SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 560

POSTUPCI OSTVARIVANJA SJENE

Mihajlo Arbanas

Zagreb, lipanj 2013

Zahvaljujem mentorici prof. dr. sc. Željki Mihajlović na pomoći pri izradi ovog rada

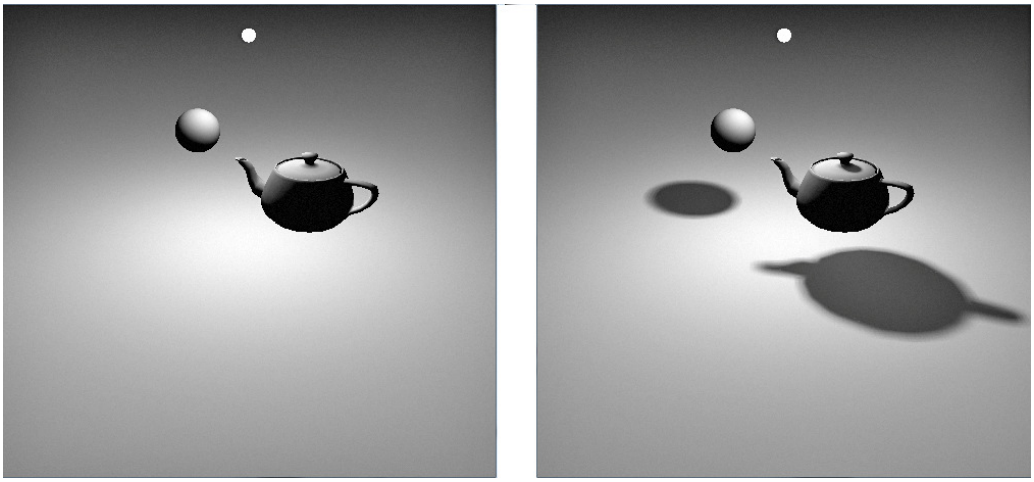
Svojim roditeljima i bratu na strpljenju i podršci prilikom studiranja

i djevojci Martini Perušina na velikoj podršci.

Sadržaj

1. Uvod

Sjena je prostor gdje je izvor svjetla blokiran nekim objektom. U računalnoj grafici sjene znatno povećavaju realizam slike. Sjene daju osjećaj pozicije objekta u odnosu na druge objekte. Bez sjena ne postoji dojam gdje se objekt nalazi u sceni.



Slika 2.1. Scena bez sjena i ista scena sa sjenama

Na slici 1.1 lijevo je prikazana scena bez iscrtavanja sjena. U takvom slučaju pozicija objekata nije poznata, odnosno oni se mogu nalaziti na podu, u zraku, bliže kameri ili još dalje od nje. Desno je ista scena sa sjenama, te se sada vidi da su oba objekta u zraku.

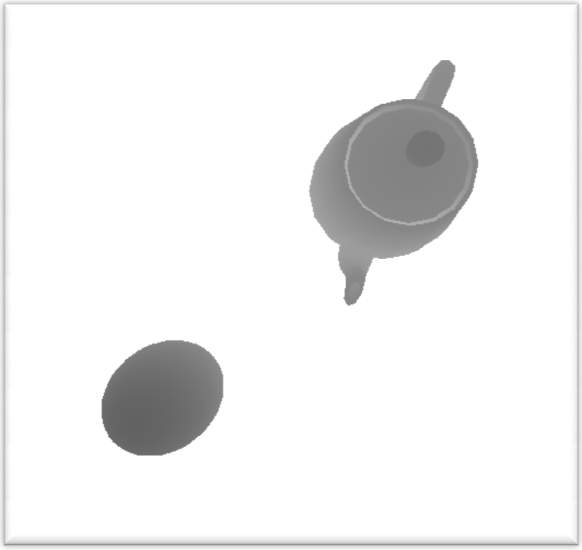
Postoje razne tehnike iscrtavanja sjena, od fotorealističnih do jednostavnih manje preciznih sjena. Fotorealistične sjene se postižu algoritmima praćenja zrake (**ray-tracing**), čime se prati stvaran put koji bi svjetlo prešlo od izvora do pogleda. U iscrtavanju u stvarnom vremenu (**real-time rendering**) algoritmi praćenja zrake nisu dovoljno brzi te se tražio lakši način iscrtavanja sjene. Projektivne sjene su jedan od jednostavnih načina u kojem bi se projektirao objekt na ravnu površinu te bi tako stvorio sjenu (Liland, 2002). Projektivne sjene rade za ravne površine, no ne i za zakrivljene koje primaju sjenu. Teksture sjena su drugi način kod kojeg bi se generirala slika objekta koji baca sjenu iz pogleda svjetla te bi se ta tekstura projektirala na objekt koji prima sjenu (Projective texture mapping). Ovime se postižu sjene i na zakrivljenim površinama, no problem se javlja kada objekt treba bacati sjenu na samog sebe.

U ovom radu će se obraditi dvije najčešće tehnike generiranja sjena, tehnika mapa sjena (**shadow mapping**) te tehnika volumnih sjena (**shadow volumes**), s kojima je moguće postići vjerni prikaz sjena.

2. Mape sjena (shadow mapping)

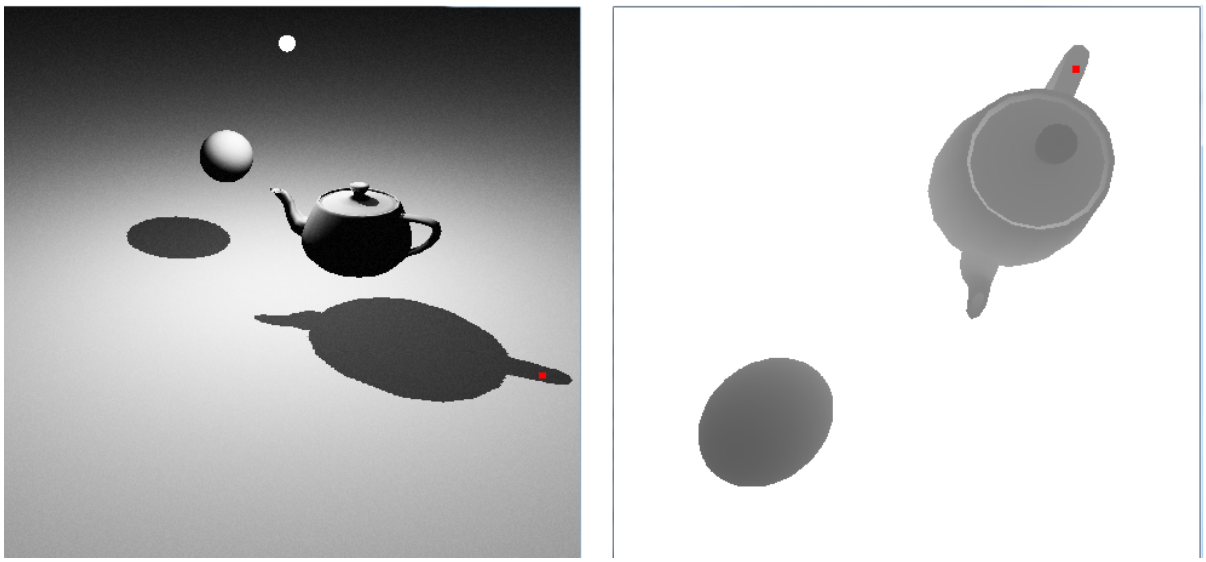
Mape sjena (shadow mapping) je tehnika prikazivanja sjena korištenjem dubinske mape generirane iz izvora svjetla. Tehniku je smislio Lance Williams 1978. godine i opisao je u članku "Casting curved shadows on curved surfaces" (Williams, 1978). Od tada se često koristi kako u **fotorealističnom**, tako i u ostvarivanju prikaza u stvarnom vremenu (**real-time rendering**).

Tehnika se temelji na generiranju dubinske slike, tj. mape (**depth map, z-map**) iz pogleda svjetla i njenom korištenju pri generiranju slike iz pogleda kamere u sceni. Prvo se pogled na scenu postavi u izvor svjetla te se generira samo dubinska mapa, pri čemu nije potrebno generirati ostale mape (npr mapa boje, **color map**). Takvu mapu je potrebno spremiti i koristiti pri ponovnom generiranju slike, ovog puta iz pogleda kamere. Prilikom novog generiranja slike (i nove dubinske mape), uspoređuje se udaljenost pojedinih elemenata slike od svjetla sa dubinom odgovarajućih elemenata iz prve dubinske mape. Ako je dubina elementa slike veća od dubine elementa iz prve dubinske mape tada je taj element u sjeni, i u slici se iscrtava zasjenjen, inače je na svjetlu i iscrtava se osvijetljen. Prije usporedbe još je potrebno elemente nove slike postaviti (projicirati) u pogled svjetla, kako bi usporedba bila točna.



Slika 2.1. Dubinska mapa iz pogleda svjetla.

Na slici 2.1. prikazana je dubinska mapa generirana iz pogleda svjetla. Svjetlija boja označava elemente koji su dalje od svjetla, a tamnija one bliže. Bijela boja (u ovom primjeru pozadina) označava elemente koji su najdalji. Moguće je podesiti generiranje dubinske mape tako da tamnija boja označava veću udaljenost.



Slika 2.2. Slika generirana iz pogleda kamere (lijevo) i dubinska mapa generirana iz pogleda svjetla (desno)

Slika 2.2. prikazuje sliku scene generirane iz pogleda kamere i dubinsku mapu generiranu iz pogleda svjetla. Crveni kvadratić na lijevoj i desnoj slici označava dva elementa čija će se udaljenost od svjetla testirati. Prije nego je testiranje moguće potrebno je element iz pogleda kamere smjestit u pogled svjetla na način da se njegova pozicija transformira matricom pogleda (**View matrix)** i projekcijskom matricom (**Projection matrix)** korištenom pri generiranju dubinske mape iz pogleda svjetla. (U OpenGL-u je također još prvo potrebno transformirati element u prostor scene inverznom transformacijom matrice pogleda kamere).

2.1.Implementacija mapa sjena

Prvo je potrebno generirati dubinsku mapu kakva se vidi iz pogleda svjetla. Nju je kasnije potrebno koristiti za određivanje sjena. Zbog toga ju je potrebno generirati kao teksturu koja se kao takva može ponovno upotrebljavati. Kako bi se to postiglo, u OpenGL API-ju je potrebno napraviti novi spremnik okvira (engl. **framebuffer)**. Oni služe kao dodatni spremnici uz glavni spremnik okvira, te se generiranje slika odnosno prikazivanje može preusmjeriti u njih.

Najprije se izrađuje tekstura u koju se sprema dubinska mapa:

```
glGenTextures(1, &depth);
```

U *depth* variablu tipa *integer* se sprema broj koji govori OpenGL-u kojom teksturom se želi baratati. Kao prvi parametar glGenTextures prima broj koliko tekstura se izrađuje.

Kada je izrađena nova tekstura potrebno je zadati OpenGL-u kako da rukuje sa njome, što će se u nju spremati, koji je njezin format. To se čini sljedećim naredbama:

```
glBindTexture(GL_TEXTURE_2D, depth);

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_DEPTH_TEXTURE_MODE,GL_INTENSITY);

glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, širina_mape,
       visina_mape, 0, GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, NULL);
```

Prvom naredbom aktivira se *depth* tekstura, za koju će se sljedećim naredbama postavljati njezini parametri. Sljedeće dvije funkcije određuju kako će se uzorkovati iz nje ako koordinate uzorkovanja budu izvan dimenzija teksture. U ovom slučaju postavljeno je tako da će se uzorkovati rub teksture. Treća i četvrta *glTexParameteri* funkcija određuju kakvo će biti uzorkovanje u slučaju kada je tekstura uvećana odnosno umanjena. Zadnja *glTexParameteri* funkcija postavlja kako će se tretirati vrijednosti u dubinskoj mapi tijekom filtriranja i preslikavanja teksture. Zadnjom naredbom postavljaju se dimenzije teksture te njezin format i tekstura je spremna za upotrebu.

Sljedeće je potrebno izgraditi novi spremnik okvira i dodati mu teksturu *depth* kao spremnik (engl. **buffer**) za spremanje dubine.

```
glGenFrameBuffersEXT(1, &frame);
glBindFrameBufferEXT(GL_FRAMEBUFFER_EXT, frame);

glFrameBufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
            GL_DEPTH_ATTACHMENT_EXT, GL_TEXTURE_2D, depth, 0);
```

Prvom funkcijom generira se jedan framebuffer koji se može referencirati variablom *frame*. Sljedećom funkcijom postavlja se taj spremnik okvira kao aktivni kako bi mu se dodala *depth* tekstura. Funkcija *glFrameBufferTexture2DEXT* uz parametar GL_DEPTH_ATTACHMENT_EXT dodaje *depth* teksturu kao buffer za spremanje dubine aktivnom framebufferu.

Sada je spremink okvira spreman za korištenje, i može se generirati dubinska mapa scene iz pogleda svjetla. Aktivira se napravljeni spremnik okvira, postavlja se pogled u izvor svjetla i iscrtavaju se objekti (modeli) u sceni. Također, potrebno je spremiti matricu koja je korištena za iscrtavanje objekata, kako bi se kasnije tijekom iscrtavanja završne slike moglo pomoću nje transformirati elemente objekta u pogled svjetla. Sljedeći dijelovi koda se izvršavaju za svaku iscrtanu sliku (engl. **frame**), i potrebno ih je staviti zajedno u petlju.

```
glDisable(GL_LIGHTING);
glEnable(GL_DEPTH_TEST);

glBindFrameBufferEXT(GL_FRAMEBUFFER_EXT, frame);
glViewport(0,0,širina_mape,visina_mape);
glClear(GL_DEPTH_BUFFER_BIT);
glDepthMask(true);
glColorMask(false,false,false,false);
glCullFace(GL_FRONT);
```

Kako bi se ubrzalo iscrtavanje dubinske mape onemogućeni su crtanje u spremnik boja (engl. **color buffer**) sa funkcijom *glColorMask(false,false,false,false)* i izračuni osvjetljenja sa funkcijom *glDisable*(*GL_LIGHTING*). Postavljanjem *glCullFace* na GL_FRONT, iscrtavaju se samo stražnji poligoni objekta kako bi se smanjili neželjeni efekti nepreciznosti spremnika dubine (engl. **z-buffer, depth buffer**). Sa *glViewport* postavlja se otvor u koji će se iscrtavati dubinska mapa.

```
float SvjetloProjectionMatrix[16];
float SvjetloViewMatrix[16];
glMatrixMode(GL_PROJECTION);

      glLoadIdentity();
      gluPerspective(FOV,1.0,Z_CLIP_NEAR,Z_CLIP_FAR);

      glGetFloatv(GL_PROJECTION_MATRIX, SvjetloProjectionMatrix);

glMatrixMode(GL_MODELVIEW);

      glLoadIdentity();
      gluLookAt(pozicija_svjetla,točka_pogleda, up_vector);

      glGetFloatv(GL_MODELVIEW_MATRIX, SvjetloViewMatrix);
```

Postavljaju se matrice kojima će se iscrtati scena. *glGetFloatv* sa zadanim parametrima dobavlja projekcijsku matricu u *SvjetloProjectionMatrix* i matricu pogleda u *SvjetloViewMatrix* koje će se koristiti kasnije. OpenGL koristi jednu matricu ModelView koja je umnožak dvije, Model i View matrice. View matricu možemo dobiti tako da (npr. sa *glGetFloatv*) se dobavi ModelView matrica prije nego što se iscrta objekt, a nakon što je postavljeno očište u pogled kamere.
Sada se mogu iscrtati objekti u sceni. Ti objekti će bacati sjenu na sve objekte.

```
glMatrixMode(GL_MODELVIEW);

      glPushMatrix();

            glTranslatef(Objekt.position.x, Objekt.position.y,
                        Objekt.position.z);
            glRotatef(Objekt.rotation.w, Objekt.rotation.x,
                  Objekt.rotation.y, Objekt.rotation.z);

glMatrixMode(GL_TEXTURE);
      glActiveTextureARB(GL_TEXTURE7);

      glPushMatrix();

            glTranslatef(Objekt.position.x, Objekt.position.y,
                        Objekt.position.z);
            glRotatef(Objekt.rotation.w, Objekt.rotation.x,
                  Objekt.rotation.y, Objekt.rotation.z);


      Render(Objekt, GL_TRIANGLES);


glMatrixMode(GL_TEXTURE);
      glPopMatrix();
glMatrixMode(GL_MODELVIEW);
      glPopMatrix();
```

Ovim naredbama se rotira i translatira objekt te se naposljetku iscrtava. Rotaciju i translaciju je potrebno postaviti u dvije matrice. U matricu *ModelView* postavlja se rotacija i translacija objekta kako bi se on pravilno pozicionirao u sceni, a u matricu Texture istim transformacijama postavljamo koordinate teksture u odgovarajuće mjesto. Aktivna tekstura za koju će se transformirati koordinate je GL_TEXTURE7, no moguće je uzeti bilo koju drugu teksturu.

