RenderMan and OpenGL Shaders

CS557

Chao Zhang

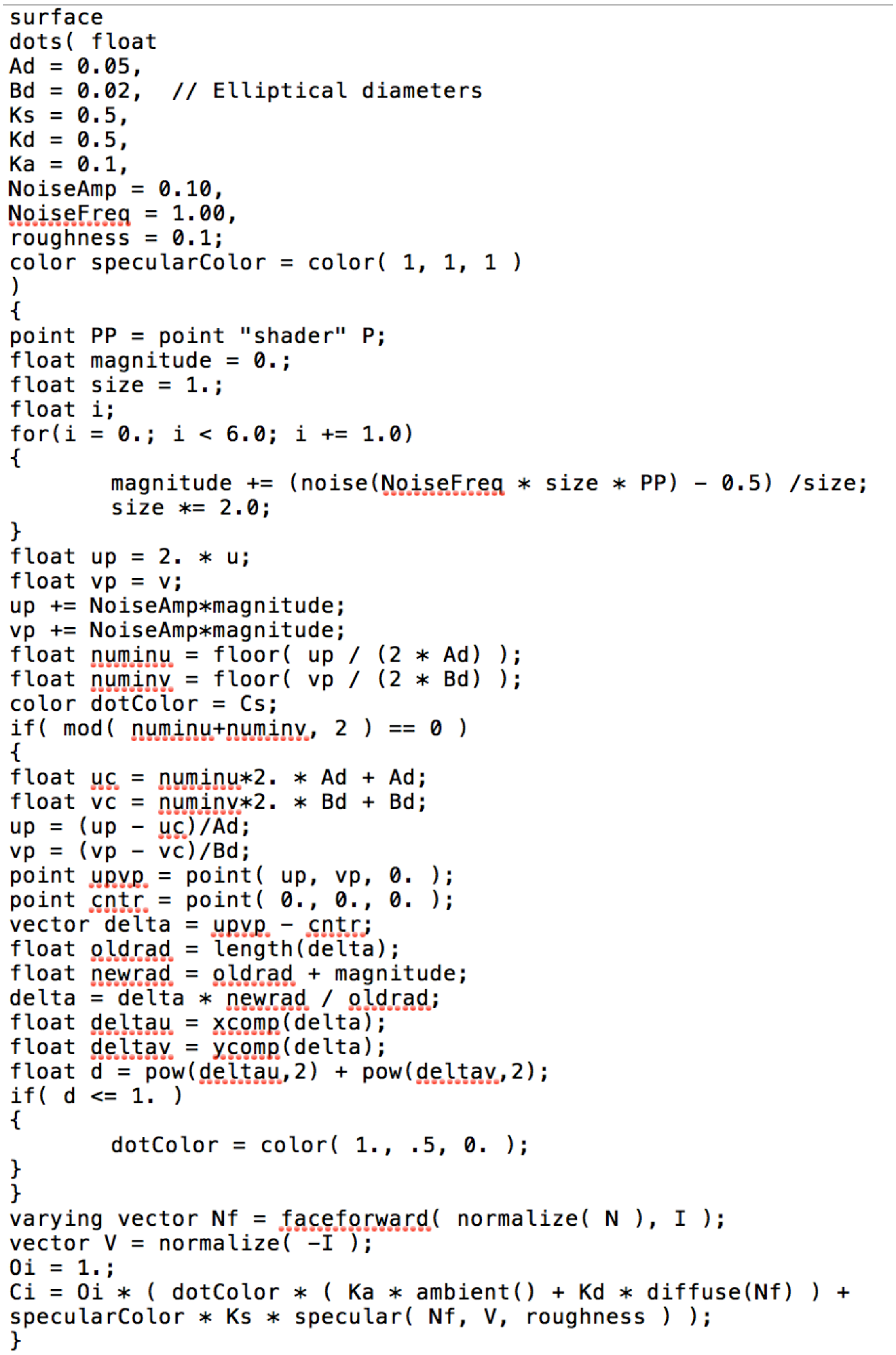
Project #2

1. Source listings

```
##RenderMan RIB
version 3.03

# declare the variables:

Declare "Ad" "uniform float"

Declare "Bd" "uniform float"

# define the output file:

Display "homework1.tiff" "file" "rgb"
Format 500 500 -1

# define the lighting:

LightSource "ambientlight" 1 "intensity" [0.25]
LightSource "distantlight" 2 "intensity" [0.75] "from" [5 8 -10] "to" [0 0 0]

# define the rendering parameters:

ShadingRate 1
Projection "perspective" "fov" [70]

# define the scene to be rendered:

WorldBegin
        Surface "dots"  "Ad" 0.05  "Bd" 0.02  "Height" 0.1 "NoiseAmp" 0.10 "NoiseFreq" 1.00
        Displacement "dotd"  "Ad" 0.05  "Bd" 0.02 "Height" 0.1 "NoiseAmp" 0.10 "NoiseFreq" 1.00
        # specify the surface shader
        Color   [1 5 4]                         # specify the Cs color
        Opacity [1 1 1]                         # specify the Os opacity
        TransformBegin
                Translate 0 0 6                 # move away from the eye (= gluLookAt)
                Rotate 90  1. 0. 0.             # rotate so don't see north pole
                Sphere 3 -3 3 360               # a full sphere
        TransformEnd
WorldEnd
```

