjena u stvarnom vremenu. Ideja ove metode je da sve točke scene koje su vidljive iz perspektive svjetla budu osvjetljene, dok su ostale točke u sjeni. Postoji više različitih izvedbi ovog algoritma, ali se sve temelje na istom osnovnom algoritmu.

Ideja je da se scena prvo iscrta iz perspektive izvora svjetlosti u teksturu koja zapravo ima ulogu Z-spremnika. Dakle, za svaku točku u sceni računa se udaljenost te točke do izvora svjetlosti. U teksturu se potom upisuje izračunata udaljenost na mjesto slikovnog elementa u koji se ta točka preslikava. Takva tekstura naziva se tekstura sjena. Ukoliko se više točaka preslikava u isti slikovni element, tada se provjerava je li udaljenost promatrane točke do izvora svjetlosti manja ili veća od trenutne upisane udaljenosti u teksturi. Ako je udaljenost manja, upisuje se nova vrijednost udaljenosti za taj slikovni element, a u protivnom se ništa ne mijenja.

Primjerice, Slika 5 prikazuje da se točke C i A preslikavaju u isti slikovni element. S obzirom da je očito točka C osvjetljena, a točka A u sjeni, rezultat je takav da će neovisno o tome koja od točaka dođe prije na obradu, na kraju udaljenost do točke C biti zapisana u teksturi jer je ona najbliža izvoru svjetlosti za taj slikovni element.



Slika 5. Primjer određivanja dubine u teksturi sjene

Nakon što smo scenu iscrtali u teksturu, sljedeći korak je iscrtavanje scene iz perspektive kamere. U ovom koraku za svaku točku uzorkujemo teksturu sjena i provjeravamo je li udaljenost točke do svjetla veća od one zapisane u teksturi. Ako je udaljenost veća to znači da se točka koju iscrtavamo nalazi u sjeni te na nju primjenjujemo samo ambijentalno osvjetljenje, a u protivnom se primjenjuje i difuzno osvjetljenje. Kako bi pravilno usporedili dubine, moramo svaku točku transformirati u koordinatni sustav svjetla jer smo u tom koordinatnom sustavu napravili teksturu u prvom koraku.

Isječak koda 2 prikazuje dio koda iz sjenčara fragmenta kojim se postiže provjera je li neka točka u sjeni.