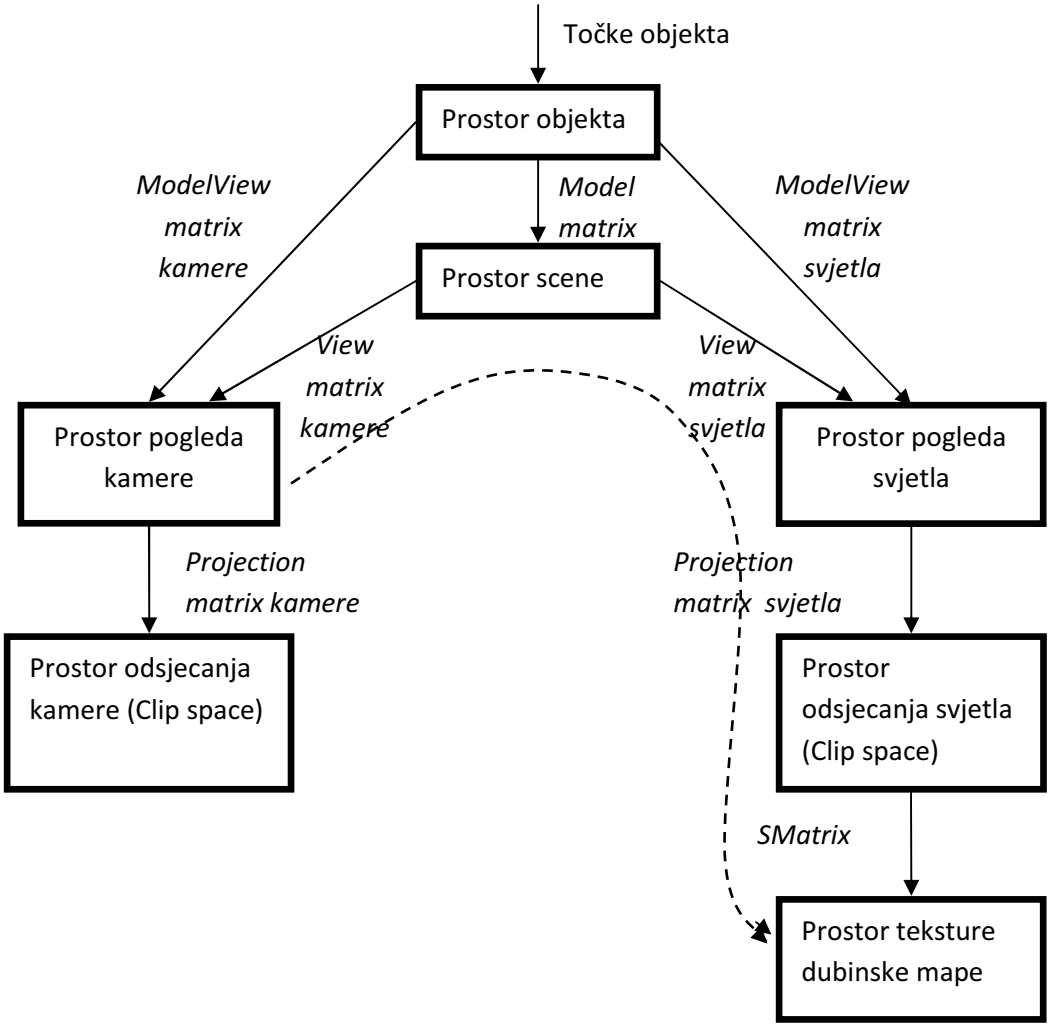
Sada je potrebno spremiti matricu *SvjetloProjectionMatrix* i *SvjetloViewMatrix* u matricu *Texture*, kako bi im se moglo pristupiti u programu za sjenčanje (**shader**).

```
float SMatrix[16] = {0.5, 0.0, 0.0, 0.0,
                     0.0, 0.5, 0.0, 0.0,
                     0.0, 0.0, 0.5, 0.0,
                     0.5, 0.5, 0.5, 1.0};
glMatrixMode(GL_TEXTURE);
glActiveTextureARB(GL_TEXTURE7);

      glLoadIdentity();
      glLoadMatrixf(SMatrix);
      glMultMatrixf(SvjetloProjectionMatrix);
      glMultMatrixf(SvjetloViewMatrix);

      glMatrixMode(GL_MODELVIEW);

glBindFrameBufferEXT(GL_FRAMEBUFFER_EXT,0);
```

U matricu *Texture* učitava se matrica *SMatrix* i množi se sa *SvjetloProjectionMatrix* i *SvjetloViewMatrix*. Matrica *SMatrix* je potrebna kako bi se koordinate točaka modela, (**vertexe**) koji su nakon svođenja u ravninu projekcije u rasponu [-1,1] (**normalizirane koordinate**), postavile u raspon [0,1] što je potrebno za pravilno uzorkovanje dubinske mape. Matrica *Texture* tada sadrži tri matrice pomnožene po redosljedu:

$$\text{TextureMatrix} = \text{SMatrix} * \text{ProjectionMatrix} * \text{ViewMatrix} \qquad \textit{(jed. 1.)}$$

U programu za sjenčanje će se tada svaka točka objekta množiti najprije sa inverzom matrice pogleda kamere (ViewMatrix kamere) i matricom *Texture*, čime se direktno dobivaju teksturne koordinate pojedinog elementa objekta. Ovaj postupak se naziva **projektivno teksturiranje** (engl. **projective texture mapping**). Dijagramom 2.1. je prikazan opisani postupak pretvorbe koordinata točaka objekta iz pogleda kamere u koordinate koje služe za uzorkovanje dubinske mape.



Dijagram 1. Prikaz transformacije točaka objekta iz pogleda kamere u prostor koordinata teksture dubinske mape

Dijagram 2.1. prikazuje kako se objekti iz svog lokalnog prostora objekta transformacijama smještaju u ostale prostore. Iscrtkanom linijom prikazane su transofmacije koje su potrebne da bi se objekt iz prostora pogleda kamere (u koji je smješten nakon množenja ModelView matricom) smjestio u prostor teksture dubinske mape. Dijagram je temeljen na dijagramu sa stranice (OpenGL Shadow Mapping Tutorial - Paul's Projects).

Dubinska mapa je generirana, te će se zatim iscrtati scena uz njenu uporabu. Iscrtavanje se obavlja uz korištenje programa za sjenčanje (shader), kojim se zatamnjuju oni slikovni elementi koji su u sjeni.

Prvo je potrebno učitati izvorne kodove programa za sjenčanje. Potrebni su izvorni kodovi za sjenčanje vrhova i sjenčanje fragmenata (engl. **vertex shader, fragment shader**). Izvorni kodovi su pisani u GLSL jeziku čija je sintaksa slična C/C++ jeziku. Oni se zatim prevode (kompajliraju) i povezuju u program za sjenčanje.

```
char* VertexShaderKod, *FragmentShaderKod;
/* učitavanje izvornog koda iz datoteka */
int vertex = glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);
glShaderSourceARB(vertex,1, VertexShaderKod,&duzina_vertex);
glCompileShaderARB(vertex);

int fragment = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
glShaderSourceARB(fragment,1, FragmentShaderKod,&duzina_fragment);
glCompileShaderARB(fragment);

int shader_program = glCreateProgramObjectARB();
glAttachObjectARB(shader_program,vertex);
glAttachObjectARB(shader_program,fragment);
glLinkProgramARB(shader_program);
```

Sada je *shader_program* spreman za korištenje. Prilikom svakog iscrtavanja program je potrebno aktivirati naredbom *glUseProgramObjectARB(shader_program)*. Tada se, umjesto korištenja cjevovoda fiksnim funkcijama (**fixed function pipeline**) kroz cijeli proces iscrtavanja slike, u određene stadije uvodi program za sjenčanje.

```
glBindFrameBufferEXT(GL_FRAMEBUFFER,0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glViewport(0,0,širina_prozora,visina_prozora);

glMatrixMode(GL_PROJECTION);

      glLoadIdentity();
      gluPerspective(FOV,1.0,Z_CLIP_NEAR,Z_CLIP_FAR);

glMatrixMode(GL_MODELVIEW);

      glLoadIdentity();
      gluLookAt(pozicija_kamere,točka_pogleda, up_vector);
      glGetFloatv(GL_MODELVIEW_MATRIX, KameraViewMatrix);
```

Postavlja se glavni spremnik (*framebuffer*) okvira kao aktivni i brišu se spremnici boja i dubine (*color* i *depth buffer*). Postavljaju se transformacijske matrice te se sa *glGetFloatv* iščitava ModelView matrica. Sada je na redu iscrtavanje objekata u sceni.

```
glUseProgramObjectARB(shader_program);

int uniView = glGetUniformLocationARB(shader_program,"ViewMatrix");
glUniformMatrix4fvARB(uniView, 1, 0, KameraViewMatrix);

int uniMapa = glGetUniformLocationARB(shader_program,"mapaDubine");
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, depth);
glUniform1iARB(uniMapa, 1);

glLightfv(0, GL_POSITION, pozicija_svjetla);

glMatrixMode(GL_MODELVIEW);

      glPushMatrix();
            glTranslatef(Objekt.position.x, Objekt.position.y,
                         Objekt.position.z);
            glRotatef(Objekt.rotation.w, Objekt.rotation.x,
                        Objekt.rotation.y, Objekt.rotation.z);

      Render(Objekt, GL_TRIANGLES);

glMatrixMode(GL_MODELVIEW);
      glPopMatrix();

glUseProgramObjectARB(0);

glutSwapBuffers();
```

Nakon što je sa *glUseProgramObjectARB shader_program* postavljen za korištenje, sa *glGetUniformLocationARB* dobavlja se referentni broj uniformne matrice ViewMatrix unutar programa za sjenčanje, koji se sprema u variablu *uniView*. S takvim brojem moguće je postaviti vrijednost odgovarajuće uniformne variable unutar programa za sjenčanje. Naredba *glUniformMatrix4fvARB* postavlja ViewMatricu unutar programa za sjenčanje na vrijednosti sadržane u *KameraViewMatrix*. Potrebno je dobaviti i referentni broj *mapeDubine* kako bi se programu za sjenčanje predala dubinska mapa *depth*. Tekstura *depth* veže se za GL_TEXTURE1 i sa funkcijom *glUniform1iARB* postavlja se GL_TEXTURE1 kao tekstura koju koristi *mapaDubine* u programu za sjenčanje. Naredbu *glGetUniformLocationARB* je potrebno samo jednom koristiti jer se referentni broj varijable unutar istog programa za sjenčanje neće mijenjati. Naredbom *glLightfv* postavlja se pozicija svjetla u sceni, što će se koristiti kao parametar pri izračunu osvjetljenja scene. Objekt se rotira i translatira te naposljetku iscrtava. Sa *glUseProgramObjectARB*(0) postavlja se na korištenje cjevovod sa fiksiranim funkcijama i naposlijetku se izmjenjuju prednji i stražnji spremnik sa *glutSwapBuffers*() te se prikazuje iscrtana scena sa sjenama.

Sada je potrebno napisati programe za sjenčanje vrhova i fragmenata (vertex i fragment shadere) koji će koristiti mapu sjena (tj. dubinsku mapu) i u skladu s tim osvjetliti elemente (piksele) slike. Programi za sjenčanje se pišu u OpenGl Shading Language (GLSL) jeziku, u običnom tekstualnom editoru, i kao takvi se učitavaju u OpenGL.

```
uniform mat4 ViewMatrix;

varying vec3 Normal;
varying vec4 VertexPosition;
varying vec4 LightDirection;
varying vec4 ShadowCoord;

void main(){

      gl_Position = ftransform();
      mat4 invViewMatrix = inverse(ViewMatrix);

      VertexPosition = gl_ModelViewMatrix * gl_Vertex;
      LightDirection = gl_LightSource[0].position – VertexPosition;
      Normal = gl_NormalMatrix * gl_Normal;
      gl_TexCoord[0] = gl_MultiTexCoord0;

      ShadowCoord = gl_TextureMatrix[7] *
                        (invViewMatrix * VertexPosition);
      }
```

U programu za sjenčanje vrhova se transformiraju koordinate tekstura i točaka objekta. Sa *gl_Position = ftransform()* izračunava se pozicija točke objekta u prostoru projekcije. Računa se i pozicija točke u prostoru pogleda kamere sa *VertexPosition = gl_ModelViewMatrix * gl_Vertex*. Smjer svjetla (*LightDirection*) i normala su potrebni pri izračunu osvjetljenja scene. U *gl_TexCoord[0]* se nalazi koordinata teksture za trenutnu točku objekta (vertex). Koordinata teksture dubinske mape (*ShadowCoord*) se računa iz pozicije točke objekta (*VertexPosition)*. Pozicija točke prvo se inverznom matricom pogleda kamere (ViewMatrix) transformira iz prostora pogleda kamere u prostor scene, te se tada iz prostora scene transformira u prostor koordinata teksture dubinske mape koristeći *gl_TextureMatrix[7]*. Matrica *gl_TextureMatrix[7]* se prethodno dobila jednadžbom [*jed. 1*] nakon iscrtavanja objekta u mapu sjena.

```
#define e 0.000005

uniform sampler2D mapaDubine;

varying vec3 Normal;
varying vec4 VertexPosition;
varying vec4 LightDirection;
varying vec4 ShadowCoord;

void main(){

      vec4 ShadowMapCoord = ShadowCoord / ShadowCoord.w;

      float shadowDist = texture2D(mapaDubine,ShadowMapCoord.xy).z;
      float shadow = 1.0;
      if(ShadowCoord.w > 0.0){
            shadow = shadowDist+e < ShadowMapCoord.z ? 0.25 : 1.0;}

      float dLight = dot(LightDirection.xyz,LightDirection.xyz);
      float dist = 1.0 / (e + dLight);

      float light_cos = dot(normalize(LightDirection.xyz),Normal);
      if(light_cos <= 0.0) dist = 0.0;
      dist *= light_cos;

      gl_FragColor =  shadow * vec4(dist,dist,dist,1.0);
      }
```

U programu za sjenčanje fragmenata se računa osvjetljenje pojedinog fragmenta (piksela) slike. Prvo se koordinate teksture dubinske mape normaliziraju dijeleći ih sa komponentom *w*.Tada se uzorkuje dubinska mapa sa funkcijom *texture2D* dobivajući dubinu iz pogleda svjetla. Ta dubina se uspoređuje sa dubinom fragmenta sadržanom u komponenti *z* vektora *ShadowMapCoord*. U slučaju da je dubina fragmenta veća od dubine dobivene iz dubinske mape fragment je u sjeni, i sjenča se tamnijom nijansom. U suprotnom fragment je u potpunosti izložen svjetlu. Kako bi se umanjila pogreška nastala zbog ograničene preciznosti z-spremnika, pri usporedbi se vrijednosti iz dubinske mape dodaje malo odstupanje *e*. Sjenčanje tamnijom nijansom može se postići tako da se izlazna nijansa za fragment množi s vrijednosti manjom od 1.0 (u ovom primjeru sa vrijednosti 0.25), smanjujući tako jačinu osvjetljenja fragmenta.

