Rapport i Avancerad Datorgrafik

# Uppgift 1

## Kod

surface enkelshader(

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

color orange = (0.6, 0.2, 0.0); // Orange 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 orange 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(orange, 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

Motivera valet av de konfigurerbara parametrarna

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.

## 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.