SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 341

POSTUPCI TEKSTURIRANJA UPOTREBOM GRAFIČKOG PROCESORA

Luka Piljek

Zagreb, lipanj 2008.

Sadržaj

Uvod

U računalnoj grafici riječ tekstura (eng. *texture map*) odnosi se u užem smislu na sliku kojom se opisuje površina nekog materijala. Tekstura u širem smislu može biti bilo kakav skup (polje) podataka čiji se elementi (eng. *texel*) pridružuju površini nekog lika pri iscrtavanju i predstavljaju neka obilježja te površine.

Tekstura tako može biti jednodimenzionalna (jedan red *texela*), dvodimezionalna (npr. slika) ili trodimenzionalna (volumna tekstura).

Postupak preslikavanja teksture na površinu virtualnog objekta zove se teksturiranje (eng. *texture mapping*). Preslikavanje (eng. *mapping*) se obavlja tako da se pojedinim vrhovima (eng. *vertex*) poligona dodijele koordinate koje odgovaraju položaju pridruženog elementa teksture u polju teksture. Ovo preslikavanje može se izvoditi na različite načine, a koordinate ne moraju biti statične tj. mogu se mijenjati ovisno o pogledu, vremenu ili na neki drugi način.

Osim samog preslikavanja teksture, vrlo je bitno i što se sve može napraviti s dobivenim podatkom. Najčešća primjena teksture je za određivanje boje materijala u nekoj točki. Međutim tekstura može predstavljati i bilo koje drugo obilježje materijala koje želimo prikazati. Tako na primjer teksturom možemo određivati sjajnost materijala, hrapavost materijala i čak dobiti izgled detaljnih neravnina na inače ravnoj površini.

Napretkom grafičkog sklopovlja postalo je popularno koristiti više tekstura odjednom pri čemu svaka opisuje neko različito obilježje površine na koju se preslikava te se tako dobiju vrlo uvjerljive simulacije stvarnih materijala. Taj se postupak naziva višestruko teksturiranje (eng. *multitexturing*) i podržavaju ga sve modernije grafičke kartice.

Uporabom programirljivih sklopova u grafičkom protočnom sustavu (jedinice za sjenčanje, eng. *Shader Unit*) programer može sam odrediti kako će koja tekstura utjecati na izgled virtualnog modela koji se prikazuje.

U ovom će radu biti obrađeno nekoliko postupaka preslikavanja tekstura, višestruko teksturiranje te nekoliko zanimljivih učinaka koji se mogu dobiti primjenom tih postupaka.

Svi će primjeri biti ostvareni na programirljivim grafičkim procesorima te napisani u programskom jeziku GLSL koji će također biti pobliže opisan u prvom poglavlju.

1. Programski jezik GLSL

Programski jezik GLSL (eng. *OpenGL Shading Language*) je jezik visokog nivoa namijenjen isključivo programima za procesor vrhova i procesor fragmenata.

Specifikacija GLSL-a definirana je od strane ARB odbora (eng. *Architectural Review Board*) koji je zadužen i za sve odredbe u vezi OpenGL-a.

Jezik GLSL omogućuje da se na jednostavan način, koristeći poznatu sintaksu programskog jezika C, isprogramira funkcionalnost grafičkog procesora tj. jedinica za sjenčanje u grafičkom protočnom sustavu. Na taj se način daje sloboda programeru da utječe na obradu podataka u grafičkom protočnom sustavu i ostvari učinke koji inače ne bi bili mogući na unaprijed ugrađenoj (fiksnoj) funkcionalnosti grafičke kartice.

Osim GLSL-a postoji još nekoliko jezika koji pružaju slične mogućnosti, npr. Cg koji je razvila *nVidia* za svoje grafičke kartice ili *Microsoftov* HLSL koji radi s *Direct3D* API-jem.

GLSL je ovdje odabran zbog njegove potpune kompatibilnosti s *OpenGL* API-jem koji je korišten za izradu aplikacije koja prikazuje rezultate ovog rada.

.

1.1. Sintaksa

Sintaksa GLSL-a vrlo je slična sintaksi C-a uz neke dodatke i neka ograničenja koja odgovaraju bitno različitoj arhitekturi grafičkog procesora, u usporedbi s centralnim procesorom, na kojem se programi napisani u GLSL-u izvode.

Glavna funkcija u programu grafičkog procesora je funkcija `main()` od koje počinje izvršavanje. Stil pisanja je slobodan, a komentari se označuju ili između „/*“ i „*/“ ili se započinju s „//“ ako su u jednom redu.

Kao i u C-u postoji kontrola toka pomoću **if** i **if else** naredbi te petlji **for**, **while** i **do while**.

Pri korištenju kontrole toka bitno je znati koji model jedinica za sjenčanje podržava grafička kartica na kojoj će se program izvoditi.

Naime stariji modeli grafičkih procesora imaju ograničenu kontrolu toka do određene razine ili ju uopće ne dopuštaju. Tako npr. grafički procesori vrhova modela SM2.0 (eng. *Shader Model*) podržavaju samo kontrolu toka do dubine 24, dok model SM3.0 nema to ograničenje.

Također u programu vrhova treba razlikovati statičku kontrolu toka od dinamičke. Statička kontrola toka znači da tok ovisi o konstanti koja je ista za sve vrhove koji se crtaju jednim pozivom funkcije za crtanje, dok dinamička kontrola toka može mijenjati tok za svaki pojedini vrh.

Mogu se koristiti i ključne riječi **break** i **continue**.

1.2. Tipovi podataka

Osim osnovnih tipova podataka GLSL podržava i vektore, matrice i tipove podataka za uzorkovanje tekstura, a mogu se i definirati strukture podataka, kao i u C-u.

Osnovni tipovi podataka u GLSL-u:

- int - cjelobrojni tip podatka
- float - realni broj
- bool - Booleov tip podatka
- void - koristi se samo za funkcije koje ne vraćaju vrijednost

Vektori:

- vec2, vec3, vec4 - vektori s dvije, tri ili četiri realne vrijednosti
- ivec2, ivec3, ivec4 - vektori s dvije, tri ili četiri cjelobrojne vrijednosti
- ivec2, ivec3, ivec4 - vektori s dvije, tri ili četiri Booleove vrijednosti

Pojedinim komponentama vektora može se pristupiti kao članovima strukture, a imena komponenata su:

- x, y, z, w
- r, g, b, a
- s, t, p, q

Dozvoljeno je i referenciranje više članova odjednom pri čemu je redoslijed proizvoljan, ali sva imena komponenata moraju biti iz iste skupine.

Na primjer:

```
vec3 a = vec3(2.0, 4.0, 1.0);
vec2 b = a.zx;
```

Nakon inicijalizacije, vektor **b** će imati vrijednosti {1.0, 2.0}.

U navedenom primjeru, za inicijalizaciju je korišten konstruktor **vec3()**. Konstruktori pretvaraju jednu ili više vrijednosti nekog tipa u jednu vrijednost drugog tipa, a koriste se kao funkcije čije je ime jednako kao naziv tipa podatka koji se dobiva konstruktorom.

Matrice:

- mat2, mat3, mat4 - matrice realnih vrijednosti (2x2, 3x3, 4x4)
- mat2x2, mat2x3, mat2x4 - matrice različitih dimenzija

Podržane su matrice dimenzija 2×2, 2×3, 2×4, 3×2, 3×3, 3×4, 4×2, 4×3, i 4×4 realnih vrijednosti. Prvi broj u nazivu tipa je broj stupaca, a drugi broj redova.

