Shaderprogrammering

Uppgift 9

Hårdvarushader

Screen Space printed effect

Johan Lavén
d15johla@student.his.se

Introduktion

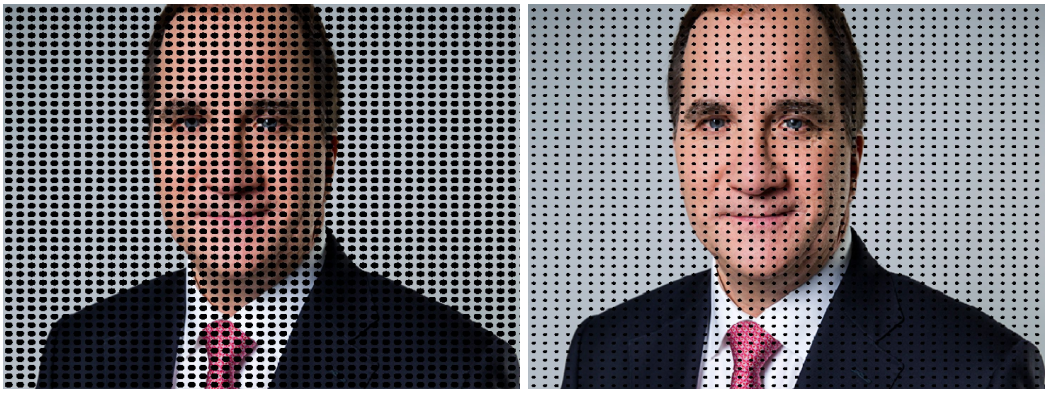
I det sista momentet av kursen var målet att på egen hand utveckla en shader med de tekniska verktyg och algoritmer som vi lärt oss under kursens gång. På så vis kan tekniker slås ihop och experimenteras med. Målet med shadern som kommer att diskuteras nedan i rapporten är att efterlikna en tryckt bild, lite som hur bilder trycktes i tidiga färg eller serietidningar. Denna shader är byggd enbart i fragment shadern och fungerar därför lite som ett instagram eller snapchat filter på inkommande data. I exemplen nedan så kommer shadern att appliceras mot ett porträtt.

Designval och implementation

Tanken var att shadern skulle agera lite som ett fotofilter. Den första effekten som skulle implementeras var prickar som med jämna mellanrum applicerades på tryckta bilder för att spara färg. Metoden som användes för att simulera den effekten var att med hjälp av V och U-koordinaterna plocka fram absolutbeloppen från två sinuskurvor, en i vardera led. De ränder som då uppstod går att slås ihop för att få upprepade fyrkanter i bilden. Genom att slänga in utvärdet från sinusoperationerna i en *pow(a, b)*-funktion så får de resulterande fyrkanterna rundade hörn. Beroende på vilket värde som slängs in i "*Pow()*"-funktionens b-parameter så manipuleras formen och storleken på prickarna. Sista beräkning som applicerades på prickarna var att göra övergången till prickarna helt kantiga, då detta saknas i resultatet på grund av sinusfunktionens mjuka natur. Denna effekt skapades med hjälp av *step*-funktionen .

```
float dotz = abs(sin(texCoord.x*amountOfDots));
dotz *= abs(sin(texCoord.y*amountOfDots));
dotz = pow(dotz, dotSize);
dotz = step(0.2, dotz);
```

Figur: Kodimplementation av flyttalsresultat som används vid prickarnas uträkning vid senare skede.



Figurer: När variabeln *dotSize* ökar så minskar prickarna samt deras form blir rundare.

Nästa effekt som skulle uppnås var att begränsa antalet färger som kunde visas.

```
vec4 outCol = orgCol;
outCol.x = floor((outCol.x * colorDepth)+0.5)/colorDepth;
outCol.y = floor((outCol.y * colorDepth)+0.5)/colorDepth;
outCol.z = floor((outCol.z * colorDepth)+0.5)/colorDepth;
```

Figur: Kod som illustrerar hur färgdjupet begränsas genom att på varje färgkanal.

Denna metod gjordes två gånger med lite värde på *colorDepth*-variabeln. Detta i syfte att färglägga prickarna i en färg som är liknande men inte likadan mot bakgrunden. Skillnaden mellan dessa två färgdjupsbegränsningar går att styra med en skalär i rendermonkey.