Rib file

The different between this file and the project 1 rib file is this rib file use two shaders, the surface and the displacement. Those two shaders will be compiled separately. And all the color and lights is the same as the project 1.



```
surface
dots( float
Ad = 0.05,
Bd = 0.02,  // Elliptical diameters
Ks = 0.5,
Kd = 0.5,
Ka = 0.1,
NoiseAmp = 0.10,
NoiseFreq = 1.00,
roughness = 0.1;
color specularColor = color( 1, 1, 1 )
)
{
point PP = point "shader" P;
float magnitude = 0.;
float size = 1.;
float i;
for(i = 0.; i < 6.0; i += 1.0)
{
        magnitude += (noise(NoiseFreq * size * PP) - 0.5) /size;
        size *= 2.0;
}
float up = 2. * u;
float vp = v;
up += NoiseAmp*magnitude;
vp += NoiseAmp*magnitude;
float numinu = floor( up / (2 * Ad) );
float numinv = floor( vp / (2 * Bd) );
color dotColor = Cs;
if( mod( numinu+numinv, 2 ) == 0 )
{
float uc = numinu*2. * Ad + Ad;
float vc = numinv*2. * Bd + Bd;
up = (up - uc)/Ad;
vp = (vp - vc)/Bd;
point upvp = point( up, vp, 0. );
point cntr = point( 0., 0., 0. );
vector delta = upvp - cntr;
float oldrad = length(delta);
float newrad = oldrad + magnitude;
delta = delta * newrad / oldrad;
float deltau = xcomp(delta);
float deltav = ycomp(delta);
float d = pow(deltau,2) + pow(deltav,2);
if( d <= 1. )
{
        dotColor = color( 1., .5, 0. );
}
}
varying vector Nf = faceforward( normalize( N ), I );
vector V = normalize( -I );
Oi = 1.;
Ci = Oi * ( dotColor * ( Ka * ambient() + Kd * diffuse(Nf) ) +
specularColor * Ks * specular( Nf, V, roughness ) );
}
```