Npr. tip matrice s četiri stupca i dva retka označava se sa: `mat4x2`

Elementima matrice može se pristupiti kao da se radi o dvodimenzionalnom polju, npr:

```
mat2x3 m = mat2x3( 1.0, 1.0, 1.0,    //prvi stupac
                   2.0, 2.0, 2.0 );  //drugi stupac

m[0] = vec3(0.5, 0.5, 0.5);          //postavlja prvi stupac matrice
m[1][2] = 2.5;                       //postavlja 3 element u 2 stupcu
```

Nad matricama i vektorima definirani su operatori množenja (+), oduzimanja (-), množenja (*) i dijeljenja (/) na način da operacije odgovaraju matematičkom rješavanju istih operacija.

Podaci za uzorkovanje tekstura:

- sampler1D - tip podatka za uzorkovanje 1D teksture
- sampler2D - tip podatka za uzorkovanje 2D teksture
- sampler3D - tip podatka za uzorkovanje 3D teksture
- samplerCube - tip podatka za uzorkovanje s kocke
- sampler1DShadow - tip podatka za preslikavanje 1D sjena
- sampler2DShadow - tip podatka za preslikavanje 2D sjena

Ovi tipovi podataka omogućuju pristupanje teksturama. Koriste se u kombinaciji s funkcijama za pretraživanje tekstura kako bi se dobio željeni element određene teksture. Nad njima nisu definirani matematički ni logički operatori.

Pristupanje teksturama bit će prikazano u kasnijim primjerima.

Strukture se mogu definirati koristeći ključnu riječ **struct** na jednak način kao i u programskom jeziku C, te se isto tako pristupa pojedinim elementima strukture, npr:

```
struct struktura {
 float foo;
 float bar;
} foobar;

foobar.foo = 0.87;
```

Postoje strukture koje su unaprijed definirane, a koriste se za pristup postavkama scene pomoću automata OpenGL-a (eng. *OpenGL state machine*). Primjer su postavke svjetla i materijala.

U GLSL-u se mogu koristiti samo jednodimenzionalna polja. Veličina polja mora biti određena konstantom prije nego se polje može koristiti.

1.3. Kvalifikatori i ugrađene varijable

Kvalifikator je oznaka koja određuje namjenu varijable tj. način na koji se može koristiti.

Varijablama označenim kao *attribute* i *uniform* može se pristupiti iz glavnog programa i mijenjati im vrijednost. Ovo je glavni način komunikacije između glavnog programa i programa vrhova i fragmenata.

1.3.1. Kvalifikator *const*

Kvalifikator *const* označava varijable čija se vrijednost nakon prevođenja programa ne može više mijenjati.

Ovim kvalifikatorom mogu se deklarirati lokalne i globalne varijable. Ovako deklarirane varijable mogu se samo čitati pa ih je potrebno inicijalizirati pri deklaraciji.

Primjer deklariranja *const* varijabli:

```
const float PI = 3.14;
const vec3 osX = vec3(1.0, 0.0, 0.0);
```

1.3.2. Kvalifikator *uniform*

Varijabla deklarirana kao *unifom* ima istu vrijednost za sve vrhove jednog lika. Koristi se za deklariranje globalnih varijabli kojima se želi pristupati iz glavnog programa kako bi se utjecalo na izvođenje programa za sjenčanje vrhova i programa za sjenčanje elemenata slike.

Kako bi se ovakvoj varijabli pristupalo iz glavnog programa, potrebno je prvo dobiti njenu memorijsku lokaciju pozivom funkcije:

```
GLint glGetUniformLocation(GLuint program, const char *name);
```

Parametri su:

`program` – upravljačka varijabla programa za sjenčanje
`name` – ime varijable

Navedena funkcija vraća lokaciju varijable kojoj se zatim dodjeljuje vrijednost nekom od sljedećih funkcija, ovisno o tipu varijable:

```
void glUniform1f(GLint location, GLfloat v0);
void glUniform2f(GLint location, GLfloat v0, GLfloat v1);
void glUniform3f(GLint location, GLfloat v0, GLfloat v1, GLfloat v2);
void glUniform4f(GLint location, GLfloat v0, GLfloat v1, GLfloat v2,
GLfloat v3);
```

Parametri su:

`location` – lokacija varijable
`v0, v1, v2, v3` – realne vrijednosti

1.3.3. Kvalifikator *attribute*

Kvalifikator *attribute* označava globalne varijable koje sadrže informacije o pojedinom vrhu, te za svaki vrh mogu biti različite. Ovakve varijable se mogu koristiti samo u procesoru vrhova i to samo za čitanje.

One također služe za prijenos podataka između glavnog programa i programa za sjenčanje vrhova, a pristupa im se na sličan način kao i *uniform* varijablama s time da se vrijednost ovih varijabli može mijenjati za svaki pojedini vrh.

Prvo se dohvaća lokacija varijable u memoriji funkcijom:

```
GLint glGetAttribLocation(GLuint program,char *name);
```

Parametri su:

program – upravljačka varijabla programa za sjenčanje
name – ime varijable

Zatim se dodjeljuje vrijednost ovisno o tipu varijable pozivom neke od funkcija:

```
void glVertexAttrib1f(GLint location, GLfloat v0);
void glVertexAttrib2f(GLint location, GLfloat v0, GLfloat v1);
void glVertexAttrib3f(GLint location, GLfloat v0, GLfloat v1,GLfloat v2);
void glVertexAttrib4f(GLint location, GLfloat v0, GLfloat v1,,GLfloat v2,
GLfloat v3);
```

Parametri su:

`location` – lokacija varijable
`v0, v1, v2, v3` – realne vrijednosti

1.3.4. Kvalifikator *varying*

Vrijednost globalne varijable deklarirane kvalifikatorom *varying* interpolira se na prijelazu iz procesora vrhova u procesor fragmenata. U procesoru vrhova u ovakvu varijablu se može pisati, a u procesoru fragmenata samo čitati.

Ove varijable služe za prijenos podataka iz programa za sjenčanje vrhova u program za sjenčanje elemenata slike.

1.3.5. Ugrađene varijable

Postoje varijable koje su unaprijed definirane u OpenGL API-u, a koriste se za postavljanje scene (svjetla, materijali, matrice) te omogućuju procesoru vrhova da na izlazu pošalje transformirane koordinate vrha, a procesoru fragmenata da upiše boju fragmenta u međuspremnik boje.

Imena tih varijabli počinju sa „gl_“, na primjer:

```
uniform gl_LightSourceParametars gl_LightSource[gl_MaxLights];
//sadrži sve podatke o svjetlima na sceni

attribute vec4 gl_Vertex;
//varijabla iz koje program vrhova čita koordinate vrha

vec4 gl_Position;
//program vrhova mora na izlazu dati transformirane koordinate vrha
//tako da ih upiše u ovu varijablu
```

1.4. Ugrađene funkcije

Programski jezik GLSL nudi mnogo ugrađenih funkcija nad vektorskim i skalarnim podacima koje se mogu koristiti u programu za sjenčanje vrhova i programu za sjenčanje elemenata slike.

