Shaderprogrammering

Uppgift 8

Hårdvarushader
Deferred Rendering

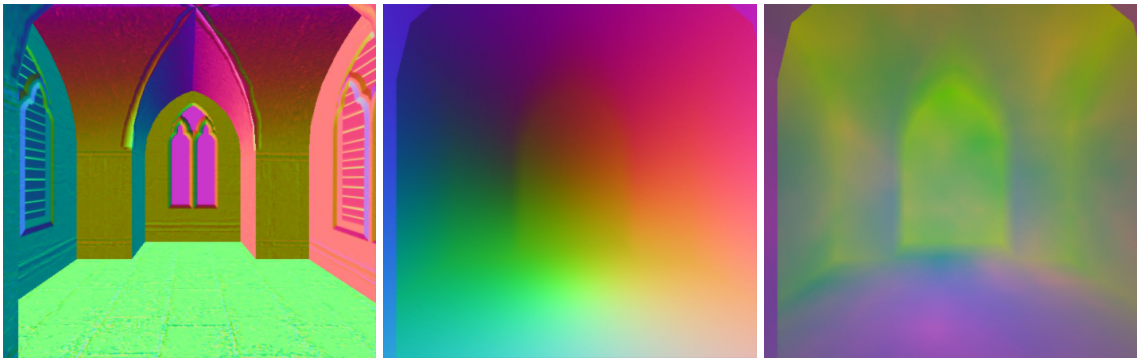
Johan Lavén
d15johla@student.his.se

Introduktion

Målet med *Deferred Rendering* är att dela upp renderingspipelinen på ett sånt sätt att man i separata pass räknar ut vilka ytor och polygoner som ska synas i scenen. Detta resulterar i att man inte behöver beräkna ljus och skuggor på texturer som inte kommer synas vilket sparar prestanda. I uppgift nummer 8 så är målet att rendera insidan av en krypta genom att experimentera med en deferred renderingspipeline.

Designval och implementation

De första tre passen gör olika beräkningar på 3D-modellen. Dessa beräkningar kan till exempel vara att skapa en normalmap, lightmap eller depthmap (textur som illustrerar djupet från kameran till polygonerna på alla koordinater). Resultaten från var och en av dessa pass används för att beräkna belysning och skuggning på olika sätt. (*Dessa är pass 0, 1 och 2*). Depthmapen innehåller även ett fraktalbrusvärde på den röda kanalen som senare kommer att användas för att rita ut bris eller dimma i ett senare pass.



Figurer: Vänster: Screenspace-representation av normal-textur
Mitten: Screenspace-represntation av avståndet till ljuskällan till alla väggar.
höger: Screenspace-representation där ljupet från kameran representeras av den gröna kanalen i vektorn, längden av ljuskällan i den blåa kanalen samt ett fraktalbrusvärde i den röda kanalen.

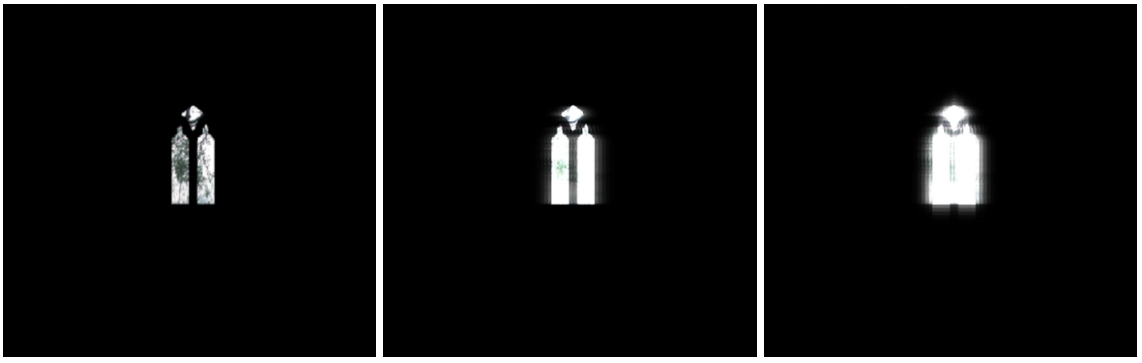
```
vec3 LightVec=(normalize(vLightVec)+1.0)*0.5;
```

Figur: Hur alla värden i LightMap-passet baseras på ljusvektor.

Scenen innehåller även en färgad textur(Diffuse) som mappas direkt mot ytan med hjälp av texturkordinaterna. (*Detta är pass 3 i renderingspipelinen*).

