Rapport i Avancerad Datorgrafik

# Uppgift 1

# P:\GitHub\AdvancedComputerGraphics\Uppgift 1\high-res ball.tif

## Kod

### enkelshader.sl

surface enkelshader(

color purple = (0.5, 0.0, 0.6); // Purple color

color red = (0.6, 0.2, 0.0); // Red color

color white = (1.0, 1.0, 1.0); // White color

float frequency = (4.0); // Frequency of meridians (stripes) in the sphere

float diffuseStrength = (0.9); // Value defining the intensity of the light being shed on the object

float roughness = (0.1); // The level of irregularity

float bumpheight = (0.8); // The depth of the texture bumps

float f = (0.1); // The interval (size) of the white area

)

{

float segments = mod((s \* frequency), 1.0); // Iterate frequencies through the meridians (s) to create sphere segments

float alignedMeridian = mod((s \* frequency + 0.125), 1.0); // Additional frequency used to correctly align with the meridians

// Smoothly transition from 0.25 units until 0.5 units at each frequency

// First value indicates the end of solid color (from 0.0)

// Second value indicates end of smoothed color, i.e. when that color ceases

float f1 = smoothstep(0.25, 0.5, segments);

float f2 = smoothstep(0.75, 1.0, segments);

float f3 = smoothstep((f - 0.065), f, v);

// Calculate a new normal by altering the global view point and create a noise effect coordinated using bumpheight

P += step(0.5, alignedMeridian) \* (noise(P \* frequency) \* N \* bumpheight);

N = calculatenormal(P);

// Calculate the normal of the shape

normal normalValue = normalize(N);

// Create a base color mix consisting of red and purple, smoothly blending them using the f1 and f2 smoothstep intervals

// Create a new color mix, adding white to the base mix amd switching/blending between the interval of the third parameter f3

color baseColor = mix(red, purple, f1-f2);

color whitedBaseColor = mix (white, baseColor, f3);

// Diffuse the color to a defined intensity by multiplying it with a diffuse function

color diffusedColor = mix (whitedBaseColor, whitedBaseColor \* diffuse(normalValue), diffuseStrength);

// Rendering instruction

Ci = diffusedColor;

Oi = Os;

}

## Reflektion

En stor del av min ansats till att strukturera detta, för mig, något förvirrande och stundvis oregelbundna programmeringsspråk finner sitt ursprung i formella och informella C-konventioner. Det förefaller mig ovant att med en spontan attityd arbiträrt deklarera konkreta värden i min kod, snarare än att samla dem i en gemensam globalt tillgänglig yta eller konstruktor. Sistnämnda alternativ är dock i hög grad en välkomnad möjlighet när det anses lämpligt. Av den anledningen strävar jag efter en välfylld parameterlista i början av min shader.

I parameterlistan finner vi färdigdefinierade färger med logiskt representerade variabelnamn; lila, orange och vitt. Detta för att underlätta framtid färghantering. Att tilldela, blanda eller använda en färg blir betydligt enklare och tydligare när färgens RGB-kod representeras av ett variabelnamn.

Vi finner även en rad flyttalsvärden: *frequency*, *roughness*, *bumpHeight*, *diffuseStrength* och det något intetsägande *f*.

* Värdet representerat av *frequency* påverkar hur många segment (meridianer) sfären ska delas upp i. 4.0 innebär att vi får fyra linjer som delar sfären i åtta segment.
* Värdet representerat av *roughness* påverkar hur kraftig faktor av oregelbundenhet som bör tillämpas på de ytor som tilldelas en ojämn yta.
* Värdet representerat av *bumpHeight* påverkar hur djupa försjunkningar den ojämna ytan skall ha. Ett högre värde innebär djupare ojämnheter.
* Värdet representerat av *diffuseStrength* påverkar intensiteten av det diffusa ljuset som kastas på det renderade objektet.
* Värdet representerat av *f* påverkar hur stor omfattningen intervallet på en smoothstep som renderar en vit ”pol” på sfären ska ha. Ett högre värde innebär i praktiken att den vita ytan blir större/sträcker sig längre från sin egna origo.
* Värdet representerat av *segments* itererar genom meridianerna för att skapa segment ur sfären.
* Värdet representerat av *alignedMeridian* är en extra frekvens som ”stöd” för den ursprungliga *frequency*-variabeln. Den används för att ge en korrekt justerad frekvens mot meridianerna.

## Procedurella Shaders kontra Färdiga Texturer

Det finns en rad olika fördelar med procedurellt beräknade texturer framför målade/ritade/skannade/fotograferade texturer. Flera av dessa anspelar på behovet av dynamisk förändring. En statisk bild är absolut i sitt utseende, något som inte alltid är önskvärt.