Izračun osvjetljenja se temelji na inverznoj udaljenosti fragmenta od izvora svjetla, dobivene iz izraza:

dLight = dot(*LightDirection.xyz*,*LightDirection.xyz*);

dist = 1.0 / (*e* + *dLight*);

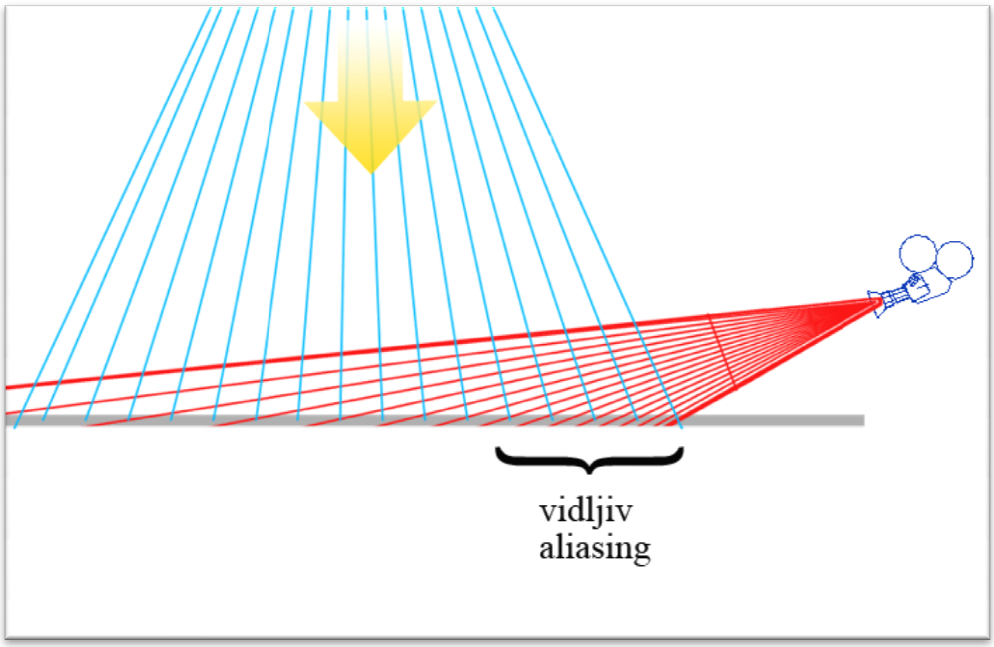
Osvjetljenje se još množi sa skalarnim produktom normale fragmenta i smjera prema svjetlu kako bi se dobilo realističnije osvjetljenje. Izlazna boja fragmenta se tada postavlja u ugrađenu varijablu *gl_FragColor*.

2.2.Nedostatci, greške i poboljšanja

Mape sjena su jedan od najčešće korištenih algoritama za generiranje sjena. No, one imaju svoja ograničenja. Najčešća greška koja je vidljiva je perspektivna pogreška uzorkovanja (engl. **aliasing**), no javljaju se i akne sjena te ograničenost područja koje je vidljivo iz pogleda svjetla.

Perspektivna pogreška uzorkovanja

Perspektivna pogreška uzorkovanja se javlja kada preslikavanje elemenata teksture (teksela) dubinske mape sa elementima (fragmentima, pikselima) slike nije u omjeru 1:1, već se više elemenata slike preslikava na isti element teksture dubinske mape. Uzrok ovome je ograničena rezolucija dubinske mape i uočljiviji je na elementima slike bližim kameri. Takvi elementi slike su i prostorno međusobno bliži, te se više njih preslikava na isti element dubinske mape. Što su elementi slike dalji, to oni pokrivaju veći prostor scene i manji broj ih se preslikava na isti element dubinske mape.

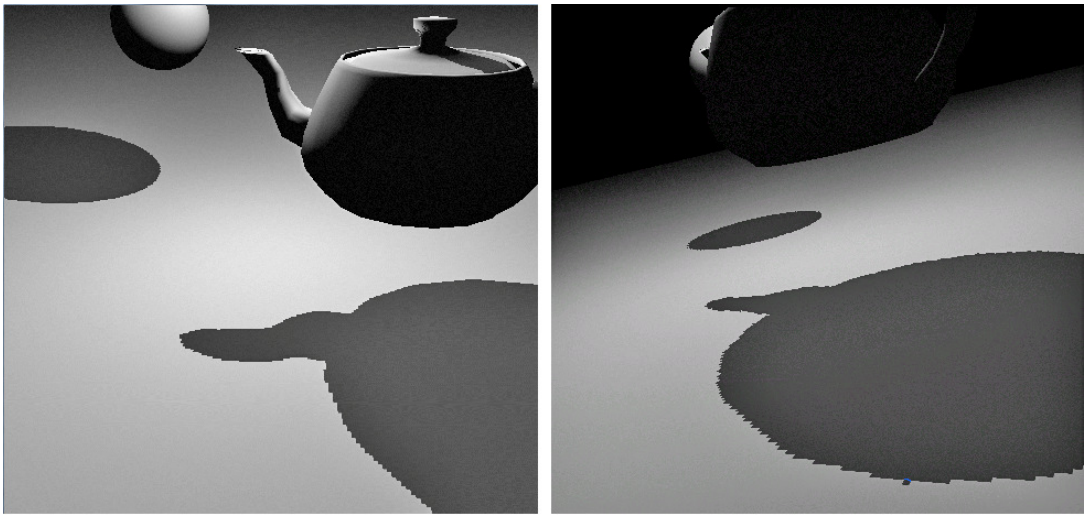


Slika 2.3. Prikaz vidljivosti perspektivnog aliasinga

Na slici 2.3. plave linije reprezentiraju projekciju pojedinih elemenata dubinske mape na ravnu površinu (označenu sivom bojom), dok crvene linije reprezentiraju projekciju elemenata slike na istu tu površinu. Između svake takve dvije linije se nalazi ono što taj element slike ili dubinske mape "vidi". U području gdje crvene linije siječu površinu koje

je bliže kameri, elementi slike su prostorno međusobno bliži te više njih obuhvaća isti element dubinske mape (omeđen sa dvije susjedne plave linije). Na tim mjestima je vidljiva perspektivna pogreška uzorkovanja. Što su crvene linije dalje od kamere, to je i razmak između sjecišta sa površinom veći, te takvi elementi slike obuhvaćaju veći prostor površine, a s tim ujedno i više elemenata dubinske mape.

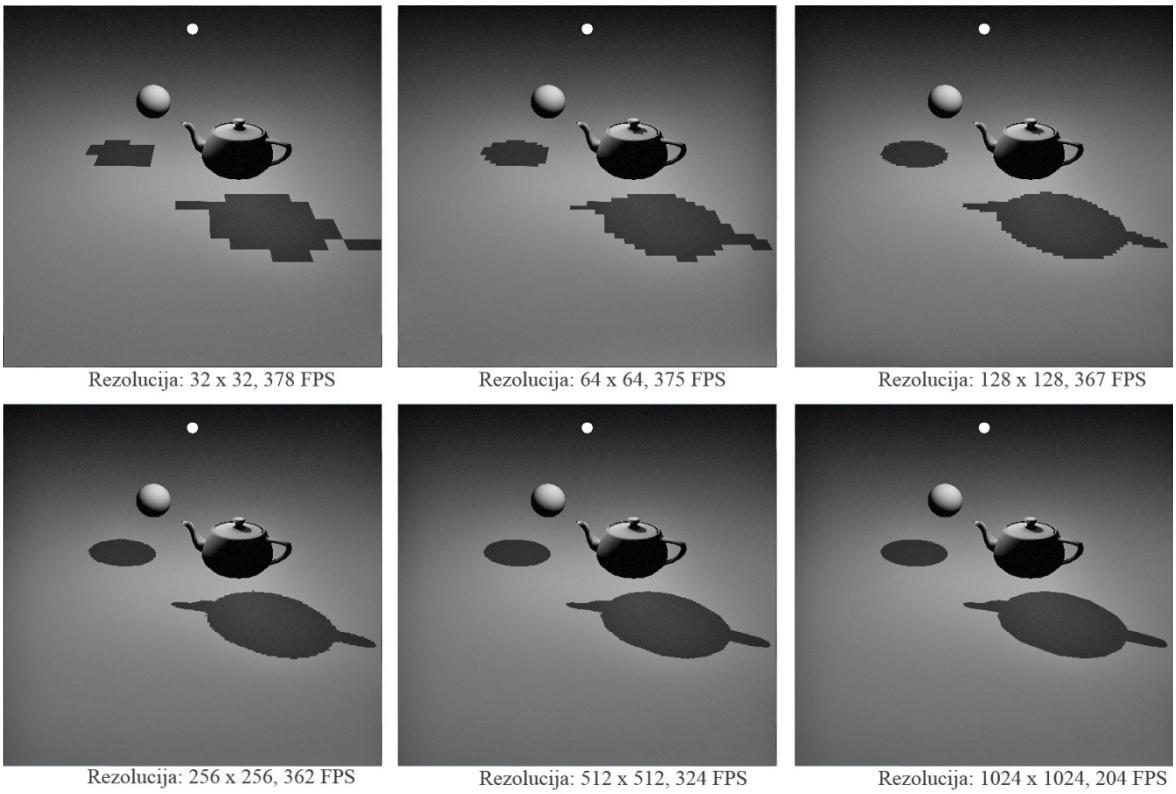
Na vidljivost aliasinga utječe blizina elementa slike kameri, no utječe i udaljenost izvora svjetla od elementa slike. Veća udaljenost izvora svjetla znači da i projekcija elementa dubinske mape obuhvaća veću površinu objekta koji prima sjenu. Ovaj slučaj aliasinga se javlja za dubinske mape koje su perspektivno projektirane. Također, što je objekt koji baca sjenu dalje od izvora svjetlosti, to je površina koju on zauzima u dubinskoj mapi manja, te je sve više vidljiv perspektivni aliasing.



Slika 2.5. Primjeri perspektivnog aliasniga

Slika 2.4. Perspektivni aliasnig za udaljeno svjetlo

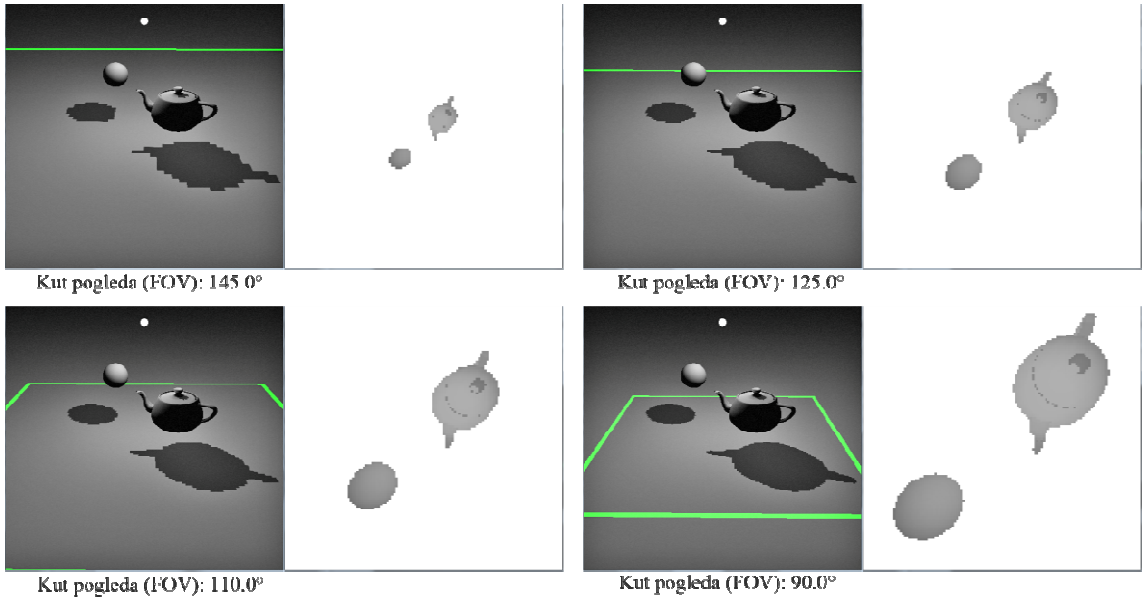
Jedan pristup rješavanju ovog problema bio bi povećavanje rezolucije dubinske mape. No, to nije dobro rješenje jer narušava brzinu iscrtavanja, a artefakti, iako u manjoj mjeri, ponovno mogu biti vidljivi. Utjecaj rezolucije spremnika dubinske mape je prikazan na slici 2.6. Povećavanjem rezolucije spremnika očito je poboljšanje kvalitete sjena. No približi li se kamera bliže objektu ponovo će se vidjeti perspektivni aliasing.