Ugrađene funkcije uvelike olakšavaju manipulaciju podacima, a u nekim slučajevima su i nezamjenjive kao što su funkcije za pregled tekstura.

Neke od operacija koje su podržane ovakvim funkcijama uključuju trigonometrijske funkcije, eksponencijalne funkcije, geometrijske funkcije, operacije nad matricama, relacije nad vektorima itd.

Ukoliko već postoji ugrađena funkcija za neku operaciju, preporuča se koristiti ju radije nego pisati svoju jer su ugrađene funkcije visoko optimirane, a mogu biti i sklopovski ubrzane.

Tablica 1-1 Primjeri nekih funkcija za pregled tekstura

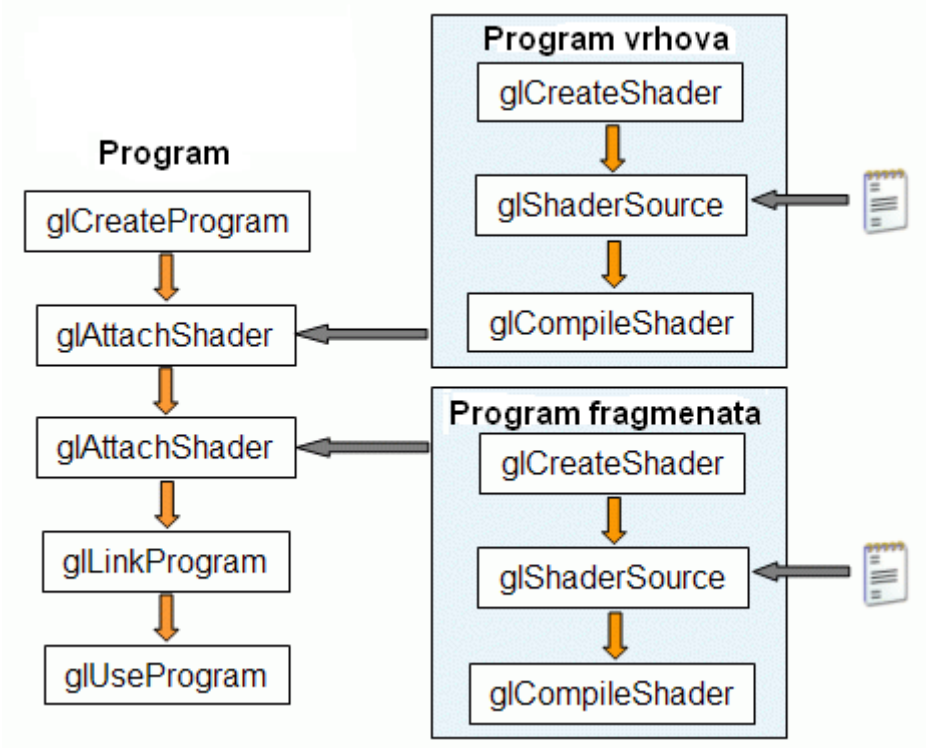
Prototip funkcije	**Opis funkcije**
vec4 **texture1D** (sampler1D *sampler,* float *coord* [, float *bias*])	Iz teksture određene varijablom za uzorkovanje teksture *sampler*, pronalazi traženi element čije su koordinate određene vektorom realnih vrijednosti *coord*. Pri tome se ovisno o postavkama može koristiti metoda mip-map kako bi tekstura bila pravilno filtrirana. Funkcije vraćaju vektor koji predstavlja vrijednost elementa teksture, najčešće boju (4 kanala, crvena, zelena, plava i alfa kanal)
vec4 **texture2D** (sampler2D *sampler*, vec2 *coord* [, float *bias*])	
vec4 **texture3D** (sampler3D *sampler*, vec3 *coord* [, float *bias*])	
vec4 **textureCube** (samplerCube *sampler*, vec3 *coord* [, float *bias*])	

Najbitnije u ovom radu su funkcije za pregled tekstura (neke su navedene u Tablici 1-1). Njima se kao argument predaje varijabla za uzorkovanje teksture (eng. *sampler*) i vektor koordinate teksture. Ove funkcije su dostupne u programu za sjenčanje vrhova kao i u programu za sjenčanje fragmenata.

1.5. Prevođenje, spajanje i učitavanje

Napisani programi vrhova i fragmenata se najčešće spremaju svaki u zasebnu datoteku te se onda iz glavnog programa učitaju, prevedu (kompiliraju) i spoje.

Programsko sučelje OpenGL nudi ugrađene funkcije za prevođenje, spajanje i učitavanje programa napisanih u programskom jeziku GLSL, a redoslijed izvođenja tih funkcija dan je slikom 1-1.



Slika 1-1 Prevođenje i spajanje programa vrhova i fragmenata funkcijama OpenGL-a

Pozivanjem funkcije **glCeateShader()** stvara se spremnik za program vrhova ili fragmenata.

```
GLuint glCreateShader(GLenum shaderType);
```

Argumenti: `shaderType` – GL_VERTEX_SHADER ili GL_FRAGMENT_SHADER.

Funkcija vraća upravljačku varijablu za taj program koju zatim kao argument, zajedno sa pokazivačem na kod programa u memoriji, predajemo funkciji **glShaderSource().**

```
void glShaderSource(GLuint shader, int numOfStrings, const char
**strings, int *lenOfStrings);
```

Argumenti: `shader` – upravljačka varijabla programa

`numOfStrings` – broj nizova znakova u polju

`strings` – polje nizova znakova

`lenOfStrings` – polje sa podacima o duljini nizova znakova

Program se zatim prevodi pozivanjem funkcije **glCompileShader()**.

```
void glCompileShader(GLuint shader);
```

Argumenti: `shader` – upravljačka varijabla programa

Ovaj postupak se mora ponoviti za oba programa, program vrhova i program fragmenata.

Prevedeni programi se zatim moraju spojiti u jedan zajednički program na sljedeći način:

Stvori se spremnik za zajednički program funkcijom **glCreateProgram()** koja vraća upravljačku varijablu za taj program.

```
GLuint glCreateProgram(void);
```

Zajedničkom programu pridruže se zasebno program vrhova i program fragmenata pozivima funkcije **glAttachShader().**

```
void glAttachShader(GLuint program, GLuint shader);
```

Argumenti: `program` – upravljačka varijabla zajedničkog programa

`shader` – upravljačka varijabla programa koji se pridružuje

Programi se spoje funkcijom **glLinkProgram()**.

```
void glLinkProgram(GLuint program);
```

Argumenti: `program` - upravljačka varijabla zajedničkog programa

Nakon spajanja program se stavlja u uporabu funkcijom **glUseProgram()** te se tako zamjenjuje ugrađena funkcionalnost grafičkog procesora ili prijašnji učitani program.

```
void glUseProgram(GLuint program);
```

Argumenti: `program` – upravljačka varijabla zajedničkog programa

Ovakvih se programa može napraviti više i tako po potrebi koristiti program koji trenutno želimo.