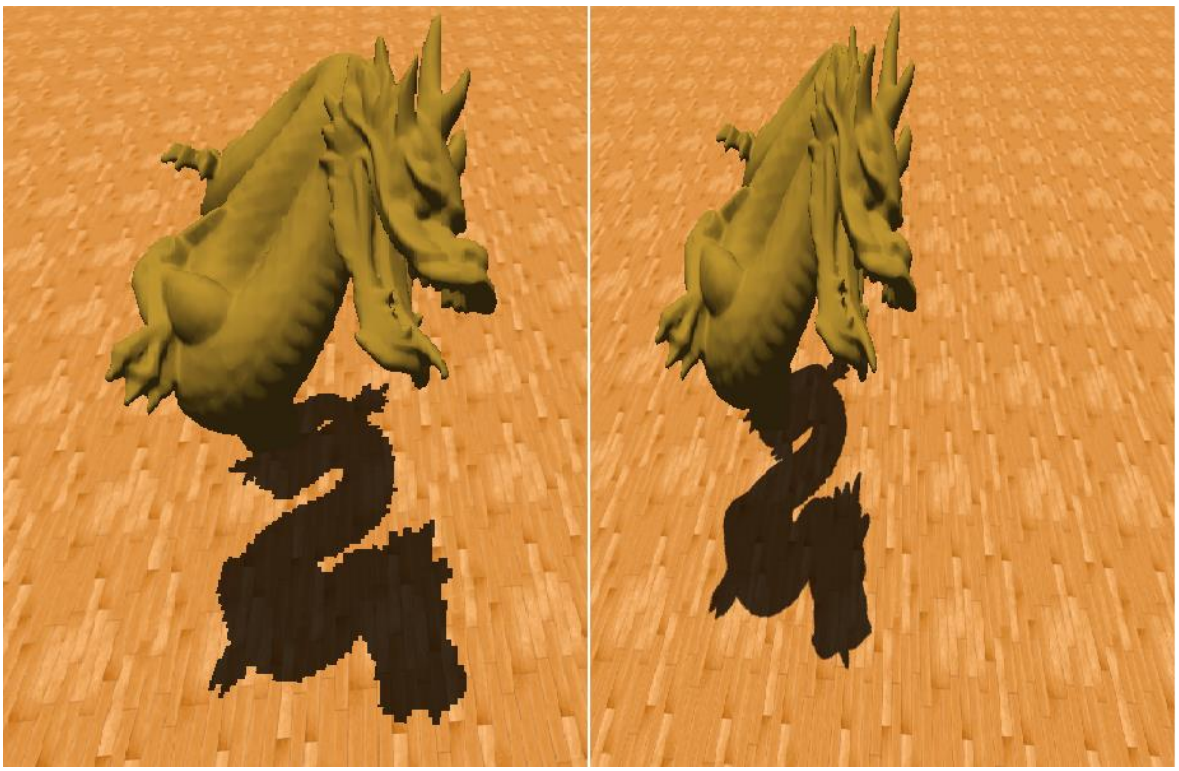
```
// Checks whether fragment is in shadow or not
// @return 1.0 if fragment is in shadow, 0.0 if not
// @param LightSpaceFragmentPos fragment position in projected light space coordinates
float CalculateShadow(vec4 LightSpaceFragmentPos)
{
    // Perform perspective divide (coordinates are now in range [-1.0, 1.0]).
    vec3 ProjectedCoords = LightSpaceFragmentPos.xyz / LightSpaceFragmentPos.w;

    // transform coordinates from range [-1.0, 1.0] to [0, 1] because
    // texture coordinates are specified in that range.
    ProjectedCoords = ProjCoords * vec3(0.5) + vec3(0.5);

    // Sample shadow map.
    float ShadowMapDepth = texture(ShadowMapSampler, ProjCoords.xy).x;

    // Current fragment depth.
    float CurrentDepth = ProjCoords.z;

    float Shadow;
    // if depth of current fragment is larger than depth written in
    // shadow map, then point is in shadow.
    if(CurrentDepth > ShadowMapDepth)
    {
        Shadow = 1.0;
    }
    else
    {
        Shadow = 0.0;
    }
    return Shadow;
}
```

Isječak koda 2. Funkcija koja provjerava je li promatrana točka u sjeni

5.1 Problemi metode teksture sjena

5.1.1 Neželjeni učinci kod sjene

U općenitom slučaju teksturni elementi nisu iste veličine kao slikovni elementi u konačnoj slici. Iz tog razloga dolazi do pojave nazubljenosti rubova sjena. Postoje zapravo dva oblika neželjenih učinaka kod sjene koje nazivamo alias. Prvi je perspektivni alias koji se javlja kad se kamera nalazi bliže sjenčanoj površini nego izvor svjetlosti, a drugi je projekcijski alias, do kojeg dolazi kad je promatrana površina pod oštrim kutom u odnosu na izvor svjetlosti [2]. Slika 6 prikazuje usporedbu nazubljenih i nenazubljenih rubova sjena. Na slici 6 (lijevo) nazubljenost je posljedica manje veličine teksture sjena. Konkretno, veličina teksture na slici 6 (lijevo) je 300x300 slikovnih elemenata dok je veličina teksture na slici 6 (desno) 1000x1000 slikovnih elemenata.



Slika 6. Usporedba nazubljenih i nenazubljenih rubova sjena

Najjednostavnije rješenje uklanjanja neželjenih učinaka kod sjena je povećati razlučivost teksture sjena, no to će rezultirati sporijim iscrtavanjem i većom potrošnjom memorije.

Naprednija metoda je korištenje kaskadnih tekstura sjena (engl. Cascaded shadow maps) kod koje se koristi više tekstura različitih razlučivosti. Ideja je da se prostor

pogleda podjeli na regije pri čemu je svakoj regiji pridružen određeni interval dubine. Za regije bliže kameri generira se tekstura sjene visoke razlučivosti te se onda prilikom iscrtavanja, na temelju dubine određuje koju teksturu treba uzorkovati.

