Shaderprogrammering

Uppgift 4

Hårdvarushader
ljussättning och skuggning

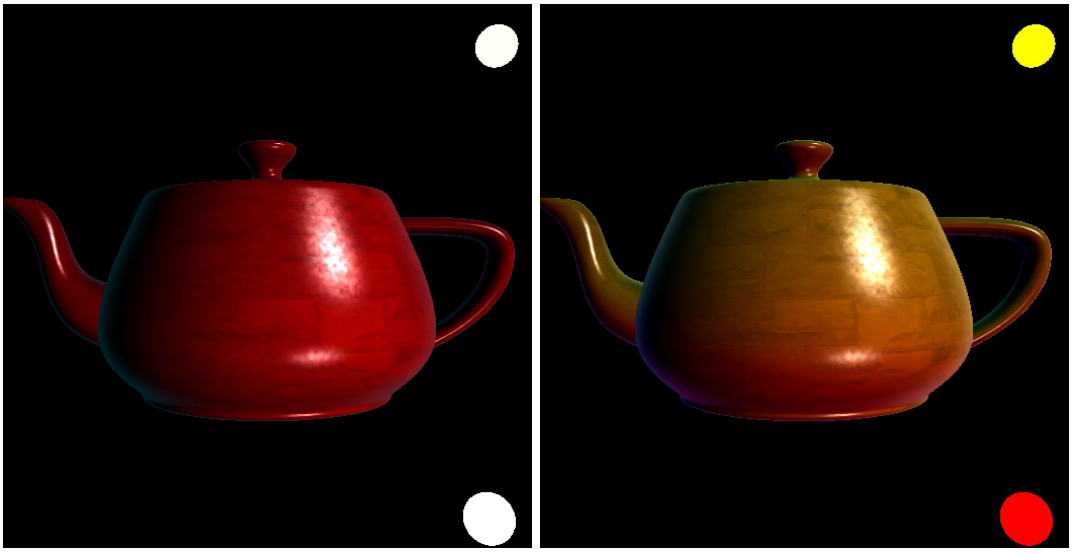
Johan Lavén
d15johla@student.his.se

Introduktion

När det gäller shading i OpenGL så finns det två olika sätt att göra det på. Det ena är vertex shading som fungerar så att all information man programmerar appliceras på varje vertex, dvs varje hörn av varje polygon. Informationen delas därefter ut som en gradient mellan informationen från de olika vertexarna. Det andra sättet att att programmera en shader är genom fragment shading. Dvs att varje pixel av varje polygon gör sin egen uträkning för att ge önskat resultat. Om man ställer dessa två metoder mot varandra så har de tydliga för och nackdelar. Några av dessa för och nackdelar är till exempel att vertex shading är betydligt billigare för hårdvaran då den koden körs en bråkdel så ofta som en fragment shader. En fördel med fragment shading över vertex shading kan vara att man har mer kontroll över detalj då informationen inte behöver baseras på hörnens information. I praktiken så blandas ofta dessa två metoder för att få en relativt billig shader som fortfarande fyller de estetiska behov som efterfrågas.

Designval och implementation

I Scenen tillsammans med tekannan så existerar två ljuskällor på två olika platser. Dessa ljuskällor ska lysa upp tekannan med varsitt färgat ljus. Tanken är att endast färgerna från ljuskällorna ska synas på kannan ifall kannans egna färg färg motsvarar det som lyses på den. Detta i syfte för att simulera verkligheten genom att alla våglängder ljus absorberas förutom de som reflekteras av ytan. Med denna metod så skulle jag kunna lysa med endast vita lampor på en blå kanna och kannan skulle förbli blå trots de andra färgerna som finns i det vita ljuset.



Figurer:
Den vänstra tekannan är röd med dubbel vit belysning.
Den högra tekannan är vit med belysning i rött och gult.

En annan detalj med ljussättningen är att ju längre ifrån ljuskällan en viss del av kannan är desto svagare ska den delen påverkas av de externa ljuskällorna. Det vill säga att ifall tekannans hällpip är placerat på bortre sidan jämfört med ljuskällorna så ska den vara mindre

belyst i jämförelse med handtaget som är på den belysta sidan. Tekannan ska heller inte vara belyst alls på de sidor som riktas från de båda ljuskällorna.

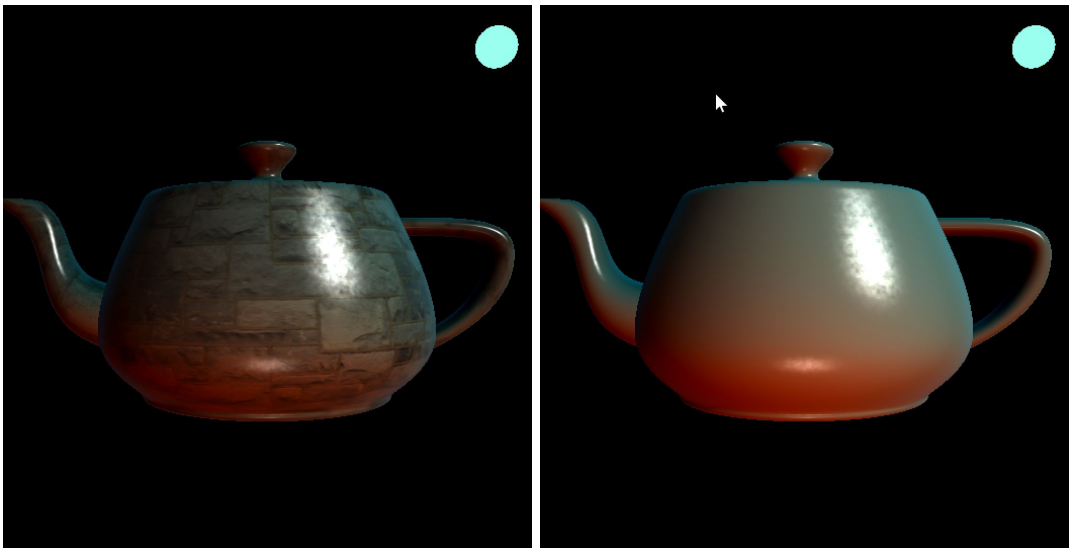
```
vColor=lightMultiplier*(((uLight2Color/light2L)*diffuse2)+
        ((uLight1Color/light1L)*diffuse1))*diffuseTeaPotColorColor;;
```