Slika 2.6. Različite rezolucije dubinske mape i brzina iscrtavanja u FPS

Drugi pristup bi bio da se iscrtavanje dubinske mape ograniči na prostor koji je trenutno vidljiv iz pogleda kamere, tj. da mapa usko obuhvati samo one objekte u sceni koji su trenutno vidljivi, smanjujući kut pogleda (**field of view**) i poklapajući projekciju pogleda svjetla sa prostorom koji vidi kamera. Tako se postiže veća rezolucija sjene za vidljive objekte, dok se ne iscrtavaju oni objekti koji ionako neće biti vidljivi. Taj pristup se naziva perspektivne mape sjena u prostoru svjetla (engl. light space perspective shadow maps). Ovime se postižu mnogo bolji rezultati (Common Techniques to Improve Shadow Depth Maps, 2012), posebno u području koje je bliže kameri gdje je potreban gušći raspored elemenata dubinske mape. Na slici 7. je prikazan utjecaj kuta pogleda na sjenu, za dubinsku mapu rezolucije 128 x 128 slikovna elementa. Sa velikim kutom pogleda obuhvaćen je i veći dio scene, te objekti u dubinskoj mapi zauzimaju manje slikovnih

elemenata, stvarajući tako lošiju sjenu sa izraženijom perspektivnom pogreškom uzorkovanja.



Slika 2.7. Utjecaj kuta pogleda na kvalitetu sjene (Dubinska mapa rezolucije 128x128)

Još jedna tehnika koja rješava problem perspektivne pogreške uzorkovanja su kaskadne mape sjena (eng. cascaded shadow maps). Tehnika se sastoji u tome da se generira više mapa sjena, svaka sa različitom razinom detalja. Pogled iz svjetla (**frustum**) se podijeli u više manjih dijelova, i za svaki se generira dubinska mapa za taj dio scene. Pri iscrtavanju slike se uzorkuje odgovarajuća dubinska mapa. Tako se postižu veći detalji uz povećanje performansi, jer se iscrtavaju dubinske mape sa manje detalja tamo gdje oni nisu potrebni (Cascaded Shadow Maps, 2012).

Pogreška uzorkovanja se može prikriti i zamućivanjem ruba sjene. Za razliku od normalnih tekstura, dubinske mape sjena se ne mogu unaprijed filtrirati kako bi se dobili glađi rubovi sjena. Stoga se zamućivanje postiže usrednjavanjem rezultata testiranja više susjednih elemenata mape sjena za svaki fragment slike zasebno. Na tom principu radi i Percentage-Closer Filtering (PCF) (Sanglard, Soft shadows with PCF, 2009). GLSL jezik ima već ugrađenu funkciju *shadow2DProj* koja automatski uzorkuje dubinsku mapu, testira na sjene i obavlja PCF nad rezultatima testiranja. Budući je *shadow2DProj* sklopovski podržana funkcija performanse neće biti puno narušene.

Upotreba *shadow2DProj* funkcije zahtijeva malu izmjenu dosadašnjeg programa. U programu za sjenčanje fragmenata će umjesto tipa podatka *sampler2D* za mapu dubine biti korišten *sampler2DShadow*.

```
uniform sampler2DShadow mapaDubine;
```

Nadalje, dio koji uzorkuje i testira dubinsku mapu

```
float shadowDist = texture2D(mapaDubine,ShadowMapCoord.xy).z;
if(ShadowCoord.w > 0.0){
      shadow = shadowDist+e < ShadowMapCoord.z ? 0.25 : 1.0;}
```

će biti zamjenjen samo jednim pozivom funkcije *shadow2DProj*:

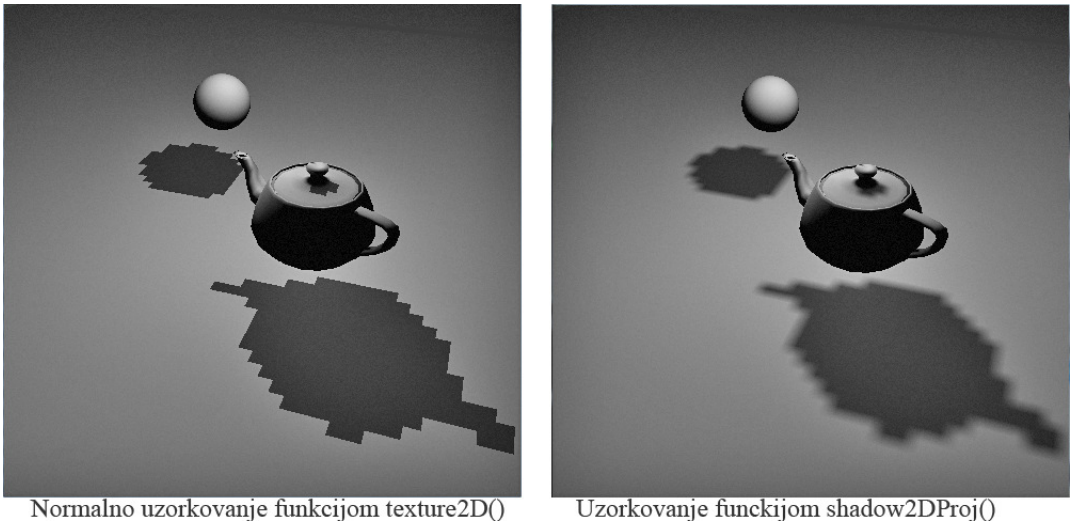
```
float shadow = shadow2DProj(mapaDubine, ShadowCoord).z;
```

Također, više nije potrebno dijeliti vektor koordinata teksture *ShadowCoord* sa komponentom *w*, jer funkcija *shadow2DProj* to čini.

U glavnom programu potrebno je dodati dvije linije u kodu koji pravi teksturu dubinske mape. Cijeli kod stvaranja nove teksture izgleda tada:

```
glGenTextures(1, &depth);

glBindTexture(GL_TEXTURE_2D, depth);

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_DEPTH_TEXTURE_MODE,GL_INTENSITY);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE,
                                        GL_COMPARE_R_TO_TEXTURE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);

glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, širina_mape,
       visina_mape, 0, GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, NULL);
```

Dvije dodane naredbe govore OpenGL-u kako se uspoređuje dubina fragmenta i dubina iz dubinske mape.



Slika 2.8. Prikaz zamućivanja sjene

Na slici 2.8. prikazane su sjene bez korištenja višestrukog uzorkovanja (lijevo) i s višestrukim uzorkovanjem (desno). Izabrana je mala veličina dubinske mape (mape sjena), 64x64 slikovnih elemenata, kako bi se bolje uočila razlika ovih dviju tehnika. Sjene su dosta ugodnije i manje je izražen aliasing, no i dalje se nadzire.

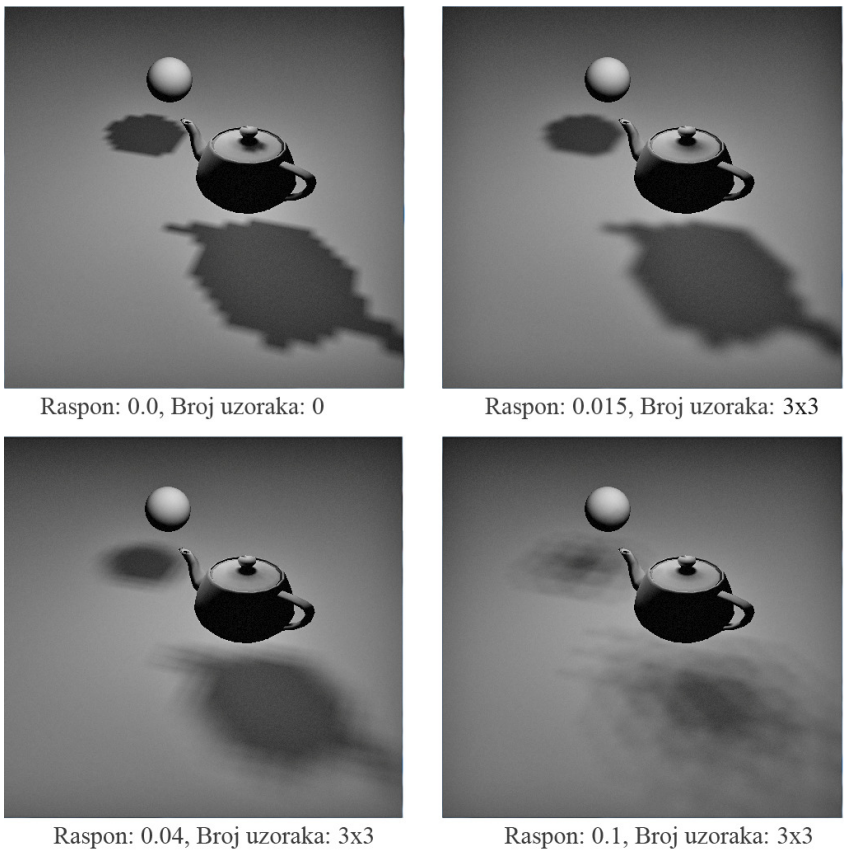
Daljnje poboljšanje je moguće postići višestrukim uzorkovanjem iz programa za sjenčanje fragmenata dodavajući malo odstupanje od koordinata teksture *ShadowCoord*. Razlika između višestrukog uzorkovanja u funkciji *shadow2DProj* i višestrukog uzorkovanja iz programa za sjenčanje fragmenata je u tome što *shadow2DProj* uzorkuje susjedne elemente dubinske mape, dok će se iz programa uzorkovati projektirana sjena. U programu za sjenčanje fragmenata je potrebno izmjeniti jedno uzorkovanje funkcijom

```
float shadow = shadow2DProj(mapaDubine, ShadowCoord).z;
```

s više uzorkovanja sa malim odstupanjem:

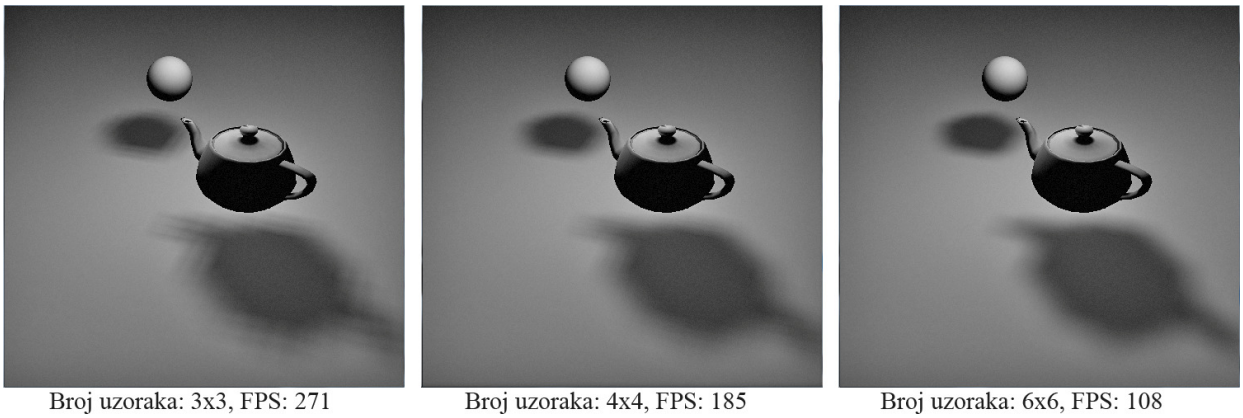
```
for(d.x = -RASPON; d.x <= RASPON; d.x +=2*RASPON/BR_UZORAKA){
      for(d.y = -RASPON; d.y <= RASPON; d.y +=2*RASPON/BR_UZORAKA){

            shadow += shadow2DProj(mapaDubine, ShadowCoord + d).z;
            count++;
      }
}
shadow /= count;
```

Raspon ne smije biti velika vrijednost, jer se javljaju pojasevi (engl. **banding**) (prikazan na predzadnjem i zadnjem primjeru na slici 2.9). Dobra vrijednost za raspon ovisi i o kutu pogleda svjetla, jer će se sa većim kutem javljati i izraženiji banding. Na slici 9. su prikazani primjeri za različite vrijednosti raspona, i uz korištenje *shadow2DProj* funkcije.



Slika 2.9. Utjecaj raspona na prikaz sjene

Veći broj uzoraka smanjuje banding i uvelike povećava kvalitetu sjene, no narušava brzinu iscrtavanja jer je povećano uzorkovanje dubinske mape.



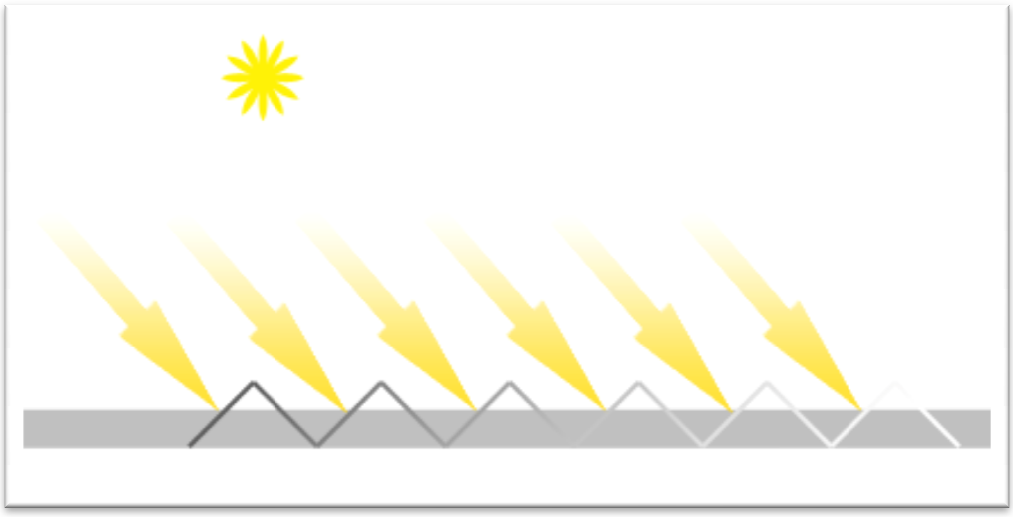
Slika 2.10. Utjecaj broja uzoraka na prikaz sjene

Na slici 2.10. prikazano je povećavanje broja uzoraka uz smanjenje broja sličica po sekundi. Raspon za sva tri primjera iznosi 0.04. Veličina dubinske mape je 64x64 slikovna elementa, i uz raspon od 0.04 i broj uzoraka 6x6 uspješno je prikrivena pogreška uzorkovanja, no na štetu performansi.

Pažljivim izborom parametara moguće je postići kvalitetan izgled sjena, te u potpunosti izbjeći perspektivna pogreška uzorkovanja, ali pri tome se treba paziti i na brzinu iscrtavanja. Za iscrtavanje slika u realnom vremenu često je potrebno i narušiti kvalitetu slike kako bi se dobilo na brzini.