2. Postupci preslikavanja tekstura

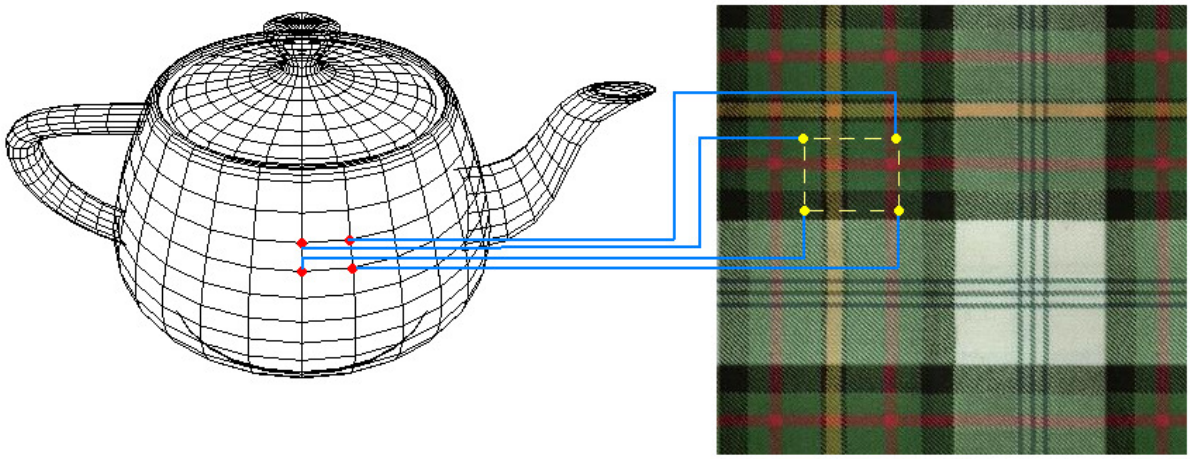
U ovom će poglavlju biti detaljnije razrađeno i objašnjeno nekoliko različitih postupaka preslikavanja tekstura upotrebom programirljivih grafičkih procesora. Također će biti prikazano koje su prednosti i namjene određenih postupaka.

Teksturiranje je izvedeno korištenjem jedinice za sjenčanje fragmenata, a sav kod za program fragmenata je napisan u programskom jeziku GLSL.

Osnovna korištena metoda je dohvat teksture (eng. *texture fetching*).

2.1. Osnovno teksturiranje

Najjednostavnije teksturiranje je preslikavanje 2D slike na površinu 3D modela. U programskom jeziku GLSL za to se koristi varijabla tipa `sampler2D` koja se predaje funkciji `texture2D()` zajedno s koordinatama elementa teksture. Budući da se radi o 2D teksturi, koordinate su tipa `vec2`.



Slika 2-1 Preslikavanje teksture

Budući da su koordinate elementa teksture pridružene pojedinom vrhu poligona, a vrhovi se obrađuju u programu za sjenčanje vrhova, potrebno je koordinate iz tog programa prenijeti u program za sjenčanje fragmenata.

Za to nam služe `varying` varijable, koje program za sjenčanje vrhova može postavljati, a program za sjenčanje fragmenata može čitati.

Kako bi se teksturi moglo pristupiti iz programa za sjenčanje, potrebno ju je prvo učitati i postaviti u glavnom programu koristeći OpenGL API.

Ovaj postupak opisan je u sljedećem odsječku koda:

```
/* glavni program */

GLuint texID;       //identifikacijski broj teksture

// prvo se generira identifikacija teksture, svaka tekstura dobiva
// različiti broj preko kojeg joj se pristupa
glGenTextures(1, &texID);

// tekstura se postavlja kao radna (trenutna)
glBindTexture(GL_TEXTURE_2D, texID);

// postave se načini filtriranja ukoliko je potrebno povećati ili
// smanjiti teksturu
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_NEAREST);

// ova funkcija čita podatke iz data i gradi potrebne mip-mape
// data       - pokazivač na učitanu teksturu u memoriji
// components- broj komponenti boje
// width      - širina slike
// height     - visina slike
// uzima još format (GL_RGB) i tip podataka (GL_UNSIGNED_BYTE)
gluBuild2DMipmaps( GL_TEXTURE_2D, components, width, height,
                   GL_RGB, GL_UNSIGNED_BYTE, data );
```

Nakon što je učitana u glavnom programu, teksturi se može pristupati u programima za sjenčanje pomoću ugrađenih varijabli pa bi osnovni program za sjenčanje vrhova s podrškom za teksture izgledao ovako:

```
/* Program za sjenčanje vrhova */
void main() {
// varying vec4 gl_TexCoord[ ] - polje kojim se transformirane
// koordinate teksture prenose iz programa vrhova u program fragmenata
// gl_TextureMatrix[] - matrica za transformaciju koordinata tekstura
// attribute vec4 gl_MultiTexCoord0 - izvorne koordinate tekstura
 gl_TexCoord[0] = gl_TextureMatrix[0] * gl_MultiTexCoord0;
 gl_Position = ftransform();
}
```

Program za vrhove uzima koordinate tekstura iz ugrađene varijable `attribute vec4 gl_MultiTexCoord0` koja je obilježje pridruženo pojedinom vrhu te ih zatim transformira množeći s matricom za transformaciju tekstura `gl_TextureMatrix[]` i dobiveni rezultat

prenosi programu za sjenčanje fragmenata preko varijable `varying vec4 gl_TexCoord[]` kao jedan od izlaza iz programa.

Program za sjenčanje fragmenata dobiva interpolirane vrijednosti varijable `gl_TexCoord[]` ovisno o položaju fragmenta između pripadnih vrhova.

Osnovni program fragmenata koji obavlja teksturiranje izgleda ovako:

```
/* Program za sjenčanje fragmenata */

uniform sampler2D tex;

 void main(){
       gl_FragColor = texture2D(tex,gl_TexCoord[0].st);
 }
```

U programu za sjenčanje fragmenata potrebno je imati varijablu za uzorkovanje teksture (`sampler2D`) kako bi se pozivom funkcije `texture2D()` dobila vrijednost elementa teksture na koordinatama `gl_TexCoord[0]` .

Dobivena vrijednost u programu se može iskoristiti na više načina, a u gornjem primjeru se izravno dodjeljuje boja fragmenta.

Slika 2-2 Rezultat izvođenja osnovnog teksturiranja pomoću grafičkog procesora

2.2. Višestruko teksturiranje

Višestruko teksturiranje je postupak preslikavanja više tekstura na istu površinu (poligon). Takav postupak otvara skoro neograničene mogućnosti kombiniranja različitih tekstura pri čemu svaka može određivati različita svojstva materijala.

Tako na primjer možemo postaviti jednu teksturu kao izvor osnovne boje, drugu teksturu kao izvor osvjetljenja (eng. *ligth map*), treću kao sliku okoline (eng. *environment map*) itd.

Teksturama možemo uvjerljivo prikazivati i detaljne neravnine (eng. *bump*) na površinama.

Sve novije grafičke kartice podržavaju višestruko teksturiranje, a pogotovo je zanimljivo taj postupak obavljati na jedinicama za sjenčanje jer nam omogućava da sami odredimo kako će se pojedina tekstura upotrijebiti.

Ako se koristi GLSL kao jezik za sjenčanje, dodavanje tekstura relativno je jednostavan postupak i sličan je osnovnom teksturiranju s tim da je potrebno koristiti više varijabli za pristup teksturama.