Još jedna metoda s kojom se uklanjaju neželjeni učinci kod sjene je posto-bliže filtriranje (engl. percentage-closer filtering PCF). Metoda je konceptualno slična bilinearnom filtriranju a radi na način da omekšava rubove sjena, čime se neželjeni učinci kod sjena ublažavaju. Ideja algoritma je za svaku točku uzeti N susjednih uzoraka dubine iz teksture sjena (primjerice, 2x2 ili 4x4), te usporediti njihove dubine s dubinom točke. Uz pretpostavku da je za M uzoraka ustanovljeno da je njihova dubina manja od dubine promatrane točke, onda je postotak točke P koji nije u sjeni dan formulom (7).

$$P = 1 - \frac{M}{N} * 100\% \qquad (7)$$

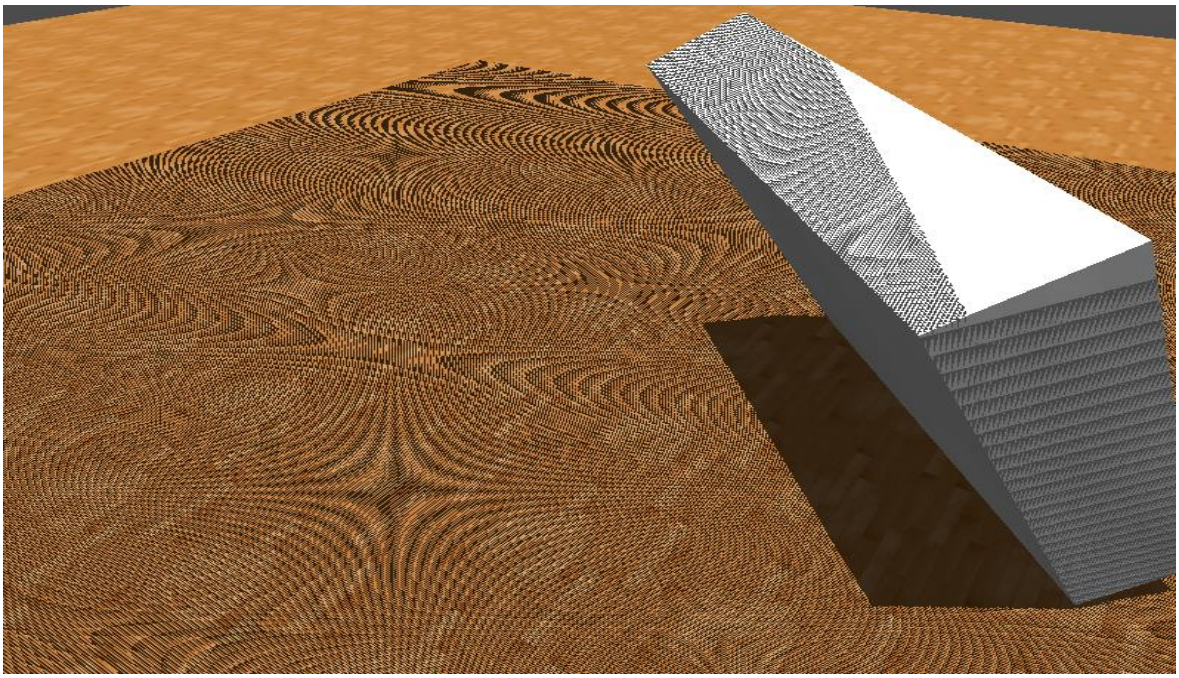
(Isječak koda 3) prikazuje dio koda iz sjenčara fragmenta koji implementira postupak posto-bližeg filtriranja.

```
// @return Percentage of fragment in shadow.
// @param LightSpaceFragmentPos fragment position in projected light space coordinates
float CalculateShadow(vec4 LightSpaceFragmentPos)
{
    // Perform perspective divide (coordinates are now in range [-1.0, 1.0]).
    vec3 ProjectedCoords = LightSpaceFragmentPos.xyz / LightSpaceFragmentPos.w;

    // transform coordinates from range [-1.0, 1.0] to [0, 1] because
    // texture coordinates are specified in that range.
    ProjectedCoords = ProjCoords * vec3(0.5) + vec3(0.5);

    // Current fragment depth.
    float CurrentDepth = ProjCoords.z;

        // Size of 1 texture element.
    vec2 TexelSize = 1.0 / textureSize(ShadowMapSampler, 0);

    float Shadow = 0.0;
    // Sample depth from 3x3 texture elements.
    for(int x = -1; x <= 1; ++x)
    {
        for(int y = -1; y <= 1; ++y)
        {
            float PcfDepth = texture(
                ShadowMapSampler, ProjectedCoords.xy + vec2(x, y) * TexelSize).x;

            if(CurrentDepth > PcfDepth)
            {
                Shadow += 1.0;
            }
        }
    }
    // Calculate shadow factor as percentage.
    Shadow /= 9.0;
    return Shadow;
}
```