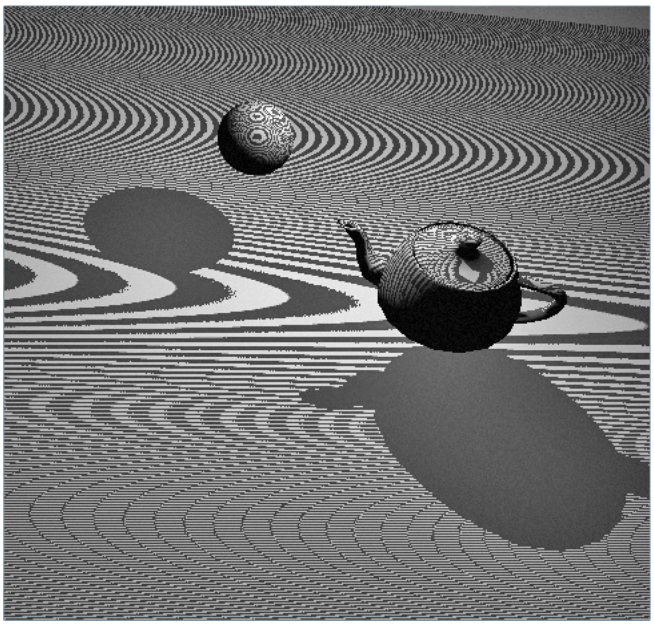
Akne sjena (shadow acne)

Akne sjena se javljaju kada udaljenost od svjetla elementa dubinske mape i odgovarajućeg elementa iz scene nisu jednake nego približne. Zbog ograničene preciznosti z-spremnika te diskretizacije dubine za područje koje pokriva element, vrijednost udaljenosti može odstupati od stvarne udaljenosti, te se za taj element s greškom utvrđuje je li u sjeni.



Slika 2.11. Diskretizacija dubine

Na slici 2.11. prikazana je diskretizacija dubine u dubinskoj mapi. Slika je temeljena na izvoru (Tutorial 16 Shadow mapping, 2013). Vodoravnom linijom je prikazana površina čija će se udaljenost od svjetla iscrtati u dubinsku mapu, a izlomljena linija prikazuje vrijednosti u dubinskoj mapi nakon iscrtavanja. U područjima gdje izlomljena linija prelazi površinu javljaju se akne sjena, jer je na tim mjestima vrijednost u dubinskoj mapi nešto



Slika 2.12. Akne sjena

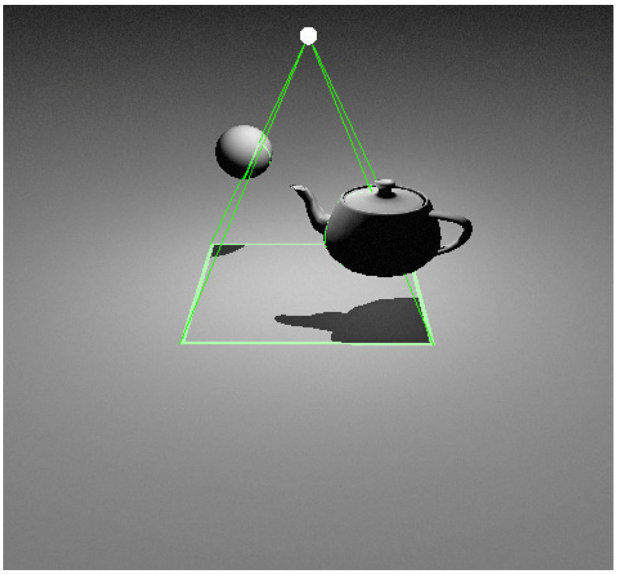
manja od stvarne udaljenosti od svjetla.

Jednostavno rješenje je da se pri usporedbi vrijednosti iz dubinske mape i stvarne udaljenosti doda malo odstupanje, čime se zapravo pomiče udaljenost dubinske mape od izvora svjetla. Vrijednost odstupanja treba biti malog iznosa, jer u protivnom sjene mogu biti odmaknute od objekta (što se zove **peter panning**) (Meyer, 2012). Ovo rješenje uklanja akne s ravnih površina, no na zakrivljenim će se one i dalje pojavljivati.

Slika 2.13. Akne sjena na zakrivljenim površinama.

Na zakrivljenim površinama razlika između stvarne udaljenosti i iz dubinske mape je veća od razlike na ravnim površinama, te je zbog toga potrebno uvesti i veće odstupanje. No uvođenjem prevelikog odstupanja sjene postaju sve odmaknutije od objekta koji ih stvara, te u određenim uvjetima mogu i nestati. Zbog toga dodavanje većeg odstupanja nije poželjno. Uklanjanje akni sa zakrivljenih površina može se postići tako da se u dubinsku mapu iscrtavaju samo oni poligoni objekta koji su okrenuti od svjetla (Sanglard, ShadowMapping with GLSL), tj. kojima je normala okrenuta u smjeru u kojem je i vektor od položaja svjetla prema gledištu. U OpenGL-u ovo se može postići sa *glCullFace(GL_FRONT)*, čime će se iscrtavati samo stražnji poligoni objekta, a prednji će biti odbačeni. Akne će se i dalje pojavljivati, no sada na stražnjim poligonima koji su okrenuti od svjetla te su već u sjeni.

Veliki nedostatak tehnike mape sjena je što su sjene ograničene na područje koje se vidi iz pogleda svjetla. Objekti koji se nalaze izvan pogleda svjetla neće bacati sjenu.

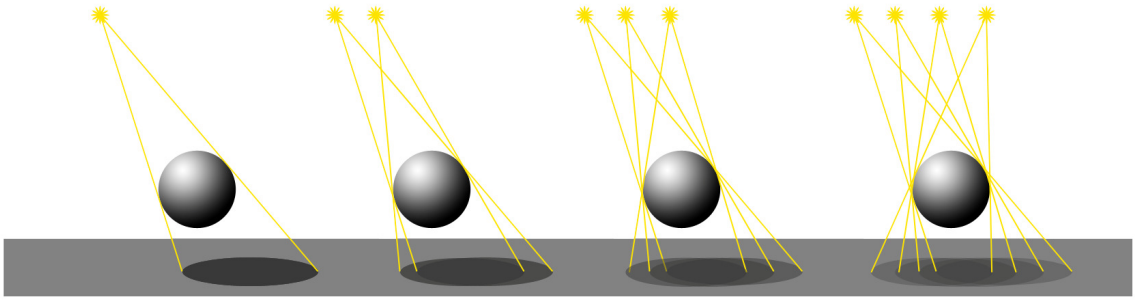


Slika 2.14. Područje koje se vidi iz pogleda svjetla.

Rješenje ovog nedostatka je moguće generiranjem 6 dubinskih mapa iz 6 različitih pogleda koje tvore jednu prostornu mapu nazvanu **cubemap**. Za svaku je potrebno imati posebnu matricu koja transformira točke objekta u sceni u koordinate teksture dubinske mape. Odgovarajuća dubinska mapa tada se može odrediti kao ona za koju su vrijednosti transformiranih x i y koordinata teksture u intervalu [0.0, 1.0]. Za sve ostale mape, transformacija pripadajućim im matricama će davati vrijednosti koordinata teksture izvan tog intervala.

Postizanje efekta mekih sjena

Meke sjene nastaju kada izvor svjetla nije točkast, već ima dimenzije širine i visine. Takav izvor svjetla može se zamisliti kao skupina mnoštva točkastih izvora koji su raspoređeni po površini svjetla. Svaki takav izvor daje objektima u sceni nešto drugačiju sjenu od ostalih izvora. Sve takve sjene se međusobno preklapaju i u konačnici daju jednu meku sjenu. Tamo gdje se najviše sjena iz točkastih izvora preklapa, meka sjena je najjača, za razliku gdje ih se preklapa manji broj ili gdje se sjene ne preklapaju. Na slici 2.15. je prikazano stvaranje meke sjene iz više točkastih izvora.



Slika 2.15. Stvaranje meke sjene

Postizanje stvarnih mekih sjena metodom mapa sjena u realnom vremenu je računski zahtjevno i traži generiranje više dubinskih mapa. Za svjetlo koje je oblika pravokutnika, potrebne su četiri dubinske mape, generirane iz svakog ugla pravokutnika. Iz svake dubinske mape se generira zasebna sjena te ih je potrebno iscrtati u zaseban spremnik, u kojemu se interpoliraju dobivši tako meki prijelaz, te se naposljetku iscrtavaju u spremnik slike.

Drugim pristupom je moguće brže postizanje efekta mekih sjena, koji ne uzima u obzir više svjetlosnih izvora. Pristup se temelji na višestrukom uzorkovanju sjena kao kod tehnike zamućivanja ruba sjene, s time da se raspon uzorkovanja povećava sa većom udaljenosti objekta koji baca sjenu od površine koja ju prima. U glavnom programu potrebno je uvesti nekoliko izmjena. Potrebno je generirati još jednu teksturu, sa funkcijiom *glGenTextures*, koja će zapravo služiti kao kopija dubinske mape. Zatim u dijelu gdje se spremaju ViewMatrix i ProjectionMatrix matrice u matricu Texture dodati dvije linije koje kopiraju dubinsku mapu u novu teksturu.

```
glMatrixMode(GL_TEXTURE);
glActiveTextureARB(GL_TEXTURE7);

      glLoadIdentity();
      glLoadMatrixf(SMatrix);
      glMultMatrixf(SvjetloProjMatrix);
      glMultMatrixf(SvjetloViewMatrix);

      glMatrixMode(GL_MODELVIEW);

glBindTexture(GL_TEXTURE_2D,depth_texture_copy);
glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, 0, 0,
                                  visina_mape, širina_mape, 0);
glBindTexture(GL_TEXTURE_2D,0);

glBindFrameBufferEXT(GL_FRAMEBUFFER_EXT,0);
```

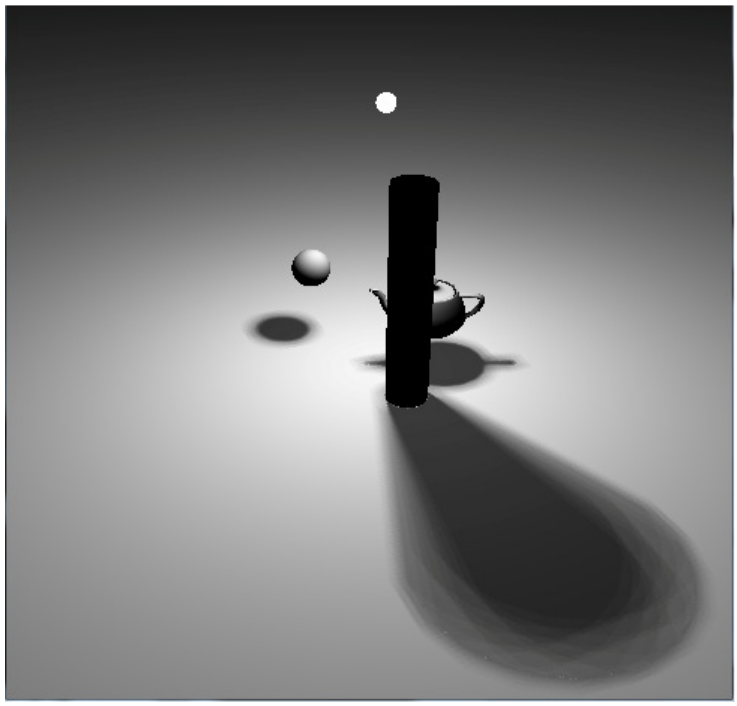
Dodane naredbe su *glBindTexture* i *glCopyTexImage2D*. One aktiviraju teksturu *depth_texture_copy* i kopiraju u nju dubinsku komponentu (GL_DEPTH_COMPONENT) aktivnog spremnika okvira (framebuffera). Potrebno je još predati tu teksturu programu za sjenčanje u dijelu koda koji predaje i matricu *Texture* te dubinsku mapu:

```
int uniView = glGetUniformLocationARB(shader_program,"ViewMatrix");
glUniformMatrix4fvARB(uniView, 1, 0, KameraViewMatrix);

int uniMapa = glGetUniformLocationARB(shader_program,"mapaDubine");
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, depth);
glUniform1iARB(uniMapa, 1);

int uniKopija =
        glGetUniformLocationARB(shader_program,"mapaDubine_copy");
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, depth_texture_copy);
glUniform1iARB(uniKopija, 1);

glUseProgramObjectARB(shader_program);
```

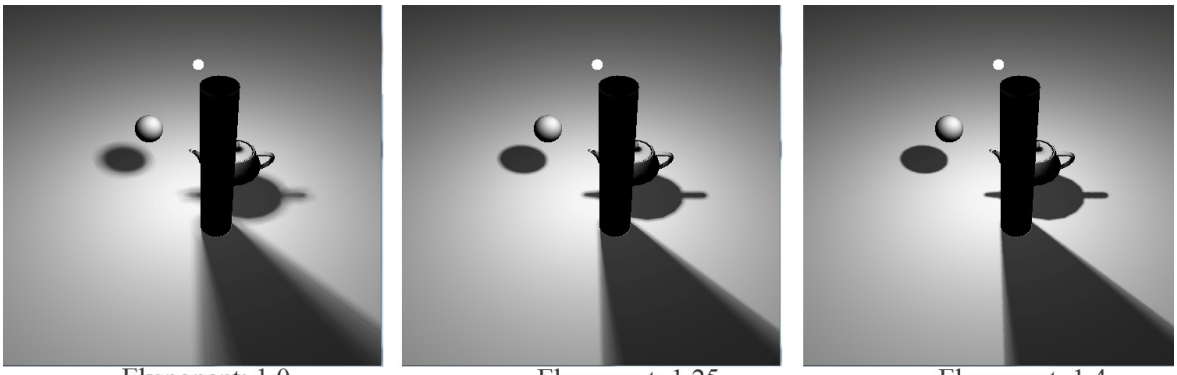
U programu za sjenčanje fragmenata dodaje se dio koda koji izračunava raspon na temelju udaljenosti od elementa iz kopije dubinske mape koja je predna programu za sjenčanje i trenutnog fragmenta koji je u sjeni. Kopija dubinske mape *mapaDubine_copy* je tipa *sampler2D*.

```
float depth = texture2D(mapaDubine_copy,ShadowMapCoord.xy).z;
RASPON = RASPON * pow(abs(depth - ShadowMapCoord.z),EKSPONENT) /
                                    ShadowMapCoord.z;
if(RASPON < e) RASPON = e;
```

Novi raspon se računa kao potencija apsolutne udaljenost između dubine iz dubinske mape i dubine fragmenta. Parametar eksponenta utječe na širinu mekog pojasa sjene i to tako da za veće vrijednosti širina pojasa je uža. U slučaju da je vrijednost raspona 0, postavlja se na vrijednost odstupanja *e,* kako nebi došlo do grešaka.



Slika 2.17 Efekt meke sjene za eksponent vrijednosti 1.0

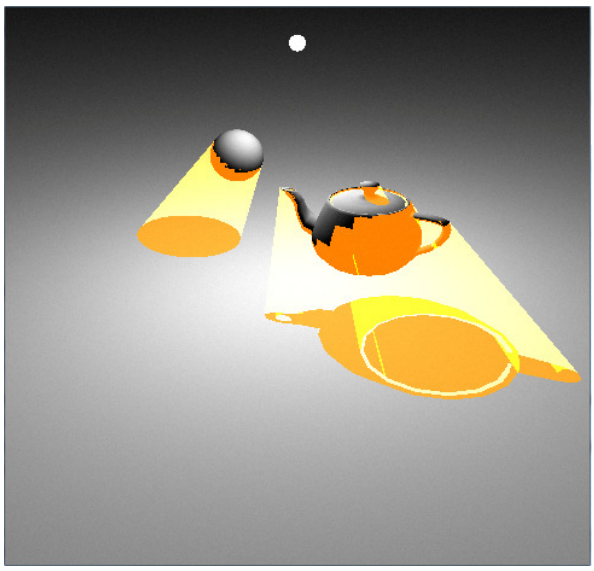


Slika 2.16 Širina pojasa meke sjene za različite vrijednosti eksponenta

3. Volumne sjene (shadow volumes)

Volumne sjene su tehnika prikazivanja sjena korištenjem volumena sjene. Tehniku je osmislio Frank Crow 1977. godine (Crow, 1977).