U programu za sjenčanje vrhova potrebno je samo dodati transformaciju koordinata za svaku teksturu koja se koristi:

```
/* Program za sjenčanje vrhova */
void main() {
// prva tekstura
 gl_TexCoord[0] = gl_TextureMatrix[0] * gl_MultiTexCoord0;

// druga tekstura
 gl_TexCoord[1] = gl_TextureMatrix[1] * gl_MultiTexCoord1;

 gl_Position = ftransform();
}
```

Program za sjenčanje fragmenata mora za svaku korištenu teksturu imati po jednu varijablu za uzorkovanje tekstura.

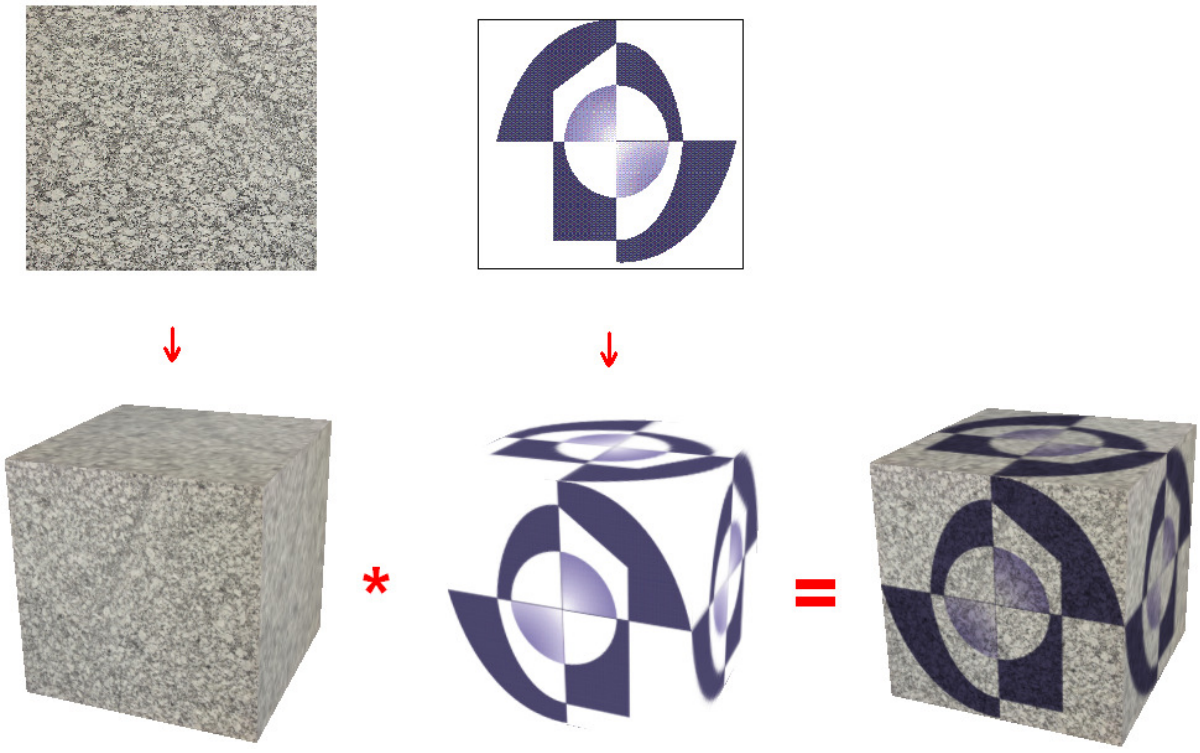
Nakon što se dohvate vrijednosti različitih tekstura, s tim se vrijednostima može postupati kao i sa bilo kojom drugom varijablom pa je na programeru da odredi koja će biti krajnja boja fragmenta i kako će pojedina tekstura na to utjecati.

Program za sjenčanje fragmenata mora na izlazu dati boju fragmenta, ali nije određeno kako će tu boju dobiti niti je obvezno koristiti sve dostupne teksture.

Jedan primjer korištenja više tekstura u programu fragmenata:

```
/* Program za sjenčanje fragmenata */

uniform sampler2D tex1, tex2;

void main(){
 vec4 color1 = texture2D(tex1,gl_TexCoord[0].st);
 vec4 color2 = texture2D(tex2,gl_TexCoord[1].st);

 gl_FragColor = color1*color2;
}
```

U ovom primjeru pretpostavljeno je da obje teksture predstavljaju boju materijala, a krajnja boja fragmenta dobivena je množenjem boja tekstura.



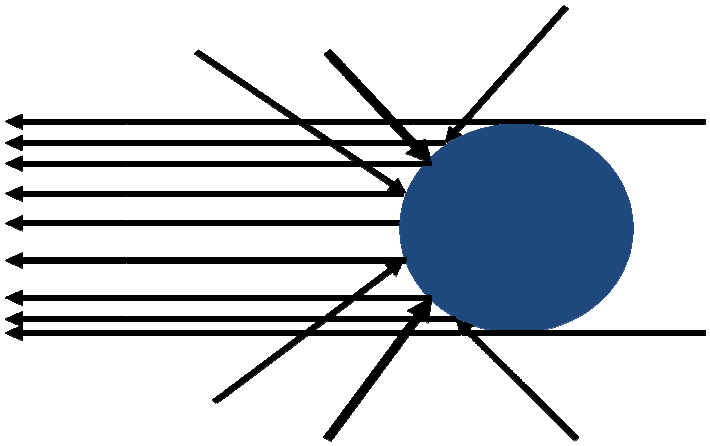
Slika 2-3 Množenje dviju tekstura preslikanih na kocku

Rezultat navedenih programa za sjenčanje prikazan je na Slici 2-3 gdje se može uočiti kako je pojedina tekstura preslikana i kako su obje spojene u završnu boju fragmenta.

2.3. Preslikavanje s kugle

Preslikavanje s kugle (eng. *sphere mapping*) je način preslikavanja okoline (eng. *environment mapping*) tako da se pomoću vektora odbijanja svjetlosti pronalazi koordinata teksture koja prikazuje sliku okoline na zamišljenoj zrcalnoj površini kugle (eng. *sphere map*). Ovakve teksture se mogu dobiti različitim postupcima, ali za potrebe ovog rada, biti će korištene gotove teksture dok će dobivanje potrebnih koordinata biti detaljnije objašnjeno.

Ukoliko se zamisli beskonačno udaljeni promatrač koji promatra zrcalnu kuglu, može se uočiti kako promatrač vidi odbijenu svjetlost koja iz svih smjerova dolazi na površinu kugle kao na Slici 2-4.