Isječak koda 3. Implementacija posto-bližeg filtriranja

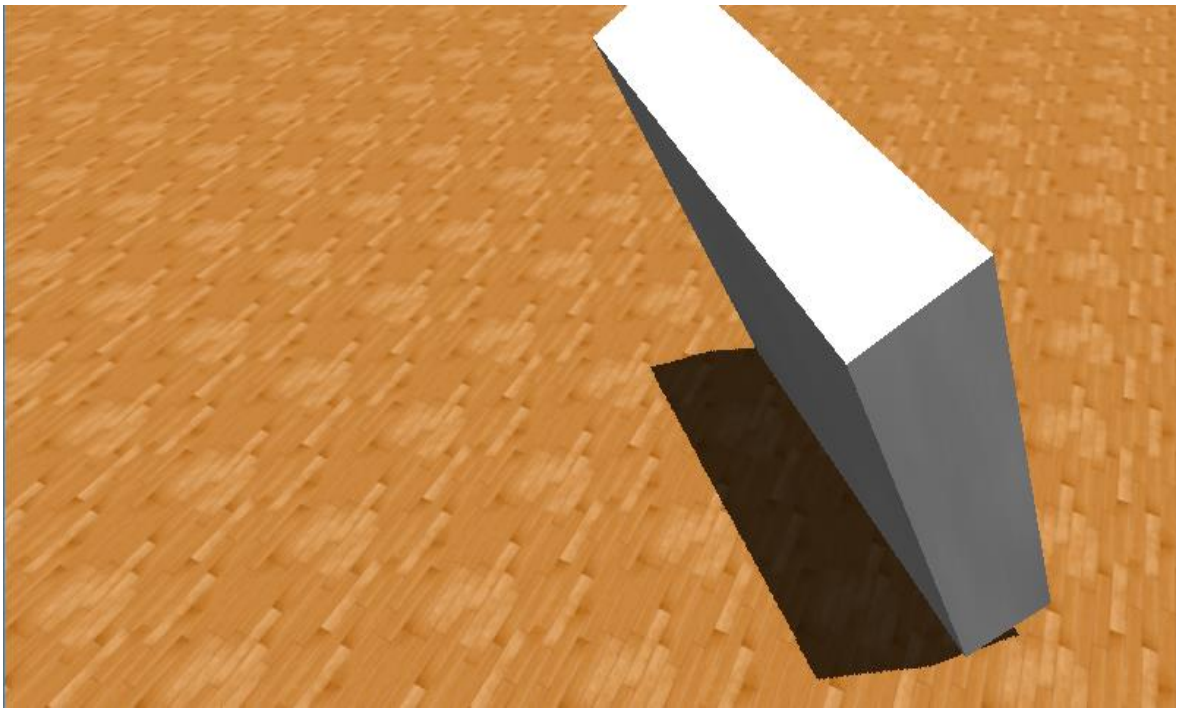
5.1.2 Problem samosjenčanja

Problem samosjenčanja (engl. self-shadowing) je pojava da poligon baca sjenu na samog sebe. Taj problem se manifestira kao tamne pruge koje se pojavljuju na površini predmeta. Do ovog problema dolazi zbog nepreciznosti prilikom usporedbe dubina zbog koje se za neke točke pogrešno izračuna da su u sjeni. Slika 7 prikazuje manifestaciju tog problema.