Figur: Illustration som visar på hur färgdjupet är lite annorlunda inuti prickarna jämfört mot utanför.

Det sista effekten som skulle appliceras var en typ av ifyllnad där det existerar skarpa kanter. Inte för att det efterliknar hur en tryckt bild ser ut, utan effekten som eftersträvas är mer av att estetiskt experiment. För att skapa dessa kanter så användes en variant av sobels algoritm. (YouTube, Computerfile, 2018). Eftersom det blir effekter då man beräknar varje färgkanal för sig så justeras först originalbilden till att vara svartvit, och att *sobel*-effekten beräknas i ett pass senare för sammanslagning med tidigare nämnda effekter.

Figur: Bilden som *sobels algorithm* appliceras på.

Det första resultatet från kantuträkningen innehåller mycket brus där det egentligen inte är en kant så som vi människor ser kanter. Kanterna är också utritade som vita mot en svart

bakgrund, detta är motsatt det effekt som vi vill åstadkomma. På grund av dessa fenomen sker lite uppstädning av slutresultatet innan det appliceras mot resultatet från färgdjups- och prickeffekterna.

Figurer: Vänster: Resultatet från sobel-samplingen.
Mitten: Inverterade färger för att ge svarta kanter.
Höger: Det naturliga bruset är uppstädat.

```
vec2 G = vec2(0.0, 0.0);
//Sampla lodräta kanter
G.x += texture2D(picture, texCoord + vec2( ofset.x, ofset.y)).x;
G.x -= texture2D(picture, texCoord + vec2(-ofset.x, ofset.y)).x;
G.x += texture2D(picture, texCoord + vec2( ofset.x,       0)).x;
G.x -= texture2D(picture, texCoord + vec2(-ofset.x,       0)).x;
G.x += texture2D(picture, texCoord + vec2( ofset.x,-ofset.y)).x;
G.x -= texture2D(picture, texCoord + vec2(-ofset.x,-ofset.y)).x;

//Sampla vågräta kanter
G.y += texture2D(picture, texCoord + vec2( ofset.x, ofset.y)).x;
G.y -= texture2D(picture, texCoord + vec2( ofset.x,-ofset.y)).x;
G.y += texture2D(picture, texCoord + vec2( 0,       ofset.y)).x;
G.y -= texture2D(picture, texCoord + vec2( 0,      -ofset.y)).x;
G.y += texture2D(picture, texCoord + vec2(-ofset.x, ofset.y)).x;
G.y -= texture2D(picture, texCoord + vec2(-ofset.x,-ofset.y)).x;

float finalEdges = 1 - sqrt(pow(G.x, 2)+pow(G.y,2));
finalEdges = smoothstep(0.4, 0.8, finalEdges);
vec4 SobelCol = vec4(finalEdges,finalEdges,finalEdges, 1.0);
```

Figur: Koden visar hur sampling för beräkning av kanter sker. De tre nedersta raderna visar på hur uppstädning av koden gick till. Refaktorering av denna kod skulle skapa samplingen inuti en samplingsloop.

För att slå ihop kanten med det tidigare resultatet användes vanlig multiplikation, Detta då tanken var att där det finns kanter, där ska bilden förbli svart.

Figur: Slutresultat av multiplikativ sammanslagning.

Problem och Lösningar

Det största problem av de som stöttes på under utvecklingen av denna shader var att lista ut hur *edge detection* (kantidentifiering) fungerar. Efter lite informationssökande fann jag information om en algoritm känd som *Sobel operation, Sobel detection* eller *Sobel algorithm*. Mitt mål med denna var att inte ta reda på hur den fungerar genom att läsa andra personers kod och lista ut det på så sätt. Det jag gjorde var att jag kollade en video som beskriver i detalj hur algoritmen fungerar utan att någon kod förklaras. Denna videon var publicerad på youtube.com under titeln *Finding the Edges (Sobel Operator)* av Computerphile. på flera olika sätt försökte jag implementera samplingen i form av nestade och onestade loopar, men det var struligt och gav inte den effekt jag var ute efter. Därför skrev jag koden kontinuerligt tillfälligt för i detta fall var resultatet viktigare än läsbarheten i koden.

Referenser

YouTube. (2015). *Finding the Edges (Sobel Operator) - Computerphile*. [online] Available at: https://www.youtube.com/watch?v=uihBwtPIBxM [Accessed 9 Mar. 2018].