Shaderprogrammering

Uppgift 6

Hårdvarushader

Skugg-Sampling

Johan Lavén
d15johla@student.his.se

Introduktion

Målet med skugg-sampling är att skapa en verklighetstrogen skugga som beräknas på ett objekt. Skuggan ska mörklägga backen på objektets motsatta sida från ljuskällan sett, samt att objektet själv ska mörkläggas på den sida som inte belyses direkt.

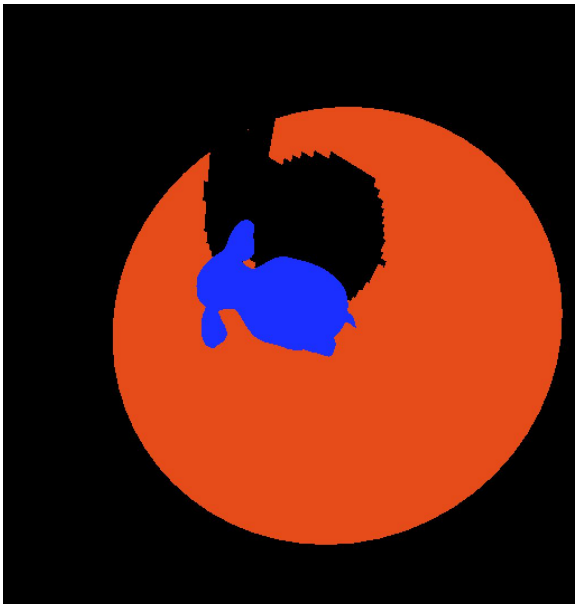
Det första som görs är att en skuggtextur (*ShadowMap*) renderas baserat på position, avstånd och rotatinen av objekten i scenen sett från ljuskällans/ljuskällornas position.

Designval och implementation

För att skugga plattformen baserat på elefanten så måste först skuggtexturen beräknas baserat på ljuskällan. Därefter så kan man basera ett flyttalsvärde utifrån 2D-projektionen av skulltexturen.

```
float unpack (vec4 colour)
{
   const vec4 bitShifts = vec4(1.0 / (256.0 * 256.0 * 256.0), 1.0 / (256.0 * 256.0),1.0 /
256.0, 1);
   return dot(colour , bitShifts);
}
...
float shadowMap = unpack(texture2DProj(ShadowMapTexture, shadowCoord);
```

Figur: Textrad ovan visar hur ett flyttal tas ut från en färg eller vector4.



Figur: Bilden ovan visar första utkastet av skuggan på basen.

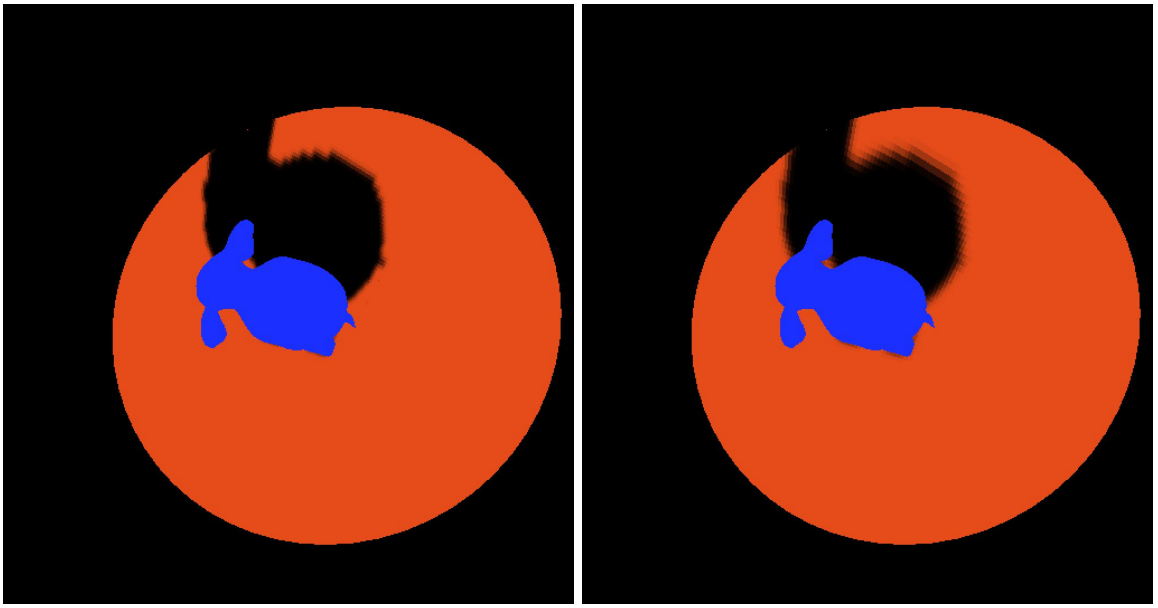
Mål nummer två är att få skuggan att se mjukare ut. Detta kommer att uppnås genom en sampling matris. Detta innebär att man tittar på värdet av skuggtexturen runt den pixel som ska påverkas, därefter adderas samtliga resultat av skuggningarna för att sen multipliceras in i slutresultatet. Idén var att göra samplingen genom en 5*5-matris för att få skuggningen att se

mjuk och verklig ut. Det ska vara möjligt att påverka avståndet av alla samplingar med hjälp av en variabel.

```
vec3 kernel[25];
kernel[0] = vec3( 0.0, 0.0, 0.1621 );
kernel[1] = vec3( 1.0, 0.0, 0.0983 );
…
kernel[23]= vec3( 2.0, 1.0, 0.0133 );
kernel[24]= vec3( 2.0,-1.0, 0.0133 );
```

```
for(int i = 0; i < KERNEL_SIZE; i++){
    shadowMap = unpack(texture2DProj(ShadowMapTexture, shadowCoord +
vec4(kernel[i].x*elephantShadowArea, kernel[i].y*elephantShadowArea,0,0)));
   shadow += float(depth < (shadowMap-shadowBias))*kernel[i].z;
}
```