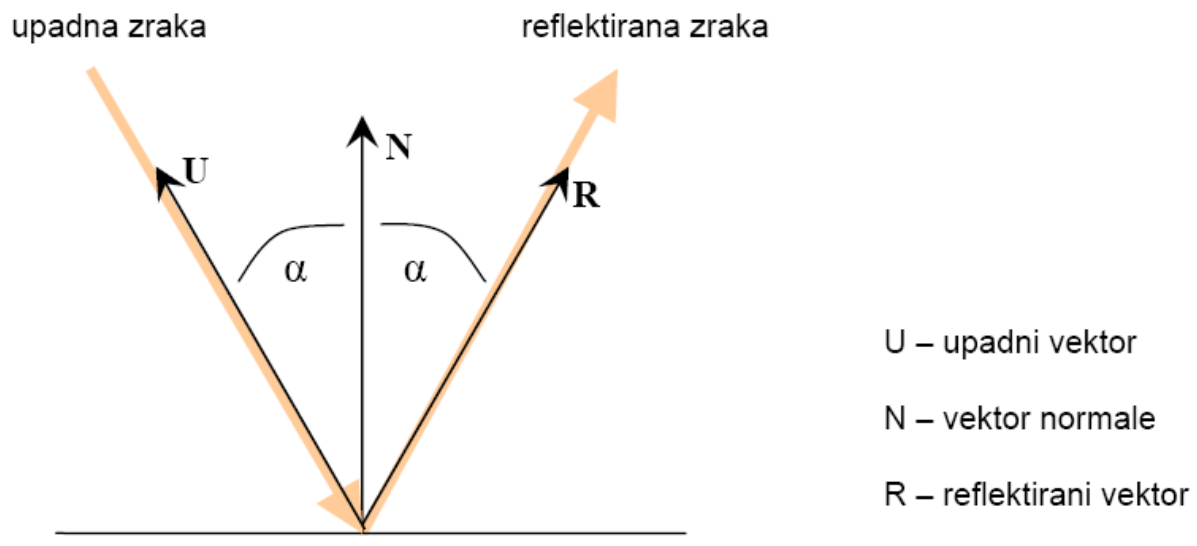
Slika 2-4 Odbijanje svjetlosti na površini kugle

Ako se sada zamisli da je kugla beskonačno mala tj. kugla postaje točka, u toj točki je moguće prikazati cijelu sliku okoline. Ta se slika projicira na 2D površinu kako bi dobili teksturu okoline (Slika 2-5).

Slika 2-5 Primjer teksture za preslikavanje s kugle

Postavlja se pitanje kako sada iz te 2D slike, ponovo napraviti zrcalni odraz okoline na 3D modelu. Odgovor na to pitanje leži u računanju vektora odbijanja svjetlosti na površini.

Kut upadne zrake na površinu lika i normale te ravnine jednak je kutu odbijene zrake ali na suprotnu stranu.



Slika 2-6 Upadna i odbijena zraka

Ako je poznat vektor normale i upadni vektor, uz uvjet da su ti vektori prethodno normirani, odbijenu zraku računamo po formuli:

$$R = (2U \bullet N) * N - U$$

Programski jezik GLSL nudi ugrađenu funkciju `reflect()` koja računa vektor smjera odbijanja.

Kada je pronađen vektor smjera odbijanja potrebno ga je projicirati na dvodimenzionalnu ravninu kako bi dobili potrebne koordinate teksture.

Način na koji se okolina prvotno projicirala na teksturu možemo zamisliti kao ortogonalnu projekciju površine jedinične kugle na ravninu teksture. Pri ortogonalnoj projekciji, svi su upadni vektori isti i, budući da gledamo iz pozitivne osi z, upadni vektor je jednak $\{0, 0, 1\}$, a kako je kugla jedinična, normala u nekoj točki na površini kugle je $\{x, y, \sqrt{1 - x^2 - y^2}\}$ što možemo i zapisati pomoću koordinata teksture kao $\{s', t', \sqrt{1 - s'^2 - t'^2}\}$ pri čemu su s' i t' u središtu teksture jednaki $\{0, 0\}$.

Ako sada taj upadni vektor i normalu uvrstimo u jednadžbu vektora odbijanja dobijemo:

$$R = 2 * N_z * N - \{0, 0, 1\}$$

Iz čega obratom dobivamo normalu:

$$N = \{R_x,\ R_y,\ R_z + 1\}/2N_z$$

Kada normalu normiramo dobijemo konačno:

$$N = \{R_x,\ R_y,\ R_z + 1\}/\sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}$$

Pa iz početnog zapisa normale pomoću koordinata teksture proizlazi:

$$s' = R_x/\sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}$$

$$t' = R_y/\sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}$$

Svi vektori su jedinični vektori, a pojedine komponente vektora su u intervalu [-1, 1].

Budući da koordinate tekstura moraju biti u intervalu [0, 1], dobivene je brojeve potrebno skalirati na sljedeći način:

$$s = \frac{s'}{2} + 0.5$$

$$t = \frac{t'}{2} + 0.5$$

Varijable **s** i **t** su tražene koordinate teksture u postupku preslikavanja s kugle.

Koristeći navedene formule može se napisati program za sjenčanje vrhova koji radi potrebne transformacije koordinata tekstura.

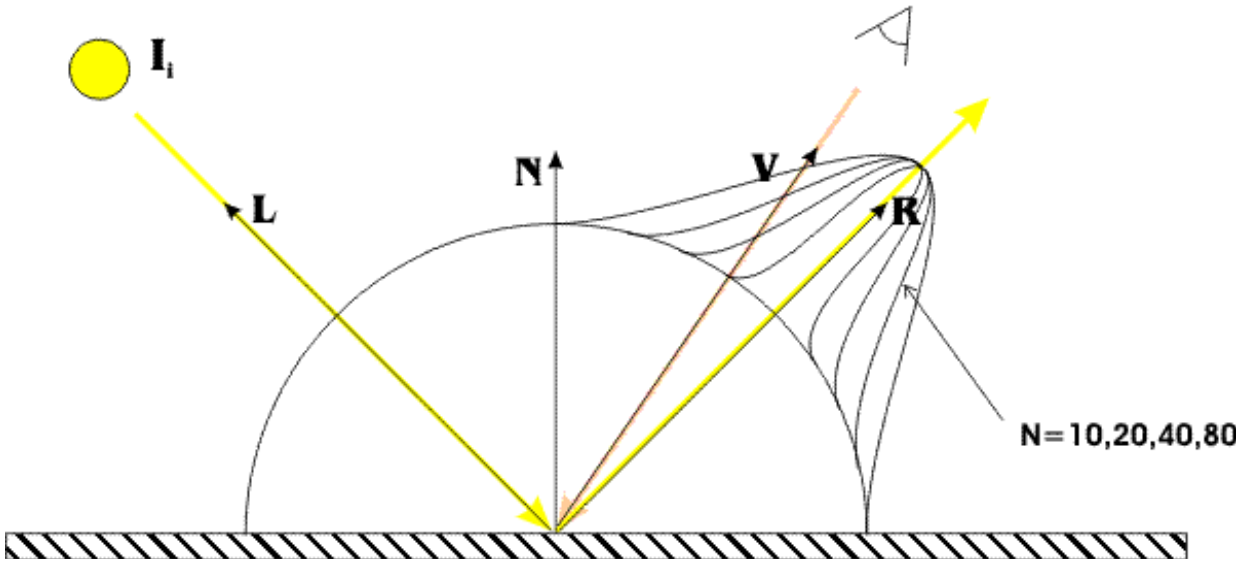
```
/* program za sjenčanje vrhova */
void main()
{
 gl_Position = ftransform();

 gl_TexCoord[0] = gl_MultiTexCoord0;

 vec3 U = normalize( vec3(gl_ModelViewMatrix * gl_Vertex) );
 vec3 N = normalize( gl_NormalMatrix * gl_Normal );
 vec3 R = reflect( U, N );
 float m = 2.0 * sqrt( R.x*R.x + R.y*R.y + (R.z+1.0)*(R.z+1.0) );
 gl_TexCoord[1].s = R.x/m + 0.5;
 gl_TexCoord[1].t = R.y/m + 0.5;
}
```

Zapisane koordinate tekstura, čitaju se u programu za sjenčanje fragmenata u kojem se i računa krajnja boja fragmenta koristeći jednu običnu teksturu i jednu teksturu okoline kako bi se dobio izgled zrcaljenja.

```
/* program za sjenčanje fragmenata */

uniform sampler2D colorMap;
uniform sampler2D envMap;

const float reflection = 0.4; // faktor odbijanja svjetlosti

void main (void)
{
 vec4 color = texture2D( colorMap, gl_TexCoord[0].st);
 vec4 env = texture2D( envMap, gl_TexCoord[1].st);

 gl_FragColor = color + env*reflection;
}
```

Izgled zrcalne površine možemo dodatno upotpuniti osvjetljenjem.