Tehnika se temelji na ekstenziji siluete objekta viđene iz izvora svjetla i detektiranju jesu li objekti unutar tog volumena. Najprije se iscrtava scena samo u dubinski spremnik. Zatim se određuje silueta objekta kakva je viđena iz pogleda svjetla, te se točke siluete produžuju dalje od svjetla, a volumen koji je pri tome obuhvaćen je volumen sjene. Taj volumen sjene se iscrtava u spremnik šablone (engl. **stencil buffer**) na način da se vrijednost u spremniku šablone povećava pri iscrtavanju prednjeg poligona volumena sjene, a smanjuje pri iscrtavanju stražnjeg poligona. Na taj način se u spremnik šablone iscrtava sjena za koju su u spremniku vrijednosti različite od 0. Zatim se iscrtava scena u spremnik slike za sve one vrijednosti u spremniku šablone koje su jednake 0. Na slici 3.1. prikazana su dva volumena sjene. Volumen sjene počinje od siluete objekta i proteže se u beskonačnost. Površina objekta koju zatvara volumen je u sjeni.



Slika 3.1. Volumen sjene

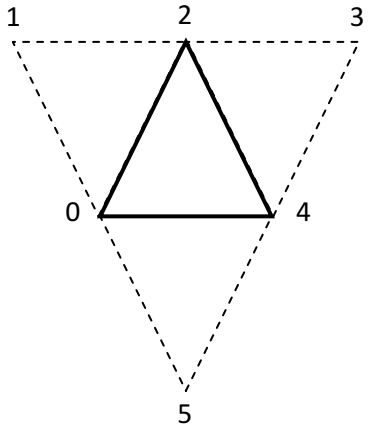
3.1.Implementacija volumnih sjena

Najprije je potrebno iscrtati čitavu scenu u spremnik dubine. Jednom kad je scena iscrtana, mogu se iscrtati volumeni sjene u spremnik šablone. Iscrtavanje scene u spremnik dubine je jednostavno, a pri tome nije potrebno scenu iscrtavati u spremnik slike. Također, i za volumne sjene je potrebno dobaviti matricu pogleda kamere koja se predaje programu za sjenčanje.

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST);
glDepthMask(GL_TRUE);
glDrawBuffer(GL_NONE);
glColorMask(GL_FALSE,GL_FALSE,GL_FALSE,GL_FALSE);
glViewport(0,0,širina_prozora,visina_prozora);

glMatrixMode(GL_PROJECTION);

      glLoadIdentity();
      gluPerspective(FOV,1.0,Z_CLIP_NEAR,Z_CLIP_FAR);

glMatrixMode(GL_MODELVIEW);

      glLoadIdentity();
      gluLookAt(pozicija_kamere,točka_pogleda, up_vector);
      glGetFloatv(GL_MODELVIEW_MATRIX, KameraViewMatrix);

      glPushMatrix();

            glTranslatef(Objekt.position.x, Objekt.position.y,
                         Objekt.position.z);
            glRotatef(Objekt.rotation.w, Objekt.rotation.x,
                      Objekt.rotation.y, Objekt.rotation.z);

      Render(Objekt, GL_TRIANGLES);

      glPopMatrix();
```

Ovim naredbama se brišu spremnici boje i dubine, te se iscrtavaju objekti samo u spremnik dubine. Sada je potrebno iscrtati volumene sjena. Volumen je potrebno generirati pomoću programa za sjenčanje geometrije (**geometry shader**). Program za sjenčanje geometrije prihvaća primitive (točke, linije, trokute) te omogućuje generiranje novih točaka, linija i trokuta. Tim programom će se na temelju ulaznih točaka nekog objekta i pozicije svjetla odrediti silueta i generirati volumen sjene.

Kako bi se programom za sjenčanje geometrije mogao generirati volumen sjene potrebno mu je predati indekse vrhova (**vertex indexes**) svih trokuta, zajedno s indeksima vrhova njima susjednih trokuta. Budući da svaki trokut ima tri susjedna trokuta s kojima dijeli po dva vrha, znači da će se programu za svaki trokut predati tri indeksa vrha tog trokuta zajedno s još tri indeksa vrha susjednih trokuta. Na slici 3.2 (Stich, Wächter, & Keller) je prikazan središnji trokut kojeg sačinjavaju vrhovi 0, 2 i 4, te njemu susjedni trokuti. Po dva vrha susjednih trokutova dijele se s središnjim trokutom te zajedno sa vrhovima 1, 3 i 5 čine susjedne trokute. Prije slanja indeksa vrhova objekta potrebno je posložiti njihov redoslijed tako da sadržavaju vrhove središnjeg trokuta i vrhove susjednih trokuta.



Slika 3.2. Trokut i njemu susjedni trokuti.

Redosljed indeksa vrhova tada bi redom naizmjenično trebao sadržavati najprije indeks jednog vrha središnjeg trokuta, a zatim indeks vrha susjednog trokuta. Za primjer trokuta na slici 3.2, indeksi su redom [0,1,2,3,4,5]. Za čitav objekt indeksi vrhova se u OpenGL na iscrtavanje mogu predati u polju u kojemu su na parnim pozicijama indeksi vrhova središnjih trokuta, a na neparnim pozicijama indeksi vrhova susjednih trokuta. OpenGL-u se polje indeksa predaje u funkciji *glDrawElements*, dok se prije poziva te funkcije, funkcijama *glVertexPointer*, *glNormalPointer*, *glTexCoordPointer* predaju polja točaka, normala i koordinata teksture respektivno.

Čitava funkcija koja iscrtava objekt tada bi izgledala:

```
void Render(Model objekt, uint RenderMode){

      glEnableClientState(GL_VERTEX_ARRAY);
      glEnableClientState(GL_NORMAL_ARRAY);
      glEnableClientState(GL_TEXTURE_COORD_ARRAY);

      glVertexPointer(3,GL_FLOAT,0,objekt.VertexBuffer);
      glNormalPointer(GL_FLOAT,0,objekt.NormalBuffer);
      glTexCoordPointer(2,GL_FLOAT,0,objekt.TexCoordBuffer);

      glDrawElements(RenderMode,objekt.BrVertexIndexes,
                  GL_UNSIGNED_INT,objekt.VertexIndexBuffer);

      glDisableClientState(GL_VERTEX_ARRAY);
      glDisableClientState(GL_NORMAL_ARRAY);
      glDisableClientState(GL_TEXTURE_COORD_ARRAY);

      }
```

Varijabla *RenderMode* za iscrtavanje indeksa vrhova sa susjednim indeksima treba biti postavljena na GL_TRIANGLES_ADJACENCY.

Izvorni kodovi programa za sjenčanje se, kao i kod mapa sjena, učitavaju iz datoteka. Kodovi se zatim prevode i povezuju u program za sjenčanje. Za iscrtavanje volumena sjena potrebni su izvorni kodovi za sjenčanje vrhova i za sjenčanje geometrije.

```
char* VertexShaderKod, *GeometryShaderKod;
/* učitavanje izvornog koda iz datoteka */

int    vertex    =    glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);
glShaderSourceARB(vertex,1, VertexShaderKod,&duzina_vertex);
glCompileShaderARB(vertex);

int geometry = glCreateShaderObjectARB(GL_GEOMETRY_SHADER_ARB);
glShaderSourceARB(geometry,1,GeometryShaderKod,&duzina_geometry);
glCompileShaderARB(geometry);

int shader_program = glCreateProgramObjectARB();
glAttachObjectARB(shader_program,vertex);
glAttachObjectARB(shader_program,geometry);
glLinkProgramARB(shader_program);
```

Ove naredbe je potrebno izvršiti jednom, pri pokretanju programa, te se dalje *shader_program* može koristiti naredbom *glUseProgramObjectARB* nakon što je program za sjenčanje preveden.

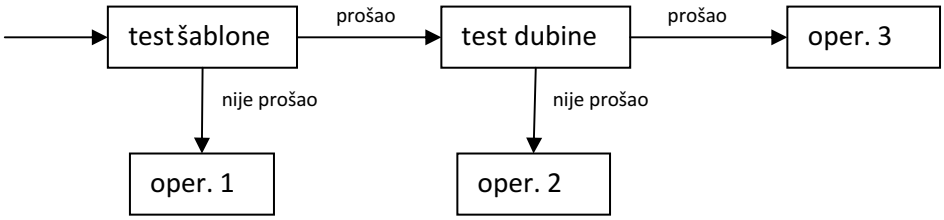
Do sada je u dubinskom spremniku iscrtana scena. Sada će se pomoću programa za sjenčanje iscrtavati volumeni sjena, no pri tome će se mijenjati vrijednosti samo u spremniku šablone, dok se volumeni sjena neće iscrtavati u dubinski spremnik.

```
glEnable(GL_DEPTH_CLAMP);
glEnable(GL_STENCIL_TEST);

glDrawBuffer(GL_NONE);
glDepthMask(GL_FALSE);
glDisable(GL_CULL_FACE);

glStencilFunc(GL_ALWAYS,0x00,0xff);
glStencilOpSeparate(GL_BACK, GL_KEEP, GL_INCR_WRAP, GL_KEEP);
glStencilOpSeparate(GL_FRONT, GL_KEEP, GL_DECR_WRAP, GL_KEEP);
```

Naredbom *glEnable*(GL_STENCIL_TEST) je omogućeno korištenje testa šablone, te se s *glStencilFunc*(GL_ALWAYS,0x00,0xff) postavlja da test elemenata spremnika šablone uvijek prođe (GL_ALWAYS) a druga dva parametra su referentni broj i maska. Test se zapravo sastoji od dva testa. Prvi je test elemenata spremnika šablone po izrazu:

```
(ref_br & maska) USPOREDBA (element & maska)
```
(jed. 1.)

Pri tome USPOREDBA može biti jedna od: GL_ALWAYS, GL_NEVER, GL_EQUAL, GL_NOTEQUAL, GL_LESS, GL_LEQUAL, GL_GEQUAL, GL_GREATER. Drugi dio testa je test elementa spremnika dubine s elementom koji se iscrtava. Funkcija *glStencilOpSeparate* postavlja operacije koje će se poduzeti nad elementom spremnika šablone u slučaju da testovi prođu. Prvi parametar *glStencilOpSeparate* funkcije služi kako bi se postavile različite operacije za prednje poligone objekta (GL_FRONT) i za stražnje poligone (GL_BACK). Drugi parametar postavlja prvu operaciju koja će se provesti nad elementom spremnika šablone u slučaju da test šablone ne prođe, treći parametar postavlja drugu operaciju za slučaj kada test šablone prođe, no test na dubinu ne prođe, a četvrti parametar je za slučaj kada i test na dubinu i test šablone prođu, te se ujedno tada i iscrtava element u spremnik slike. Operacija koja se može obaviti nad elementom spremnika šablone je da se elementu ne promijeni vrijednost (GL_KEEP), da mu se poveća vrijednost (GL_INCR, GL_INCR_WRAP), i da se smanji vrijednost (GL_DECR, GL_DECR_WRAP), postaviti na nulu (GL_ZERO), zamijeniti vrijednost (GL_REPLACE), i invertirati vrijednost (GL_INVERT).

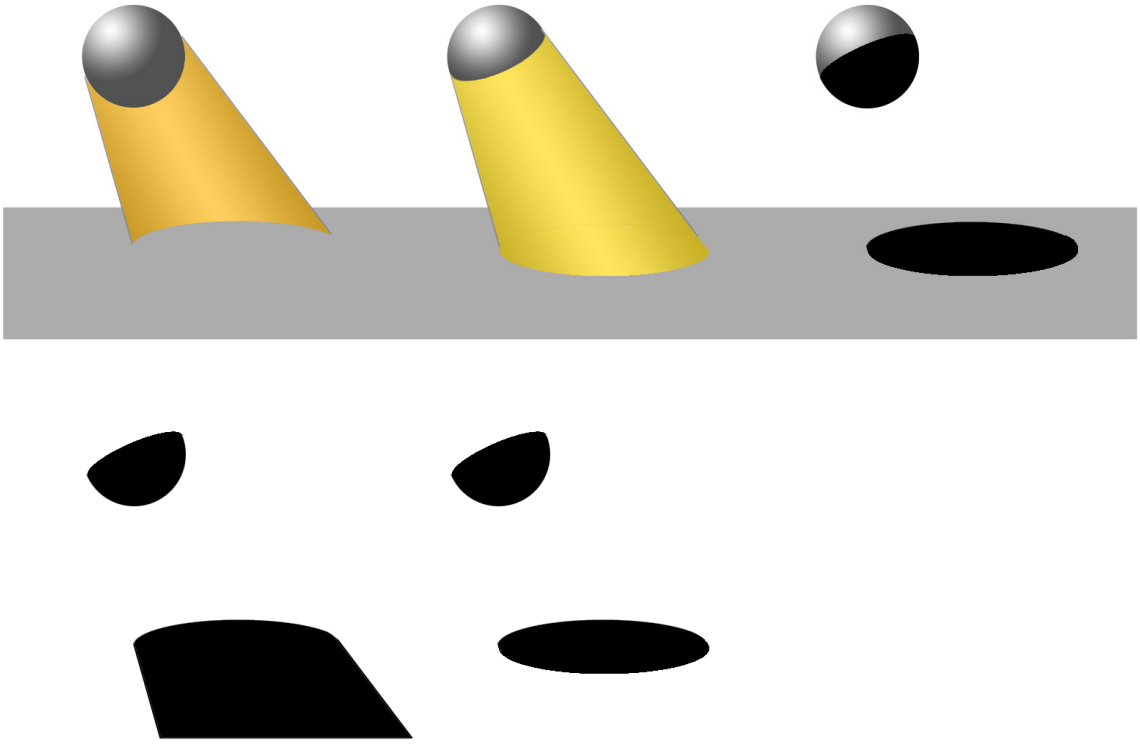


Dijagram 3.1. Test šablone

Na dijagramu 3.1 je prikazan test šablone. Operacije 1, 2 i 3 odgovaraju onim operacijama koje su postavljene funkcijom *glStencilOpSeparate*, tj. u drugom trećem i četvrtom parametru respektivno.

Za iscrtavanje volumnih sjena operacije su postavljene tako da u slučaju kada se iscrtava stražnji poligon i test dubine nije prošao, vrijednost elementa spremnika šablone se povećava, a kada se iscrtava prednji poligon i test dubine nije prošao vrijednost elementa spremnika šablone se smanjuje. Ovime se postiže da u spremniku šablone oni elementi koji su u sjeni imaju vrijednost različitu od 0, a oni koji nisu u sjeni imaju vrijednost jednaku 0. Ova tehnika je poznata kao z-fail tj. kao Carmack's Reverse.

Postoji također i tehnika z-pass koja je nešto starija od tehnike z-fail. Kod nje umjesto mijenjanja vrijednosti elementa spremnika šablone u slučaju da test dubine nije prošao, vrijednost elementa se mijenja kad je test dubine prošao. Rezultat je isti kao i kod z-fail tehnike, no ova tehnika ima nedostatak u slučaju kada se kamera nalazi unutar volumena sjene pa se tada sjene invertiraju, odnosno oni elementi slike koji su u sjeni sada su na svjetlu, a oni koji su bili na svjetlu sada su u sjeni. Ovo se događa zbog toga što je kamera unutar volumena sjene, najbliži su joj stražnji poligoni tog volumena, te se za taj slučaj vrijednosti u spremniku šablone povećavaju, a budući da kamera ne vidi prednje poligone volumena, vrijednosti se neće smanjiti. Drugi volumeni sjena će i dalje povećavati i smanjivati vrijednosti u spremniku šablone, no vrijednosti spremnika šablone su od početka bile uvećane, te dolazi do inverzije sjena. Zbog toga je tehnika z-fail bolja, jer se kod nje vrijednosti spremnika šablone mijenjaju samo na mjestima gdje se nešto nalazi ispred volumena sjene, a problem kada je kamera unutar volumena ne utječe na sjene.



Slika 3.3. Prikaz iscrtavanja volumena sjene u spremnik šablone, te krajnji rezultat

Na slici 3.3 je prikazano iscrtavanje volumena sjene u spremnik šablone, upotrebom tehnike z-fail. Gornji red slika prikazuje s lijeva na desno: iscrtavanje stražnjih poligona volumena sjene, zatim iscrtavanje prednjih poligona, te naposljetku rezultat slike sa sjenama. Donji red prikazuje stanje u spremniku šablone. Pri iscrtavanju prednjih poligona volumena sjene tehnikom z-fail, elementima u spremniku šablone se povećava vrijednost za one elemente koji nisu prošli test dubine. Kada se iscrtavaju prednji poligoni volumena sjene, elementima koji nisu prošli test dubine se u spremniku šablone smanjuje vrijednost. Tada ostaju s promijenjenom vrijednosti oni elementi koji su zapravo u sjeni. Naposljetku se postavi iscrtavanje slike tako da se iscrtavaju oni elementi slike za koje vrijednosti u spremniku šablone nisu promijenjene. Važno je napomenuti da u ovom primjeru volumen sjene nije zatvoren te se zbog toga pojavljuje sjena i na objektu koji generira tu sjenu (u ovom primjeru na kugli). Jednostavno rješenje ovog problema jest da se volumen sjene u potpunosti zatvori.

Nakon što su se postavile operacije nad spremnikom šablone, iscrtavaju se volumeni sjena upotrebom programa za sjenčanje.