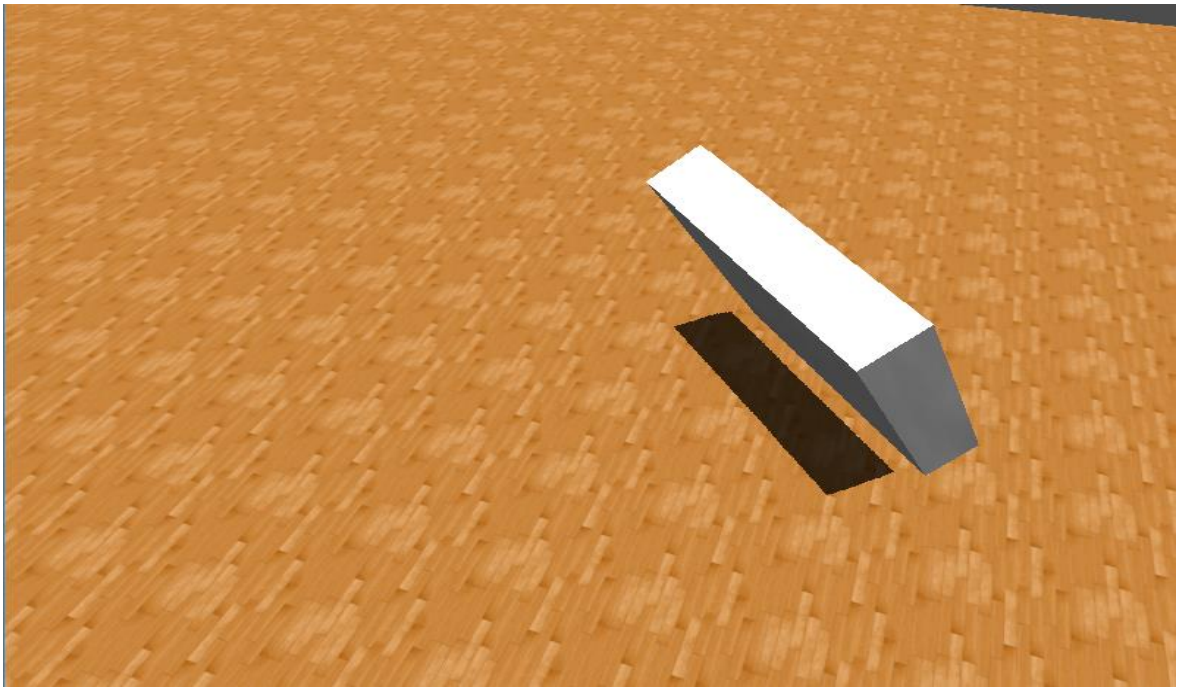
Slika 7. Problem samosjenčanja

Problem je moguće jednostavno ukloniti uvođenjem korektivnog člana (engl. Bias) koji se oduzme od dubine u točki prije usporedbe s dubinom iz teksture sjena.
Slika 8 prikazuje scenu nakon uvođenja korektivnog člana.



Slika 8. Scena nakon uvođenja korektivnog člana

Međutim, veći iznos korektivnog člana rezultira učinkom „Petra Pana“, to jest pojavom da je sjena pomaknuta u odnosu na predmet pa taj član treba postaviti na manju vrijednost. Slika 9 prikazuje učinak „Petra Pana“.



Slika 9 Učinak "Petra Pana" koji nastaje uvođenjem prevelikog korektivnog člana

Isječak koda 4 prikazuje dio koda iz sjenčara fragmenta koji implementira postupak posto-bližeg filtriranja te uvodi korektivni član kojim se rješava problem samosjenčanja.

```
// @return Percentage of fragment in shadow.
// @param LightSpaceFragmentPos fragment position in projected light space coordinates
float CalculateShadow(vec4 LightSpaceFragmentPos)
{
    // Perform perspective divide (coordinates are now in range [-1.0, 1.0]).
    vec3 ProjectedCoords = LightSpaceFragmentPos.xyz / LightSpaceFragmentPos.w;

    // transform coordinates from range [-1.0, 1.0] to [0, 1] because
    // texture coordinates are specified in that range.
    ProjectedCoords = ProjCoords * vec3(0.5) + vec3(0.5);

    // Current fragment depth.
    float CurrentDepth = ProjCoords.z;

        // Size of 1 texture element.
    vec2 TexelSize = 1.0 / textureSize(ShadowMapSampler, 0);

    // bias value that eliminates self shadowing problem.
    float Bias = 0.05;

    float Shadow = 0.0;
    // Sample depth from 3x3 texture elements.
    for(int x = -1; x <= 1; ++x)
    {
        for(int y = -1; y <= 1; ++y)
        {
            float PcfDepth = texture(
                ShadowMapSampler, ProjectedCoords.xy + vec2(x, y) * TexelSize).x;

            if(CurrentDepth - Bias > PcfDepth)
            {
                Shadow += 1.0;
            }
        }
    }
    // Calculate shadow factor as percentage.
    Shadow /= 9.0;
    return Shadow;
}
```