Osvjetljenje možemo računati po Phongovom modelu koji svjetlost rastavlja na tri komponente: svjetlost iz okoline (eng. *ambient*), raspršenu svjetlost (eng. *diffuse*) i zrcaljenu svjetlost (eng. *speccular*).



Slika 2-7 Odbijanje svjetlosti po Phongovom modelu

Prema Phongovom modelu osvjetljenje je najjače u smjeru odbijanja te zatim opada s kutem.

Potpuna jednadžba osvjetljenja glasi:

$$I = I_a k_a + I_d k_d (L \cdot N) + I_s k_s (R \cdot V)^n$$

Pri čemu su s *I* označeni intenziteti svjetlosti (intenziteti pojedinih komponenti, svojstvo izvora svjetlosti), *k* koeficijenti odbijanja svjetlosti (svojstvo materijala), *L* vektor smjera svjetla, *N* vektor normale, *R* vektor smjera odbijanja i *V* vektor pogleda. Svi vektori moraju biti prethodno normirani.

Operator „ ⋅ “ predstavlja skalarni produkt vektora, a u GLSL-u postoji i ugrađena funkcija koja obavlja ovu operaciju:

```
float dot( vec4, vec4 );
```

U kombinaciji s ovim modelom osvjetljenja, preslikavanje okoline s površine kugle daje relativno dobre rezultate i izgled zrcaljenja na površini modela.

Međutim, ovo je poprilično neprecizna metoda preslikavanja okoline jer je stvarna okolina samo aproksimirana teksturom. Također, zbog načina projiciranja okoline, projekcija je najkvalitetnija u sredini teksture, a prema rubu joj kvaliteta drastično opada.

Ponekad se pri uzorkovanju rubova teksture javljaju lako uočljive pogreške (artefakti).

Slika 2-8 Preslikavanje okoline s površine kugle i Phongov model osvjetljenja

Unatoč mnogim poteškoćama koje se javljaju kod ove metode, u nekim slučajevima daje vrlo dobre rezultate, a zbog svoje iznimne jednostavnosti još se uvijek često koristi.

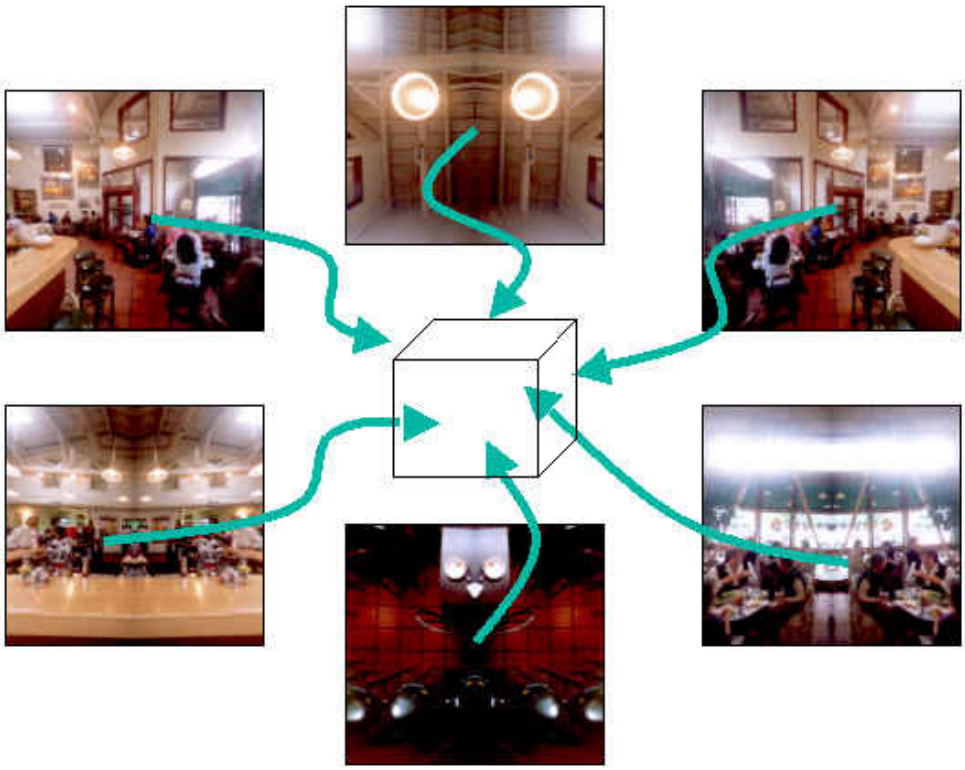
2.4. Preslikavanje s kocke

Za razliku od preslikavanja s kugle, metoda preslikavanja s kocke (eng. *cube mapping*) koristi teksturu na kojoj je okolina prikazana kao rastvorena kocka (slika 2-9). Zbog toga je ovakva tekstura zapravo sastavljena od šest manjih slika koje odgovaraju stranicama kocke.

Ovaj način preslikavanja okoline daje puno bolje rezultate od preslikavanja s kugle i nema nikakvih problema s pogreškama u prikazu jer projekcija nije zaobljena kao u slučaju preslikavanja s kugle te je potpuno neovisna o smjeru pogleda.

Nedostatak ovakvog postupka je nešto složenija izvedba i potreba za šest tekstura.

Šest strana kocke odgovara poluosima u trodimenzionalnom koordinatnom sustavu.



Slika 2-9 Preslikavanje okoline s površine kocke

Pri preslikavanju okoline ovim postupkom, svi vektori moraju biti izraženi u koordinatnom sustavu „svijeta“ tj. ne koristi se lokalni koordinatni sustav kamere i gledanog objekta već koordinatni sustav okoline s obzirom na objekt, koji je neovisan o pogledu.

Današnji grafički sklopovi nude podršku za ovaj način preslikavanja pa tako u programskom jeziku GLSL postoji tip varijable za uzorkovanje ovakvih tekstura `samplerCube` i pridružena funkcija `textureCube()` koja dohvaća željeni element teksture.

Funkcija `textureCube()` uzima varijablu za uzorkovanje te trodimenzionalni vektor koji odgovara smjeru odbijanja vektora pogleda na određeni vrh. Iz tog vektora, funkcija određuje koju stranu kocke treba uzorkovati, te zatim projicira vektor na tu plohu kako bi se dobile dvodimenzionalne koordinate elementa teksture.