Figur: Kod som slår ihop ljuskällor med färger(uLight1Color) och delar dem med distansen till de båda ljuskällorna.

Figurer:
Figuren till höger har ljuskällorna hälften så långt ifrån jämfört med figuren till vänster, detta resulterar i en mer upplyst tekanna.

Även spekulera reflektioner ska baserad på de två ljuskällorna och deras relativa positioner till tekannan samt en kantbelysning som artificiellt ljussätter tekannans kanter för en mer estetiskt tilltalande effekt, denna effekt förankras inte i verklig effekt.

Det ska också finnas en variabel som bestämmer hur mycket av texturen ska påverka slutresultatet. Detta avgör i stora drag om kannans egna diffusa färg ska vara dominant eller texturen som mappats på dess yta.



Figurer: Kannan till vänster illustrerar effekten när den mappade texturen är helt dominant medans kannan till höger har sin egna diffusa färg som dominant. I bägge fall så påverkas slutresultatet till stor del från den belysning som sker i scenen.

```
vec4 finalOutputColor = mix(base2D*vColor.xyz, vColor.xyz, TextureOrDiffuse);
```

Figur: Raden ovan visar sammanslagningsmetoden för att välja mellan texturen eller den diffust färgade ytan. Variabeln *TextureOrDiffuse* ska ha ett värde mellan 0 till 1.

För- och Nackdelar med ljusapproximationer

Det finns många olika sätt att beräkna hur ljus reflekteras på föremål. Vissa metoder är mer kostsamma att beräkna medans andra är billigare. Det är i många fall en balansgång om målet är att simulera verkligt ljus då man för ett så realistiskt ljus som möjligt kräver många mätpunkter, vilket resulterar i fler draw calls. Man kan uppnå fortfarande uppnå relativt realistiska ljus och reflektioner genom att göra uppskattningar. Till exempel så kan man beräkna färre gånger totalt, till exempel på varje vertex istället för att göra mätningen per pixel. Resultatet man får i de fall man beräknar per vertex ger ett aningen kantigare ljus och skuggning då ljuset graderas från hörn till hörn per polygon.

En billig metod som fortfarande har möjligheten att ge bra resultat är något som kallas för en *lightmap* eller "bildbaserad upplysning" (*OpenGL shading language: 2009)* . Denna metod går ut på att man med hjälp av en fördefinierad textur beskriver vilka delar av modellen eller scenen som ska lysas upp och vad som ska skuggas med hjälp av de olika färg kanalerna.

RenderMan kontra RenderMonkey

Den första skillnaden mellan att programmera shaders i RenderMan och RenderMonkey är att shaderspråket som används skiljer sig. I RenderMonkey så programmerar man renderingspipelinen i språket GLSL *(OpenGL shading Language)*. Om man jämför det med språket RSL *(Renderman Shading Language)* så är användandet snarlikt.Färger representeras som *colors* och koordinater avbildas i form av datatypen punkter *(point)* i RSL medans man använder vektorer *(vec3, vec4)* i GLSL. Utöver namnskillnader på datatyper så har språken ett gemensamt syntax till stor del och medlemsfunktioner fungerar i princip på samma sätt.

En annan mer betydelsefull skillnad är att openGL använder sig av renderingspass samt att renderingspipelinen skiljer på vertexshading *(shading per polygonhörn)* och fragmentshading *(shading per pixel)*. Med denna uppdelning så har man mer kontroll över hur hårdvaran i den renderande datorn belastas. Detta gör det då möjligt att programmera shaders på ett mer prestandaeffektivt sätt då vissa delar av shaders kan bli optimerade för prestanda i utbyte mot lite kontroll eller ett lite grövre slutresultat.

I och med att RenderMonky använder sig av openGL och att det körs i realtid betyder det att man kan se resultatet av parameterförändring direkt utan att behovet för omkompilering finns där. Detta gör det smidigare att finslipa och förändra slutresultatet av en rendering utan att vänta på en kompilering som kan ta tid beroende på hur mycket information som ska beräknas. Denna realtidsrendering möjliggör även rotation av objektet samt förflyttning av ljuskällor för att se hur den specifika shader ser ut i lite andra förhållanden.

En fördel med RSL över GLSL är att språket bygger upp fler polygoner vid behov. Alltså kan modeller i scenen definieras som ett plan men att shadern själv lägger till extra polygoner vid behov. Ett exempel på detta är att utgå från ett plan och att genom endast användning av shaders modifiera det till formen av ett landskap.

Referenser

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