Isječak koda 4. Implementacija posto-bližeg filtriranja i uvođenje korektivnog člana u izračun utjecaja sjene

5.2 Ograničenja metode teksture sjena

Osim problema s nazubljenosti i samosjenčanjem, metoda teksture sjena također ovisi o vrsti izvora svjetlosti. Isječak koda 4 za izračunavanje učinka sjene u točki je dobar ukoliko je izvor svjetlosti usmjereno svjetlo. To proizlazi iz svojstva tog izvora svjetlosti, a to je da sve zrake svjetlosti koje dolaze od usmjerenog svjetla imaju isti smjer. Iz tog razloga za takvu vrstu izvora dovoljno je imati samo jednu teksturu sjene. Ako metodu teksture sjena želimo koristiti s točkastim izvorima svjetlosti potrebno je stvoriti kockastu teksturu sjene. Ideja je da se scena iscrta iz perspektive svjetla šest puta u šesterostranu kockastu teksturu. Takva tekstura onda sadrži podatke o dubini za čitavu okolinu svjetla. U tom slučaju teksturu uzorkujemo trodimenzionalnim vektorom.

Još jedno ograničenje metode teksture sjena je ta da kvaliteta sjene ovisi o veličini teksture. Taj problem se u teoriji može riješiti povećanjem razlučivosti teksture, ali to je zbog memorijskih ograničenja i brzine moguće samo do određene mjere. Tako se primjerice na mom računalu s grafičkom karticom Nvidia GTX 850m, scena s jednim objektom od 50000 poligona bez sjene prikazuje u 700 sličica u sekundi (engl. Frames per second) dok se sa uključenim izračunom sjene gdje je tekstura sjene veličine 1000x1000 slikovnih elemenata broj sličica smanjuje na 360 sličica u sekundi. Ovakav rezultat dobio sam mjereći vrijeme prije poziva funkcije za iscrtavanje te neposredno nakon završetka te funkcije. Oduzimanjem tih dvaju vremena dobivamo vrijeme potrebno za iscrtavanje jedne sličice. Iako ovi izračuni nisu u potpunosti točni s obzirom da je teško odrediti koliko točno traje iscrtavanje koje izvodi grafička kartica, ipak možemo zaključiti da iscrtavanje sjene ima velik utjecaj na brzinu izvođenja programa te je potrebno pronaći kompromis između kvalitete i brzine. Iz tog razloga razvile su se brojne tehnike kojima se pokušava riješiti taj problem.

6. Metoda bacanja zrake

Bacanje zrake (engl. Ray casting) je općenita metoda kojom se može riješiti niz problema u računalnoj grafici pa tako i problem određivanja sjene u sceni. Posebno je pogodna ako se za iscrtavanje scene koristi metoda praćenja zrake jer se u tom slučaju za metodu bacanja zrake može koristiti programski kod koji implementira metodu praćenja zrake. Primjerice, ako scenu iscrtavamo algoritmom praćenja zrake, a boju u svakoj točki računamo Phongovim modelom tada u sceni nećemo imat sjene jer ih Phongov model ne uzima u obzir. Međutim, algoritmom bacanja zrake možemo na jednostavan način utvrditi je li neka točka u sjeni ili nije.

Ideja je da za svaku točku scene „ispalimo“ zraku od točke koju promatramo prema izvoru svjetlosti. Ukoliko se zraka siječe s neprozirnim predmetom, lokalni doprinos osvjetljenja je nula. Pritom možemo u obzir uzeti i prozirne predmete na način da intenzitet svjetlosti moduliramo faktorom prozirnosti. Postupak je jednostavan jer ukoliko koristimo iscrtavanje metodom praćenja zrake, tada već imamo implementirane metode koje traže presjek zrake s objektima u sceni, pa taj kod onda koristimo i za pronalaženje presjeka između objekata u sceni i zrake koja ispituje utjecaj sjene u točki.