I scenen finns tre pass för att mappa en ljustextur mot ett fysiskt fönster samt att ge det effekt som gör kanterna lite mjukare och suddiga. (*Pass 4,5 och 6*). Anledningen till att det krävs tre pass för att slutföra en sådan effekt är för att i första passet så mappas texturen mot modellen och i de senare två passen skapas mjukheten/suddigheten separat i x- och y-led. Anledningen till att beräkningarna sker i två pass istället för ett är att man då kan nyttja två stycken

filter-arrayer för att sampla med en offset istället för en tvådimensionell array. Detta ger ökad prestanda på så sätt att samplingarna sker O(2n) gånger istället för O(n^2).



Figurer: Representationer av alla pass i ordning

```
vec3 kernel[15];
kernel[0] = vec3( 0.0, 0.0, 0.1995);
…
kernel[14]= vec3( 0.0,-7.0, 0.0004);
vec3 tmp = texture2D(Texture, texCoord).xyz;
for(int i = 0; i < 15; i++){
   tmp += (texture2D(Texture, (texCoord + (kernel[i].xy*LmapBlurArea))).xyz * kernel[i].z);
}
tmp += texture2D(Texture, texCoord).xyz;
```

Figur: Skapandet och användandet av en filter-array(*Kernel*) med vikt på z-elementet. Vikterna följer en Gausskurva med standardavvikelsen 2.

I pass 7 av totalt 9 pass så slås informationen från tidigare pass ihop. Textures från pass 3 appliceras med hjälp av normaltexturen generad i pass 0 och djuptexturen från pass 1. Ljusstyrkan av vardera pixel baseras på ljustexturen renderad i pass 2. Även dimman läggs till i detta pass baserat på x-värdet från djuptexturen som tidigare nämnt.

Figur: Vänster: Slutresultat efter pass 7 med alla sammanslagningar.
Höger: Samma pass med sepiatoning och förhöjd ljusstyrka.

Pass 7 har även ansvar för färgkorrigeringen av slutprodukten. Där innom ligger kod som har möjlighet att skapa sepiatoning över hela bilden samt ändra kontrasten och ljusstyrka.

I pass 8 och 9 så finns återigen två filter-arrayer som samplar och blurar slutresultatet från pass 7 i pass 8 fall och pass 8 när det kommer till pass 9. Denna mjukhet/suddighet fungerar på samma sätt som i pass 5 och 6 med den lilla skillnaden att suddigheten får mer effekt baserat på avståndet från kameran. Detta genom att multiplicera in avståndet taget från djuptexturens x-element till förskjutningen i filterlopen. I 9e passet finns även de sista två effekterna inlagda. En *Screenspace radial blur* som gör det grynigt ute i kanterna, ser lite ut som effekten att kameran skulle röra i snabb hastighet, samt ett direkt-renderat sikte ispererat av spel av skjutspel i förstapersonsvy.

Figur: Slutresultatet med alla effekter. Sepia, kontrast och ljusstyrka är på orginalvärden. Krysset i mitten slås ihop addiativt vilket resulterar i att det inte syns eller syns dåligt där den röda kanalen(*x-elementet*) redan har ett högt värde.

```
vec4 crosshair = vec4(0.0,0.0,0.0,0.0);
crosshair.x  = step(0.498, texCoord.x) - step(0.503, texCoord.x);
crosshair.z  = step(0.55,  texCoord.x) - step(0.45,  texCoord.x);
crosshair.y  = step(0.498, texCoord.y) - step(0.502, texCoord.y);
crosshair.w  = step(0.55,  texCoord.y) - step(0.45,  texCoord.y);
crosshair.y *=  crosshair.z;
crosshair.x *=  crosshair.w
```

Figur: Siktet definierat.
Dessa värden är hårdkodade och skulle behöva bytas ut mot justerbara parametrar.

För och nackdelar deferred rendering

Grundtanken med *deferred rendering* är att skjuta upp på tunga kalkyleringar som ljussättning och skuggning av en volymetrisk (3-dimensionell) scen tills shadern bestämt vilka polygoner som kommer att synas på den slutliga bildrutan. Detta resulterar i sparad prestanda på en scen som skulle kunna se exakt likadan ut då man försäkrar att inga renderingscyklar går ut på att beräkna information om en pixel som inte syns. I och med all denna extra prestanda som finns tillgänglig så får designers och grafiker mer artistisk frihet, Scenen kan göras större eller så kan den till exempel innehålla fler ljuskällor. Det är vanligt att ljuskällor i deferred rendering har en maxdistans som ljuset kan nå, och på så sätt så

behöver inte alltid alla ljuskällor i scenen räknas på samtliga synliga pixlar. En nackdel med tekniken är att den kräver lite mer minne hos grafikprocessorn då den för varje given stund sparar undan fler texturer för senare användning, det görs i en så kallad (*Frame Buffer*). I moderna system är detta sällan ett problem.

Referenser

Rost, R., Licea-Kane, B. and Ginsburg, D. (2013). *OpenGL shading language*. 3rd ed. Upper Saddle River, NJ [u.a.]: Addison-Wesley, pp.392-400.