En procedurellt skapad textur kan förstoras, förminskas, förvrängas, lappas sömlöst och ökas respektive minskas i sin upplösning på vis som vore omöjliga för en statisk bild. Man kan även med fördel justera enstaka parametrar för att få varierande och multilaterala representationer av sin textur, samtidigt som bilderna tar minimal fysisk plats på hårddisken i kontrast till en mängd statiska bilder. Allt detta är givetvis endast fördelaktigt om det är värt besväret att skapa en helt egen shader för en enda textur. Det krävs med andra ord en förutsättning om att texturen behöver nå krav på flexibilitet och dynamik som berättigar det faktiska arbetet.

Det är inte alltid smidigare att skapa en specifik bild än att ”programmera” en textur. En statisk bild behöver eventuellt en konstnär, utrustning, material och tid för att utföra en rad olika processer. Dessutom kommer resultatet inte kunna ta hänsyn till saker som vilken ljussättning bilden bör förhålla sig till om detta inte är specificerat till extrem detaljnivå. De olika typerna av ljus och deras egenskaper är en handfull vetenskaper av sig själva, och inte ens de mest avancerade (eller tidskrävande) post-produktionseffekterna kan konkurrera med förmågan att anpassa samtliga faktorer på plats. Däremot saknar procedurella texturer förmågan att på ett tidseffektivt vis ändra och skapa övertygande och precisa detaljer. När en konstnär lätt kan lägga till, ändra eller ta bort exempelvis en fläck kan det vara mycket svårare att programmera en så pass skarp specifikation.

# Uppgift 2

P:\GitHub\AdvancedComputerGraphics\Uppgift 2\galaxy.tif

## Kod

### stars.sl

#define snoise(p) (2 \* (float noise(p)) - 1)

float fBm (point p; uniform float octaves, lacunarity, gain)

{

uniform float amp = 1;

varying point pp = p;

varying float sum = 0;

uniform float i;

for (i = 0; i < octaves; i += 1) {

sum += amp \* snoise (pp);

amp \*= gain;

pp \*= lacunarity;

}

return sum;

}

// The Worley cell noise function creates cells and measures the distance between them,

// offsetting them by random values. This creates an irregular "noisy" cell pattern.

// We utilize these scattered cells by inverting and narrowing their color values, creating a realistic

// and spectacular night sky texture with distant, glowing stars.

void voronoi\_f1f2\_2d (float ss, tt; output float f1; output float spos1, tpos1; output float f2; output float spos2, tpos2;)

{

float jitter=1.0;

float sthiscell = floor(ss)+0.5;

float tthiscell = floor(tt)+0.5;

f1 = f2 = 100;

uniform float i, j;

for (i = -1; i <= 1; i += 1) {

float stestcell = sthiscell + i;

for (j = -1; j <= 1; j += 1) {

float ttestcell = tthiscell + j;

float spos = stestcell + jitter \* (cellnoise(stestcell, ttestcell) - 0.5);

float tpos = ttestcell + jitter \* (cellnoise(stestcell+23, ttestcell-87) - 0.5);

float soffset = spos - ss;

float toffset = tpos - tt;

float dist = soffset\*soffset + toffset\*toffset;

if (dist < f1) {

f2 = f1;

spos2 = spos1;

tpos2 = tpos1;

f1 = dist;

spos1 = spos;

tpos1 = tpos;

}

else if (dist < f2) {

f2 = dist;

spos2 = spos;

tpos2 = tpos;

}

}

}

f1 = sqrt(f1); f2 = sqrt(f2);

}

surface stars(

color white = (1.0, 1.0, 1.0); // White color

color grey = (0.5, 0.5, 0.5); // Grey color

color black = (0.0, 0.0, 0.0); // Black color

color red = (1.0, 0.0, 0.2); // Red color

color blue = (0.2, 0.0, 1.0); // Blue color

float starlightIntensity = (1.0); // Modifies the intensity of the starlight

float lacunarity = (6.0); // Level of "gappiness" or inhomogeneity

float octaves = (2.0); //

float gain = (0.4); //

)

{

// Initiate variables to be used as parameters for the Worley Noise Cell algorithm

float f1;

float spos1, tpos1;

float f2;

float spos2, tpos2;

// Execute the Worley Cell Noise algorithm

voronoi\_f1f2\_2d(s\*0.1, t\*0.1, f1, spos1, tpos1, f2, spos2, tpos2);

// Create two smoothsteps to manage the strength and fade of the starlight

// The second smoothstep will take an in-parameter to control the total light strength

float f1a;

float f1b;

f1a = smoothstep(0.05 \* starlightIntensity, 0.15, f1);

f1b = smoothstep(0.05 \* starlightIntensity, 0.8, f1) \* 0.5;

// Sum and invert the two smoothsteps to gain a potent center light for effect

f1 = 1 - (f1a+f1b);

// Darken the color range by applying a gray color

color newColor = f1 \* grey;

// Mesh the stars to give the scene more realism

float fBm1 = clamp( abs(fBm(P\*0.05, octaves, lacunarity, gain) ), 0, 1);

newColor = (newColor \* fBm1);