Isječak koda 5 prikazuje primjer implementacije kojom se provjerava je li neka točka u sjeni metodom bacanja zrake.

```
bool IsPointInShadow( Point point )
{
    // Create ray from point to light
    Ray ray = new Ray(point, lightPosition);

    // Check intersection with every object in scene
    for(int i = 0; i < numberOfObjects; ++i)
    {
        // objects variable represents array of all objects in scene
        bool intersection = objects[i].intersection(ray);

        // if intersection with any object is found then point is in shadow
        if(intersection)
        {
            return true;
        }
    }
    // if there is no intersection with any object in the scene then point is not in
shadow.
    return false;
}
```

Isječak koda 5. Provjera je li točka u sjeni metodom bacanja zrake

7. Zaključak

U računalnoj grafici postoji mnoštvo metoda kojima se sjene mogu iscrtavati. Svake godine pojavljuju se nove metode i varijacije postojećih metoda od kojih svaka ima svoje prednosti i nedostatke.

U ovom radu prikazan je osnovni koncept metode teksture sjena i njegova implementacija kao i neka jednostavna poboljšanja kojima se mogu ostvariti realističnije sjene. Osim metode teksture sjena koja je pogodna za iscrtavanje u realnom vremenu, opisana je i metoda bacanja zrake koja je vrlo prikladna kada se kao postupak iscrtavanja koristi algoritam praćenja zrake jer omogućava da se programski kod koji se koristi pri iscrtavanju scene koristi i za izračun sjene.

Budući da svi algoritmi iscrtavanja sjena uključuju određene aproksimacije, možemo zaključiti kako svaka od metoda iscrtavanja sjena ima svoje prednosti i nedostatke te se nedostaci na razne načine pokušavaju prekriti. Na kraju, odluka o tome koji algoritam koristiti ovisi o onome što želimo postići, ograničenjima uređaja te potrebnoj brzini izvođenja.

LITERATURA

[1] M. Čupić i Ž. Mihajlović, Interaktivna računalna grafika kroz primjere u OpenGL-u, Zagreb: Sveučilište u Zagrebu, 2016.

[2] I. Pandžić, T. Pejša, K. Matković, H. Benko, A. Čereković i M. Matijašević, Virtualna Okruženja: Interaktivna 3D grafika i njene primjene, Zagreb: Element, 2011.

[3] Learn OpenGL https://learnopengl.com/ Travanj 2017.

[4] OpenGL Step by Step - OpenGL Development, ogldev.atspace.co.uk. Travanj 2017.

[5] Microsoft MSDN, https://msdn.microsoft.com/en-us/library/windows/desktop/ee416324(v=vs.85).aspx. Travanj 2017.

[6] Wikipedia, https://en.wikipedia.org/wiki/Shadow_mapping. Travanj 2017.

[7] Wikipedia, https://en.wikipedia.org/wiki/Ray_tracing_(graphics). Travanj 2017.

[8] OpenGL tutorial www.opengl-tutorial.org. Travanj 2017.

Preslikavanje Sjene

Sažetak

Ovaj završni rad obrađuje temu preslikavanja sjene. S obzirom da je svjetlost usko povezana sa sjenama, prvi dio rada je iz tog razloga posvećen izvorima svjetlosti te osvjetljenju. U njemu je opisan jedan od načina modeliranja osvjetljenja. Dan je pregled matematičkih formula pomoću kojih se računa osvjetljenje u pojedinoj točki scene te je prikazana jedna moguća implementacija takvog modela osvjetljenja. Drugi dio rada opisuje dvije različite metode za iscrtavanje sjene: metodu teksture sjena i metodu bacanja zrake. U njemu su izložene osnovne ideje tih dviju metoda, nedostatci i prednosti tih metoda i njihove programske implementacije.

Ključne riječi: metoda teksture sjena, metoda bacanja zrake, Phongov model osvjetljenja

Shadow Mapping

Abstract

This final paper describes shadow mapping techniques. Since the light is closely related to shadows, the first part of the work is therefore devoted to light sources and lighting. It describes one of the ways of modeling the illumination. It contains an overview of mathematical formulas by which light is calculated at each point in the scene and one possible implementation of such lighting model is shown. The second part of the paper describes two different methods used for shadow rendering: Shadow mapping method and ray casting method. It describes the basic ideas of these two methods, their advantages and disadvantages and their programming implementations.

Keywords: shadow mapping, ray casting, Phong lighting model.