Figurer: Exempel på hur definieringen av matrisen är implementerad i form av en array med 3-dimensionella vektorer (ovan). Exempel på hur samplingen går till när arrayen loopas igenom(nedan).



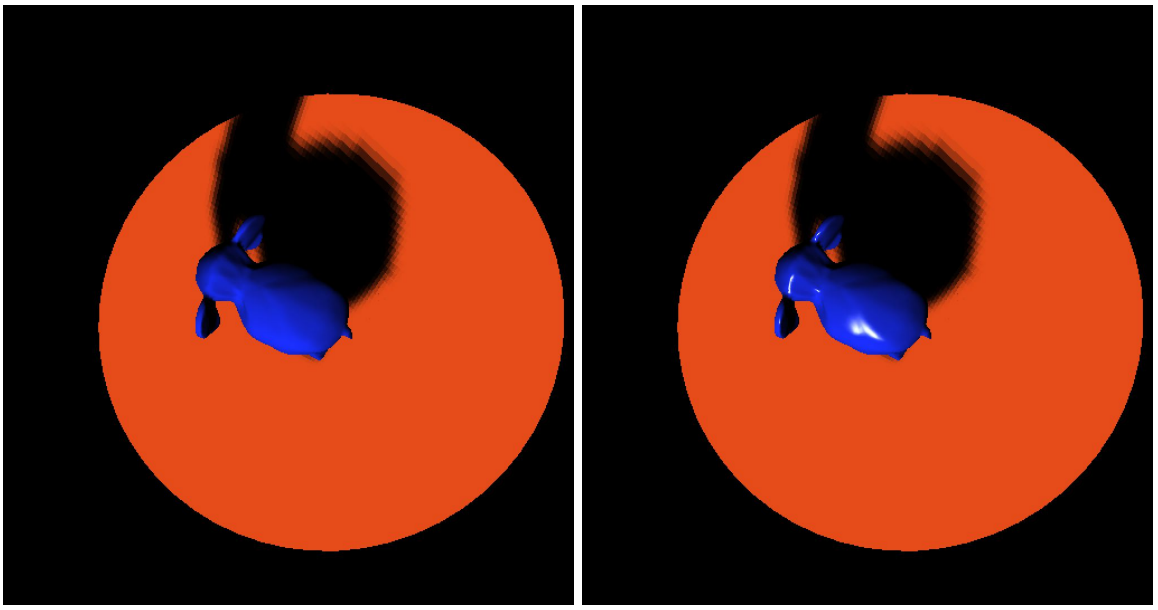
Figur: Illustration hur mjukheten på skuggan påverkas när avståndet mellan provtagningarna (*Samplingarna*) höjs. Exemplet till vänster har avståndet av en pixel och det till höger har avståndet två pixlar.

De sista två uppgifter som skulle implementeras är direkt skuggning på modellen samt en spekulär belysning på ytan. För att implementera skuggningen på elefantmodellen så används samma metod med en array bestående av offsets i x- och y-led samt en vikt som säger hur

mycket som varje sampling påverkar slutprodukten. Den spekulära belysningen är en klassisk phong belysning som räknas i fragmentprogrammet.

```
float specular = pow(clamp(dot(R,E),0.0,1.0),40.0) * specIntensity;
gl_FragColor = difCol*(shadow)+specular*(shadow);
```

Figur: Kod ovan visar på uträkningen av belysning samt hur den appliceras tillsammans med skuggningen.



Figurer: Vänster: Skuggan applicerad med samplingsmetoden
Höger: Applicerad phong spekulär belysning.

För och nackdelar med samplingskuggning

Metoden för att sampla i runtliggande områden kallas för "*Filtering*". I boken *OpenGL Shading Language* (Rost, Licea-Kane and Ginsburg, 2013) så beskrivs den här samplingsmetoden som processen att beräkna ett ensamt värde enligt provtagningar baserat på runtliggande värden. En fördel med detta är att man på ett smidigt sätt kan få mjuka texturer och skuggningar utan att behöva beräkna flera skuggtexturer (*shadow maps* eller *Depth map*) från olika vinklar. Då shadern slipper att beräkna fler än en skuggtextur så sparar man mycket prestanda genom att basera hela skuggan på en ensam textur. En nackdel med denna metod är att den inte riktigt stämmer överens med hur ljuset fungerar i verkligheten då ljuset sällan kommer från en ensam punkt. Metoden fungerar dock bra nog för att generera en elegant lösning för att uppnå ett estetiskt resultat.

Referenser

Rost, R., Licea-Kane, B. and Ginsburg, D. (2013). *OpenGL shading language*. 3rd ed. Upper Saddle River, NJ [u.a.]: Addison-Wesley, pp.487-495, 690.