// Add red and blue nebula clouds and neatly layer them together

// Mesh the clouds to give them a naturally irregular appearance

// Remember to move the point P in different directions to spread the mesh layers

color fBm2a = clamp(fBm((P+231)\*0.05, 5, lacunarity, gain), 0, 1)\*red;

color fBm2b = clamp(fBm((P-231)\*0.05, 5, lacunarity, gain), 0, 1)\*blue;

newColor = (newColor + fBm2a + fBm2b);

// Finalize the output by adding the color

Ci = Cs \* newColor;

Oi = Os;

}

### galaxy.sl

surface galaxy(

float reachFrequency = (12.0); // The diameter/"reach" of the spirals

float ovalFactor = (3.5); // How intensely the spirals are twisted

float galaxySize = (0.5); // The size/diameter of the final galaxy

color galaxyColor = (0.0, 0.4, 0.045); // The color of the galaxy

)

{

// Create alternative coordinates depending on u and v.

float uu = (u - 0.5) \* reachFrequency;

float vv = (v - 0.5) \* reachFrequency;

// Create two points, one center (origo) and one at the alternative coordinates uu and vv.

point p1 = point(uu, vv, 0);

point p2 = point(0, 0, 0);

// Calculate the distance between the origo point (p1) and the new point (p2).

float dist = distance(p1,p2);

// Create spiraling stripes from the center towards the edges of the texture.

// atan(uu, vv) is utilized to create the curved lines.

float spirals = step(1.2, mod(atan(uu, vv \* ovalFactor) + dist, 1.57));

// Smoothstep the spirals to fade out the stripes toward the edges

spirals = spirals \* (clamp(1 - (smoothstep(1.0 \* galaxySize, 4.0 \* galaxySize, dist)), 0, 1));

// Add a glowing "haze", filling the galaxy with a lit substance

spirals = spirals + clamp(1 - (smoothstep(0.8 \* galaxySize, 3.5 \* galaxySize, dist)), 0, 1);

// Finalize the output by adding color

Ci = spirals \* galaxyColor;

Oi = spirals;

}

## Väsentliga Saker i Shaderkoden

*Inga stora mängder text utan de viktigaste ca 10 raderna (som kan vara på olika ställen i koden) skall beskrivas med mellan en halv till högst en sida totalt inklusive citerade programkodsrader.*

<http://imagej.nih.gov/ij/plugins/fraclac/FLHelp/Lacunarity.htm>

## Låt Oss Leka med Lite Värden

[Klicka här för att se en animerad .gif av de olika bilderna.](http://imgur.com/K05WSUs)

P:\GitHub\AdvancedComputerGraphics\Uppgift 2\basic.tif

Detta är en oförändrad version av rymdlandskapet. Galaxen är grön och relativt liten.  
Stjärnorna är diskreta i sin storlek och nebulosorna är konsistenta och molnliknande.

P:\GitHub\AdvancedComputerGraphics\Uppgift 2\2_starlightIntensity.tif

I denna version är parametern starlightIntensity ökad från 1.0 till 2.0 vilket innebär att stjärnljuset, som kontrolleras av en smoothstep, får ett högre tröskelvärde. Detta gör att stjärnorna förefaller lysa starkare eller vara större än i ursprungsbilden.

P:\GitHub\AdvancedComputerGraphics\Uppgift 2\6_lacunarity.tif

**Här är parametern** lacunarity **ökad från 2.0 till 6.0. Detta gör att frekvensen i mönstrets ”luckor” som maskar nebulosamolnen blir tätare. Resultatet blir att molnen förlorar sin oslipade mjukhet och istället får en tydligare definierad grynighet.**

P:\GitHub\AdvancedComputerGraphics\Uppgift 2\1.0_red_galaxyColor.tif

Här har galaxyColor som styr galaxens färg ändrats från (0.0, 0.4, 0.045) till att inkludera ett fullt rödvärde: (1.0, 0.4, 0.045). Resultatet blir helt enkelt att galaxen blir orangefärgad i sitt utseende.

P:\GitHub\AdvancedComputerGraphics\Uppgift 2\1.5_galaxySize.tif

Här har parametern galaxySize ökats från 0.5 till 1.5. Parametern agerar som en faktor i den smoothstep som används för att skapa galaxens spiraler och ”dimma”. Genom att öka värdet på faktorn ökar vi även den konkreta storleken på galaxen.

## Problem som kan Uppstå med Aliasing i Procedurella Texturer

*I rapporten skall det även finnas en kort diskussion (max en halv sida) kring de problem som kan uppstå med aliasing i procedurella texturer. I boken på sidan*[*263-268*](http://books.google.se/books?id=6_4VaJiOx7EC&lpg=PA537&dq=shader%20antialiasing%20advanced%20renderman&pg=PA263#v=onepage&q=shader%20antialiasing%20advanced%20renderman&f=false)*beskrivs problemen med aliasing i detalj.*