```
glUseProgramObjectARB(shader_program);

int uniView = glGetUniformLocationARB(shader_program,"ViewMatrix");
glUniformMatrix4fvARB(uniView, 1, 0, KameraViewMatrix);

glLightfv(0, GL_POSITION, pozicija_svjetla);

glMatrixMode(GL_MODELVIEW);

      glPushMatrix();
            glTranslatef(Objekt.position.x, Objekt.position.y,
                        Objekt.position.z);
            glRotatef(Objekt.rotation.w, Objekt.rotation.x,
                        Objekt.rotation.y, Objekt.rotation.z);

      Render(Objekt, GL_TRIANGLES_ADJACENCY);

      glPopMatrix();

glUseProgramObjectARB(0);

glEnable(GL_CULL_FACE);
glDisable(GL_DEPTH_CLAMP);
glDepthMask(GL_TRUE);
glDrawBuffer(GL_BACK);
```

Programu za sjenčanje predaje se matrica pogleda kamere koja je prethodno dobavljena. Postavlja se pozicija svjetla naredbom *glLightfv*, te se iscrtavaju objekti načinom za iscrtavanje GL_TRIANGLES_ADJACENCY. Naposljetku se omogućava iscrtavanje u spremnik za dubinu i spremnik slike. Spremnik šablone je sada spreman za korištenje.

Sada se iscrtava scena sa sjenama u spremnik slike. Potrebno je izbrisati dubinski spremnik i uključiti korištenje spremnika šablone. Prvo se iscrtava osvjetljeni dio scene, a zatim onaj dio koji je u sjeni. Osvijetljeni dio scene potrebno je iscrtati s jačim osvjetljenjem od dijela koji je u sjeni.

```
glClear(GL_DEPTH_BUFFER_BIT);

glEnable(GL_STENCIL_TEST);
glStencilOpSeparate(GL_FRONT, GL_KEEP, GL_KEEP, GL_KEEP);
glStencilFunc(GL_EQUAL, 0x00, 0xff);

glEnable(GL_LIGHTING);
RenderLight(0, pozicija_svjetla, jačina_svjetla);
glMatrixMode(GL_MODELVIEW);

      glPushMatrix();
            glTranslatef(Objekt.position.x, Objekt.position.y,
                        Objekt.position.z);
            glRotatef(Objekt.rotation.w, Objekt.rotation.x,
                        Objekt.rotation.y, Objekt.rotation.z);

      Render(Objekt, GL_TRIANGLES);

      glPopMatrix();
```

Za sjenu će se jačina svjetla smanjiti. Potrebno je postaviti test šablone da prolazi samo za one elemente koji nisu jednaki nula.

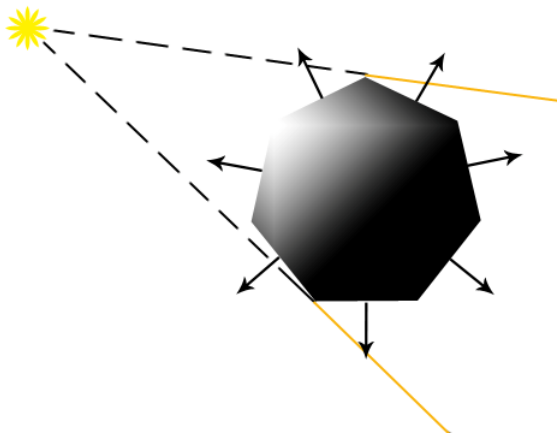
```
glStencilFunc(GL_NOTEQUAL, 0x00, 0xff);

RenderLight(0, pozicija_svjetla, 0.25 * jačina_svjetla);

glMatrixMode(GL_MODELVIEW);

      glPushMatrix();
            glTranslatef(Objekt.position.x, Objekt.position.y,
                        Objekt.position.z);
            glRotatef(Objekt.rotation.w, Objekt.rotation.x,
                        Objekt.rotation.y, Objekt.rotation.z);

      Render(Objekt, GL_TRIANGLES);

      glPopMatrix();

glDisable(GL_STENCIL_TEST);
glutSwapBuffers();
```

Scena sa sjenama je iscrtana. U nastavku će se objasniti dobivanje volumena sjene pomoću programa za sjenčanje.

Program za sjenčanje sadrži program za sjenčanje vrhova i program za sjenčanje geometrije. Program za sjenčanje fragmenata nije potreban. Program za sjenčanje vrhova je jednostavan i služi samo za transformaciju vrhova.

```
uniform mat4 ViewMatrix;

void main(){

      mat4 glModelMatrix = inverse(ViewMatrix) *
                                    gl_ModelViewMatrix;

      gl_Position = glModelMatrix * gl_Vertex;

      }
```

Program za sjenčanje geometrije generira volumen sjene. On prima jedan trokut objekta i vrhove susjednih trokuta. Za ovo je bilo potrebno proširiti polje indeksa da uključi i susjedne vrhove. Ideja generiranja volumena sjene je u tome da se pronađe silueta objekta kakva je viđena iz svjetla. Za izračun siluete potrebne su normale trokuta. Ako je normala središnjeg trokuta okrenuta prema svjetlu, a normala susjednog trokuta nije, tada je brid između ta dva trokuta brid siluete. Na slici 3.4 je prikazan objekt s normalama njegovih



Slika 3.4 Određivanje siluete

trokuta. Gledajući iz svjetla, silueta se nalazi na bridu ona dva trokuta kod kojih je jedna normala okrenuta prema svjetlu, a druga normala je okrenuta od svjetla. Silueta se tada projektira u smjeru od svjetla stvarajući pri tome volumen sjene.

```
#define epsilon 0.0000000001
#define offset 0.001

layout(triangles_adjacency) in;
layout(triangle_strip, max_vertices = 64) out;

uniform mat4 ViewMatrix;
mat4 ViewProjectionMatrix;
vec4 LightPosition;

void EmitirajTrokut(vec3 v);
void EmitirajQuad(vec3 a, vec3 b);


void main(){

      ViewProjectionMatrix = gl_ProjectionMatrix * ViewMatrix;
      LightPosition = inverse(ViewMatrix) *
                                 gl_LightSource[0].position;

      vec3 v0 = gl_in[0].gl_Position.xyz;
      vec3 v1 = gl_in[1].gl_Position.xyz;
      vec3 v2 = gl_in[2].gl_Position.xyz;
      vec3 v3 = gl_in[3].gl_Position.xyz;
      vec3 v4 = gl_in[4].gl_Position.xyz;
      vec3 v5 = gl_in[5].gl_Position.xyz;

      vec3 brid02 = v2 - v0;
      vec3 brid04 = v4 - v0;
      vec3 brid01 = v1 - v0;
      vec3 brid23 = v3 - v2;
      vec3 brid24 = v4 - v2;
      vec3 brid05 = v5 - v0;

      vec3 centar = (v0 + v2 + v4) / 3.0;
      vec3 Normala = normalize(cross(brid02,brid04));
      vec3 LightDir = LightPosition.xyz - centar;
      vec3 nLightDir = normalize(LightDir);
```

U ovom dijelu koda programa za sjenčanje geometrije izračunavaju se pozicija svjetla te bridovi trokuta. Bridovi su potrebni kako bi se pronašle normale trokuta. Varijabla *Normala* sadrži normalu središnjeg trokuta. Također se računa i smjer svjetla iz centra središnjeg trokuta. Kao što je prethodno navedeno, središnji trokut je omeđen vrhovima 0, 2 i 4. Zatim će se smjer svjetla upotrijebiti kako bi se provjerila orijentacija središnjeg trokuta u odnosu na svjetlo. Ako je središnji trokut orijentiran prema svjetlu, provjeravati će se njegovi susjedni trokuti.

```
if( dot(Normala,nLightDir) > epsilon ){

      vec3 normalaT1 = normalize(cross(brid01,brid02));
      vec3 normalaT2 = normalize(cross(brid23,brid24));
      vec3 normalaT3 = normalize(cross(brid04,brid05));

      vec3 lightDirT1 = normalize(LightPosition.xyz -
                                  ((v0+v1+v2)/0.3));
      vec3 lightDirT2 = normalize(LightPosition.xyz -
                                  ((v2+v3+v4)/0.3));
      vec3 lightDirT3 = normalize(LightPosition.xyz -
                                  ((v0+v4+v5)/0.3));

      if( dot(normalaT1,lightDirT1) <= epsilon ){
            EmitirajQuad(v0,v2);}
      if( dot(normalaT2,lightDirT2) <= epsilon ){
            EmitirajQuad(v2,v4);}
      if( dot(normalaT3,lightDirT3) <= epsilon ){
            EmitirajQuad(v4,v0);}
```

Provjerava se orijentacija središnjeg trokuta pomoću njegove normale i smjera svjetla. Ako je skalarni umnožak normale i vektora smjera svjetla veći od 0, odnosno zbog greške od neke male vrijednosti *epsilon*, tada je središnji trokut orijentiran prema svjetlu. Za taj slučaj onda se izračunavaju normale susjednih trokuta, a također i smjer svjetla od njihovih centara. Zatim se provjerava orijentacija svakog susjednog trokuta i, ako je skalarni umnožak normale i vektora smjera svjetla manji od 0, emitira se četverokut (engl. **quad**) funkcijom *EmitirajQuad*. Na kraju je još potrebno zatvoriti volumen sjene, tj emitirati prednji i stražnji poklopac.

```
vec3 lightV0Dir = normalize(v0 - LightPosition.xyz);
vec3 lightV2Dir = normalize(v2 - LightPosition.xyz);
vec3 lightV4Dir = normalize(v4 - LightPosition.xyz);

vec3 mali_pomak = offset *lightV0Dir;
gl_Position = ViewProjectionMatrix * vec4(v0 +
                                  mali_pomak,1.0);
EmitVertex();
mali_pomak = offset *lightV2Dir;
gl_Position = ViewProjectionMatrix * vec4(v2 +
                                  mali_pomak,1.0);
EmitVertex();
mali_pomak = offset *lightV4Dir;
gl_Position = ViewProjectionMatrix * vec4(v4 +
                                  mali_pomak,1.0);
EmitVertex();
EndPrimitive();
```

Prednji poklopac se emitira malim pomakom točaka središnjeg trokuta u smjeru od svjetla za što je potrebno izračunati smjer od točaka trokuta do svjetla.

```
                gl_Position = ViewProjectionMatrix *
                                        vec4(lightV2Dir,0.0);
                EmitVertex();
                gl_Position = ViewProjectionMatrix *
                                        vec4(lightV0Dir,0.0);
                EmitVertex();
                gl_Position = ViewProjectionMatrix *
                                        vec4(lightV4Dir,0.0);
                EmitVertex();
                EndPrimitive();
        }

}
```

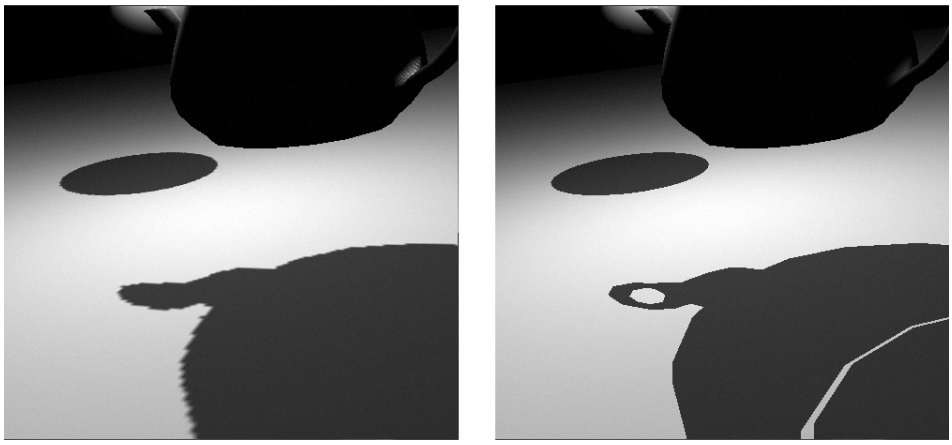
Stražnji poklopac se emitira tako što se homogena komponenta *w* vektora od točke trokuta do svjetla postavi na 0, čime je ona projektirana u beskonačnost. Volumen sjene je tada generiran i zatvoren s prednje i stražnje strane. U nastavku je kod dviju funkcija koje emitiraju četverokut volumena sjene.