Program za sjenčanje vrhova koji koristi metodu preslikavanje okoline s kocke u kombinaciji s osnovnom teksturom izgleda ovako:

```
/* Program za sjenčanje vrhova */

// matrica koja određuje položaj vrha u odnosu na okolinu
uniform mat4 ModelWorld4x4;
// pozicija pogleda u odnosu na okolinu
uniform vec4 vViewPosition;

void main( void )
{
   gl_Position = ftransform();

   // podmatrica za transformaciju normale
   mat3 ModelWorld3x3 = mat3( vec3( ModelWorld4x4[0]),
                              vec3( ModelWorld4x4[1]),
                              vec3( ModelWorld4x4[2]) );

   // pozicija u odnosu na okolinu
   vec4 WorldPos = ModelWorld4x4 *  gl_Vertex;

   // normala u odnosu na okolinu
   vec3 N = normalize( ModelWorld3x3 * gl_Normal );

   // vektor pogleda u odnosu na okolinu
   vec3 Eye = normalize( WorldPos.xyz - vViewPosition.xyz );

   // vektor odbijanja svjetlosti
   gl_TexCoord[1].xyz = reflect( Eye, N );

   gl_TexCoord[0].xy  = gl_MultiTexCoord0.xy;
}
```

Pripadajući program za sjenčanje fragmenata:

```
/* program za sjenčanje fragmenata */

uniform sampler2D baseMap;
uniform samplerCube cubeMap;

void main( void )
{
   // boja dobivena osvjetljenjem i osnovnom teksturom
   vec4 base_color = texture2D( baseMap, gl_TexCoord[0].xy );

   // dohvacanje elementa teksture okoline
   vec3 cube_color = textureCube(cubeMap, gl_TexCoord[1].xyz).rgb;

// konacna boja je kombinacija osnovne teksture i teksture okoline
   gl_FragColor = vec4((base_color.xyz+cube_color)/2, 1.0);
}
```

Rezultat ovih programa za sjenčanje je model koji primjenjuje dvije teksture, jednu kao osnovnu boju površine, a drugu metodom preslikavanja okoline s površine kocke što daje vrlo uvjerljiv izgled zrcaljenja okoline.

Ako tome dodamo i Phongov model osvjetljenja, dobivena slika jako dobro oponaša stvarni izgled zrcaljenja na glatkim površinama.

Slika 2-10 Preslikavanje okoline s površine kocke

Zaključak

U svijetu računalne grafike najveći je pothvat što vjernije imitirati stvarne objekte, materijale i scene. Jedna takva imitacija je i teksturiranje, tj. nanošenje teksture na virtualnu površinu. Različitim postupcima teksturiranja, možemo dobiti raznovrsne učinke od jednostavnog korištenja teksture kao boje, do nanošenja više tekstura pri čemu svaka predstavlja različito svojstvo materijala.

Napredak grafičkog sklopovlja omogućio je programerima pristup samom protočnom sustavu i tako otvorio novu eru u primjeni računalne grafike. Programibilni grafički procesori omogućavaju promjenu ugrađene funkcionalnosti u grafičkom sustavu. Programski jezik za sjenčanje GLSL ima poznatu C sintaksu, a daje pristup svim naprednim mogućnostima grafičkog procesora.

Današnje grafičke kartice imaju po nekoliko stotina jedinica za sjenčanje na kojima se izvode programi za sjenčanje vrhova i fragmenata.

U tim se programima, teksture mogu proizvoljno dohvaćati i koristiti na razne načine u svrhu dobivanja krajnje boje fragmenta na ekranu.

Neke od najčešćih primjena tekstura pri višestrukom teksturiranju su preslikavanje okoline i preslikavanje površinskih neravnina.

Preslikavati okolinu možemo na više načina npr. preslikavanjem s kugle, paraboloidnim preslikavanjem, preslikavanjem s kocke itd.

Preslikavanje s kugle relativno je jednostavan postupak preslikavanja okoline, pri čemu se koristi jedna tekstura koja predstavlja odsjaj okoline na zrcalnoj površini kugle. Ovaj postupak daje lošije rezultate u nekim slučajevima jer je ovisan o kutu pogleda tj. za neke kutove će biti vidljiva pogreška.

Bolji način preslikavanja okoline je preslikavanje s kocke koje uvijek daje ispravan odraz okoline na površini virtualnog objekta, ali zahtjeva korištenje 6 tekstura.

Literatura

[1] Kessenich, John; Baldwin, Dave; Rost, Randi: "The OpenGL® Shading Language", Datum: 7. 9. 2006.

[2] António Ramires Fernandes: "GLSL Tutorial", 6. 2. 2006. http://www.lighthouse3d.com/opengl/glsl/index.php?intro

[3] MSDN - Resource Types, Microsoft Corporation, 2005. http://msdn.microsoft.com/en-us/library/bb205133(VS.85).aspx

[4] Jerome Guinot: "The Art of Texturing Using The OpenGL Shading Language" http://www.ozone3d.net/tutorials/glsl_texturing.php

[5] ATI Development Pages, http://developer.amd.com/GPU/Pages/default.aspx

[6] nVidia Developer Zone, http://developer.nvidia.com/page/home.html

[7] Matt Pharr; Randima Fernando: "GPU Gems 2", ožujak 2005.

[8] Wikipedia: "Texture mapping", lipanj 2008.

[9] Wikipedia: "Reflection mapping", lipanj 2008.

Dodatak A: Korištena programska potpora

Većina primjera programa za sjenčanje napravljeno je i testirano u programu *RenderMonkey* kojega je razvio ATI specifično za namjene programiranja programa za sjenčanje jezikom GLSL.

Osim navedenog, korišten je i program *ShaderDesigner* od *TyphoonLabs*-a također za namjenu razvoja programa za sjenčanje.

U sklopu završnog rada, priloženo je programsko rješenje svih navedenih primjera te poseban program napisan u C++ koji učitava i pokreće korištene programe za sjenčanje.

Glavni program otvara prozor, učitava sve dostupne programe za sjenčanje, prevodi ih, povezuje i korisniku omogućava odabir načina sjenčanja.

Sažetak

Postupci teksturiranja upotrebom grafičkog procesora

U ovom je radu obrađen programski jezik za sjenčanje GLSL te njegova primjena u konkretnim slučajevima s teksturiranjem. Razrađeno je nekoliko postupaka korištenja tekstura u programu za sjenčanje vrhova i programu za sjenčanje fragmenata. Na primjerima je prikazana primjena višestrukog teksturiranja za dobivanje različitih učinaka. Jedan od tih učinaka je zrcaljenje okoline koje je ostvareno preslikavanjem s kugle i preslikavanjem s kocke. Također je ukratko opisan Phongov model osvjetljenja te kombiniranje osvjetljenja s teksturiranjem.

Ključne riječi: OpenGL, jezik za sjenčanje, GLSL, jedinica za sjenčanje vrhova, jedinica za sjenčanje fragmenata, teksturiranje, višestruko teksturiranje, preslikavanje okoline, preslikavanje s kugle, preslikavanje s kocke

Abstract

Texture mapping by using graphics processing unit

This paper explains the use of GLSL programming language and its application in specific texturing procedures. It is shown how to implement several of these texturing procedures using vertex and fragment shaders. It is also shown by examples how to use multitexturing to achieve variety of effects. One of these effects is environment reflection which is done by sphere mapping and cube mapping. Also there is a brief explanation of Phong lighting model and how to combine it with texturing.

Keywords: OpenGL, shading language, GLSL, vertex shader, fragment shader, texture mapping, multitexturing, environment mapping, sphere mapping, cube mapping