```
void EmitirajTrokut(vec3 v){

      vec3 LightDir = normalize(v - LightPosition.xyz);
      vec3 mali_pomak = offset * LightDir;
      gl_Position = ViewProjectionMatrix * vec4(v+mali_pomak,1.0);
      EmitVertex();
      gl_Position = ViewProjectionMatrix * vec4(LightDir,0.0);
      EmitVertex();

}

void EmitirajQuad(vec3 a, vec3 b){

      EmitirajTrokut(a);
      EmitirajTrokut(b);
      EndPrimitive();

}
```

EmitrajQuad prihvaća dvije točke i emitira trokut za svaku točku. *EmitirajTrokut* zapravo emitira brid između točke *v* (sa malim pomakom), i točke u smjeru od svjetla koja je u beskonačnosti. Ponovnim pozivom funkcije *EmitrajTrokut* emitira se i drugi brid četverokuta.

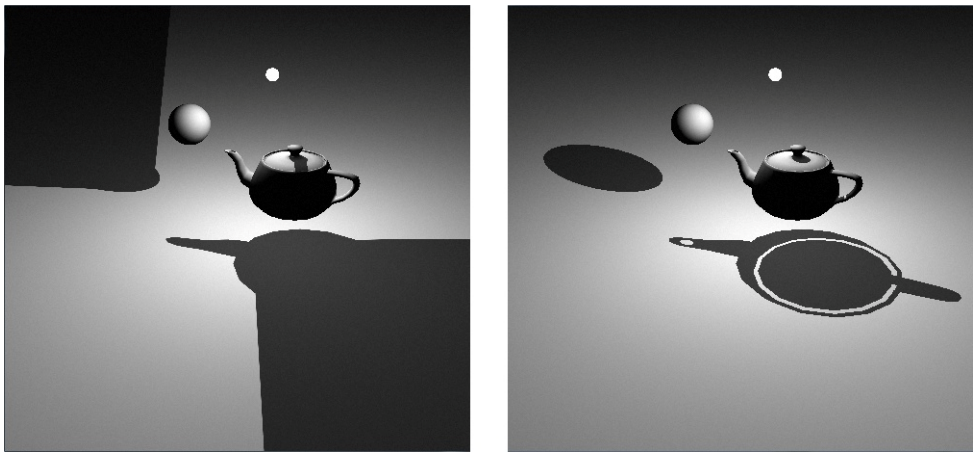
3.2.Prednosti i nedostatci u odnosu na mape sjena

Glavna prednost tehnike volumnih sjena u odnosu na tehniku mapa sjena je sprječavanje pojavljivanja perspektivnog aliasinga i akni sjena. Perspektivni aliasing se javlja kada rezolucija mape sjena (tj. dubinske mape) nije dovoljna te se više elemenata slike preslikava u jedan element mape sjena. Budući da se volumne sjene generiraju za svaki element spremnika šablone, a time i za svaki element slike, perspektivnog aliasinga nema. Akne sjena se u tehnici mapa sjena javljaju zbog nepreciznosti udaljenosti elementa od svjetla dobivene iz pogleda svjetla i udaljenosti dobivene iz pogleda kamere. Kod volumnih sjena test na dubinu se provodi iz istog pogleda, a budući da se volumen sjene proteže u beskonačnost prilikom testiranja na dubinu (bilo z-fail ili z-pass tehnikom), neće doći do grešaka.



Slika 3.5. Aliasing kod mapa sjena (lijevo) i sjene bez aliasinga kod volumnih sjena (desno)

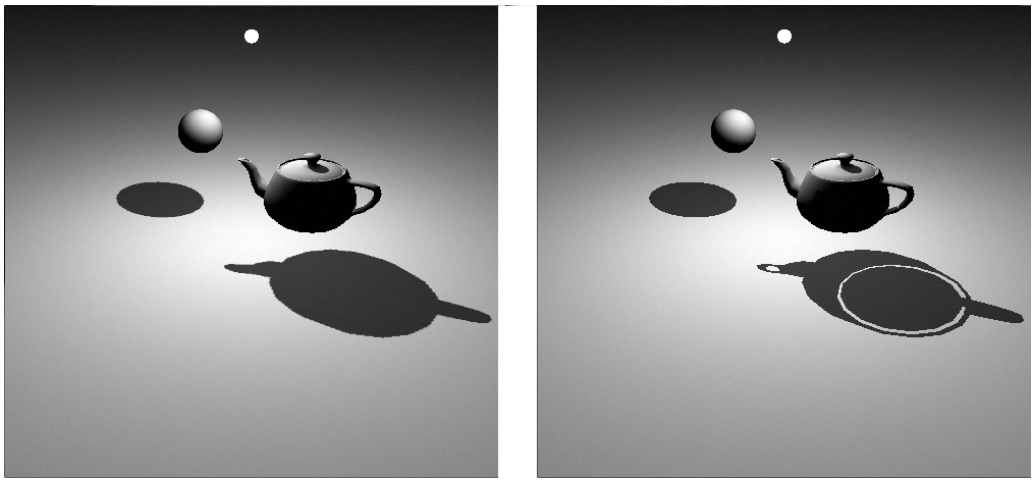
Volumne sjene nisu ograničene na prostor pogleda svjetla kao što je to kod mapa sjena, tj. sjene se mogu generirati za čitav prostor.



Slika 3.6. Ograničenost prikaza mapa sjena (lijevo) i volumne sjene (desno)

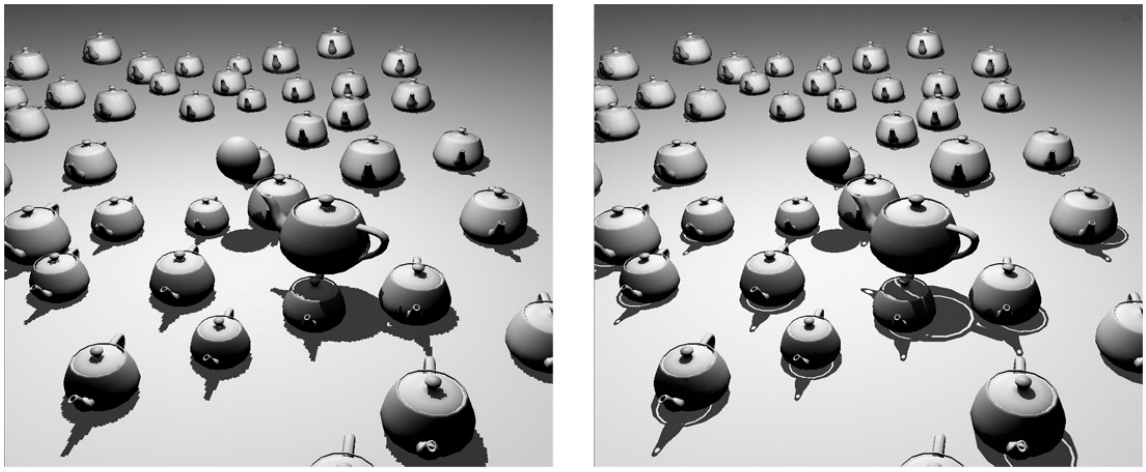
Problem kod volumnih sjena je što nije moguće napraviti točnu sjenu za objekte s transparentnom teksturom. Budući da se volumen sjene generira iz siluete objekta, transparentne teksture ne utječu na oblik siluete. Zbog toga objekti s transparentnim teksturama (npr. lišće, ograda, itd...) neće imati pravilnu sjenu.

Objekti koji bacaju sjenu moraju biti potpuno zatvoreni, što nije slučaj kod mapa sjena kod kojih nije od velike važnosti geometrija objekta. Ako objekt nije u potpunosti zatvoren, za otvoreni dio će se također generirati volumen sjene. Na slici 3.7 prikazan je objekt čajnika koji nije u potpunosti zatvoren, te se za desni primjer iscrtavanja volumnih sjena vidi nepravilna sjena.



Slika 3.7. Sjena za otvoreni objekt (čajnik) kod mapa sjena (lijevo) i kod volumnih sjena (desno)

No, najveći nedostatak volumnih sjena je što je za istu scenu potrebno iscrtati više poligona nego kod mapa sjena. Za volumne sjene su potrebna tri iscrtavanja geometrije, prvi put za iscrtavanje dubinske mape, drugi put za iscrtavanje volumena sjena, te treći put iscrtavanje u spremnik slike. Kod mapa sjena potrebna su dva iscrtavanja, jednom za dubinsku mapu iz pogleda svjetla, te drugi put iz pogleda kamere. Zbog toga je iscrtavanje mape sjena brže od volumnih sjena. Volumne sjene također zahtjevaju generiranje volumena sjena, što dodatno usporava iscrtavanje, posebice ako nisu podržani programi za sjenčanje geometrije, u kojem slučaju se volumeni generiraju na glavnom procesoru.



Mape sjena: 61 FPS Volumne sjene: 22 FPS

Slika 3.8. Brzina iscrtavanja za tehniku mapa sjena (lijevo) i za tehniku volumnih sjena (desno)

Na slici 3.8 prikazano je iscrtavanje složene scene sa velikim brojem poligona. Za dubinsku mapu veličine 2048x2048 slikovna elementa tehnika mapa sjena je približno trostruko brža od tehnike volumnih sjena.

4. Zaključak

U ovom radu opisane su dvije najčešće tehnike iscrtavanja sjena u stvarnom vremenu. Unaprjeđenjem grafičkog sklopovlja, dodavanjem novih mogućnosti i povećanjem protočnosti podataka, omogućeno je izračunavanje sve kompleksnijih algoritama u stvarnom vremenu. Tehnika volumnih sjena tako se može u potpunosti izvoditi na grafičkom sklopovlju, a i neki napredniji načini uzorkovanja poput funkcije *shadow2DProj* u GLSL jeziku povećavaju kvalitetu iscrtanih sjena. Tehnikama mapa sjena i volumnih sjena moguće je postizanje visokog stupnja realizma. Ovo je najviše izraženo u tehnici mapa sjena gdje je moguća veća kontrola pri iscrtavanju sjene primjerice zamućivanjem ruba sjene višestrukim uzorkovanjem te postizanjem efekta mekih sjena. S druge strane volumnim sjenama moguć je precizan prikaz sjene i za najmanje detalje objekta. Zbog toga su ova dva pristupa popularna i često korištena kako u iscrtavanju u stvarnom vremenu primjerice kod video igara tako i u fotorealističnim prikazima korištenima primjerice u filmskoj industriji.

5. Literatura

1. *Cascaded Shadow Maps.* (2012). Preuzeto 12. 6 2013 iz msdn.microsoft.com: http://msdn.microsoft.com/en-us/library/windows/desktop/ee416307%28v=vs.85%29.aspx
2. *Common Techniques to Improve Shadow Depth Maps.* (2012). Preuzeto 10. 6 2013 iz msdn.microsoft.com: http://msdn.microsoft.com/en-us/library/windows/desktop/ee416324%28v=vs.85%29.aspx
3. Crow, F. C. (1977). SHADOW ALGORITHMS FOR COMPUTER GRAPHICS. Austin, Texas, United States of America: University of Texas at Austin.
4. Liland, Ø. (2002). *Projection shadows*. Preuzeto 21. 6 2013 iz www.ia.hiof.no: http://www.ia.hiof.no/~borres/cgraph/explain/shadow/p-shadow.html
5. Meyer, N. (7. 5 2012). *Shader Effects: Shadow Mapping*. Preuzeto 12. 6 2013 iz devmaster: http://devmaster.net/posts/3002/shader-effects-shadow-mapping
6. *OpenGL Shadow Mapping Tutorial - Paul's Projects*. (n.d.). Preuzeto 1 2012 iz Paul's Projects: OpenGL tutorials: http://www.paulsprojects.net/tutorials/smt/smt.html
7. *Projective texture mapping*. (n.d.). Preuzeto 21. 6 2013 iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Projective_texture_mapping
8. Sanglard, F. (n.d.). *ShadowMapping with GLSL*. Preuzeto 1 2012 iz Fabien Sanglard's non-blog: http://fabiensanglard.net/shadowmapping/index.php
9. Sanglard, F. (14. 2 2009). *Soft shadows with PCF*. Preuzeto 1 2012 iz Fabien Sanglard's non-blog: http://fabiensanglard.net/shadowmappingPCF/index.php
10. Stich, M., Wächter, C., & Keller, A. (n.d.). *Chapter 11. Efficient and Robust Shadow Volumes Using Hierarchical Occlusion Culling and Geometry Shaders.* Preuzeto 15. 5 2013 iz GPU Gems: http://http.developer.nvidia.com/GPUGems3/gpugems3_ch11.html
11. *Tutorial 16 Shadow mapping*. (7. 4 2013). Preuzeto 24. 5 2013 iz OpenGL tutorial: http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/
12. Williams, L. (1978). CASTING CURVED SHADOWS ON CURVED SURFACES. *CASTING CURVED SHADOWS ON CURVED SURFACES* . Old Westbury, New York 11568: New York Institute of Technology.

6. Sažetak

(Postupci ostvarivanja sjene)

Sjene su neizostavan dio prilikom postizanja realističnih slika. Zbog toga se razvio veliki broj tehnika koje postižu sjene. Tehnike za postizanje fotorealističnih slika poput algoritma praćenja zrake simuliraju stvarne uvjete nastajanja sjena, no zbog intezivnih računarskih zahtjeva one nisu prikladne za iscrtavanje u stvarnom vremenu. Razvijane su jednostavnije tehnike koje s različitim stupnjem pojednostavnjuju proces nastajanja sjene. U radu su predstavljene dvije tehnike koje se najčešće koriste u prikazivanju sjena u stvarnom vremenu. Tehnika mapa sjena prigodna je za postizanje ugodnih tj. mekih sjena, a s današnjim grafičkim sklopovljem postižu se velike brzine iscrtavanja sjene. Tehnikom volumnih sjena mogući su precizni prikazi sjena i pogodna je za točkaste izvore svjetla kod kojih su izraženi rubovi sjene.

Ključne riječi: sjene, volumne sjene, mape sjena, meke sjene, OpenGL, spremnik dubine, spremnik šablone, iscrtavanje u stvarnom vremenu.

7. Abstract

(Shadowing techniques)

Shadows are an essential detail in acomplishing realistic images. For that reason a great number of techniques has been developed for generating shadows. Fotorealistic techniques like ray-tracing algorithm simulate real world conditions by which shadows are formed, but because of intensive computing demands those techniques are not suitable for rendering in real time. Simpler techniques have been developed which simplify shadow forming process with a different degree. In this document two popular techniques are presented for generating shadows in real time. Shadow mapping technique is suitable for smooth, soft shadows, and with today's graphics hardware capabilities great frame rates are achived. Shadow volume technique displays precise shadows, and is suitable for point lights with pronounced shadow edges.

Keywords: shadows, shadow volumes, shadow mapping, soft shadows, OpenGL, depth buffer, stencil buffer, real-time rendering.