surface sl file

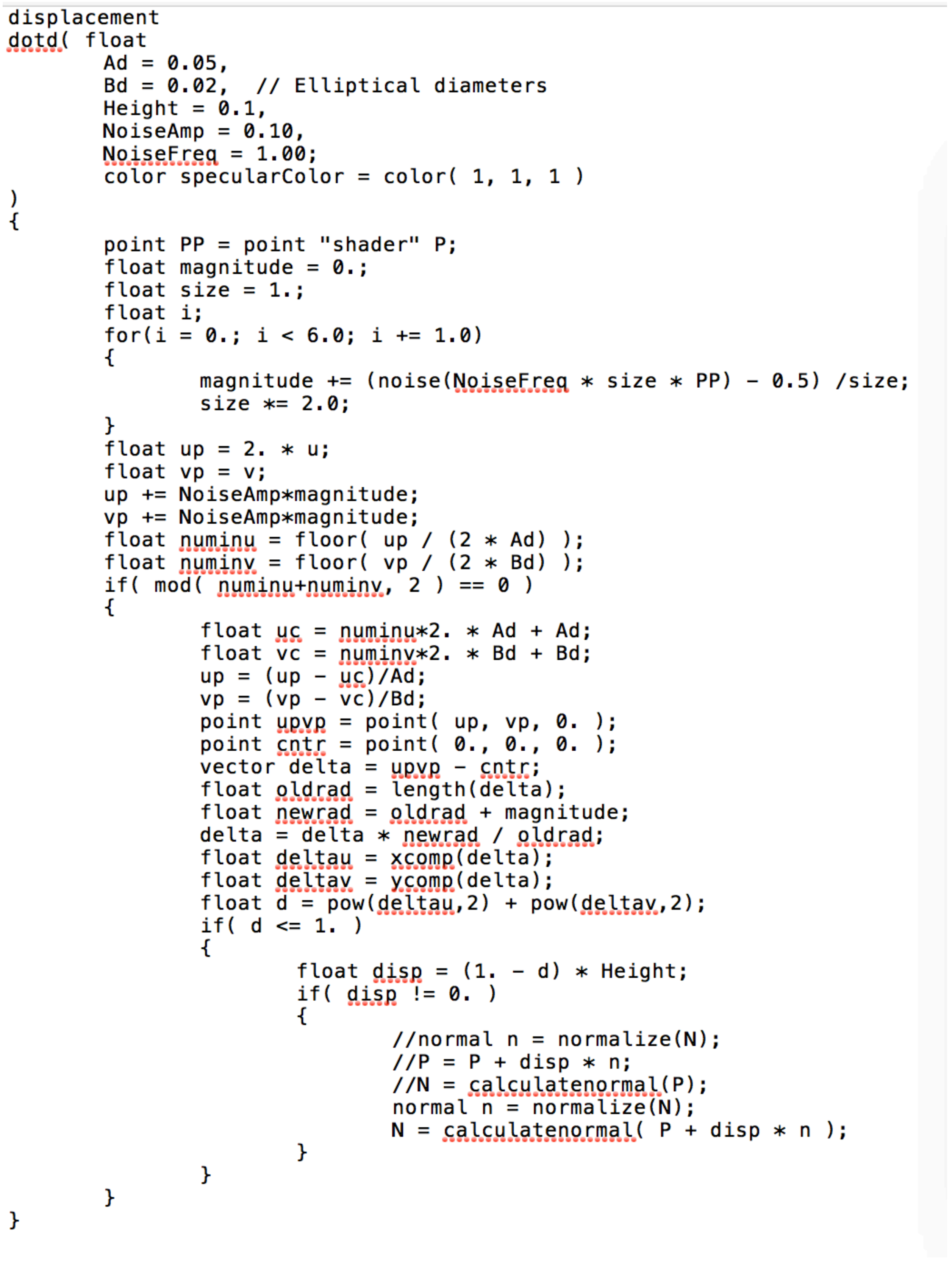
This sl file is for the surface shader. This code is based on the first project's sl file, I add the noise on it. For the noise, I set NoiseAmp and nosieFreq. magnitude += (noise(NoiseFreq * size * PP) - 0.5) /size is used to get the magnitude for of the noise the after this, this point will be changed by this magnitude as

up += NoiseAmp*magnitude;
vp += NoiseAmp*magnitude;

The code vector delta = upvp – cntr used to get the vertor from the center to the point(u, v). oldrad is the length of the vector and the newrad is the length of the point after noise to the center. Then I use delta = delta * newrad / oldrad to get the new vector.

deltau = xcomp(delta);
deltav = ycomp(delta);

Those two can get the position to the new point. With the new position of the point, we can get the distance from this point to center which is d = pow(deltau,2) + pow(deltav,2); Then I give the point the color of beaver orange.



```
displacement
dotd( float
        Ad = 0.05,
        Bd = 0.02,  // Elliptical diameters
        Height = 0.1,
        NoiseAmp = 0.10,
        NoiseFreq = 1.00;
        color specularColor = color( 1, 1, 1 )
)
{
        point PP = point "shader" P;
        float magnitude = 0.;
        float size = 1.;
        float i;
        for(i = 0.; i < 6.0; i += 1.0)
        {
                magnitude += (noise(NoiseFreq * size * PP) - 0.5) /size;
                size *= 2.0;
        }
        float up = 2. * u;
        float vp = v;
        up += NoiseAmp*magnitude;
        vp += NoiseAmp*magnitude;
        float numinu = floor( up / (2 * Ad) );
        float numinv = floor( vp / (2 * Bd) );
        if( mod( numinu+numinv, 2 ) == 0 )
        {
                float uc = numinu*2. * Ad + Ad;
                float vc = numinv*2. * Bd + Bd;
                up = (up - uc)/Ad;
                vp = (vp - vc)/Bd;
                point upvp = point( up, vp, 0. );
                point cntr = point( 0., 0., 0. );
                vector delta = upvp - cntr;
                float oldrad = length(delta);
                float newrad = oldrad + magnitude;
                delta = delta * newrad / oldrad;
                float deltau = xcomp(delta);
                float deltav = ycomp(delta);
                float d = pow(deltau,2) + pow(deltav,2);
                if( d <= 1. )
                {
                        float disp = (1. - d) * Height;
                        if( disp != 0. )
                        {
                                //normal n = normalize(N);
                                //P = P + disp * n;
                                //N = calculatenormal(P);
                                normal n = normalize(N);
                                N = calculatenormal( P + disp * n );
                        }
                }
        }
}
```

displacement sl file

The displacement file is almost like the surface file. This file have no code about the light or color. After noise, the points also need to have a height.
disp = (1. - d) * Height
the 1 – d can make it smooth. When the d is bigger, the result of 1- d will be smaller. This means the point near the center will higher than the point far from the center and it will be smooth.

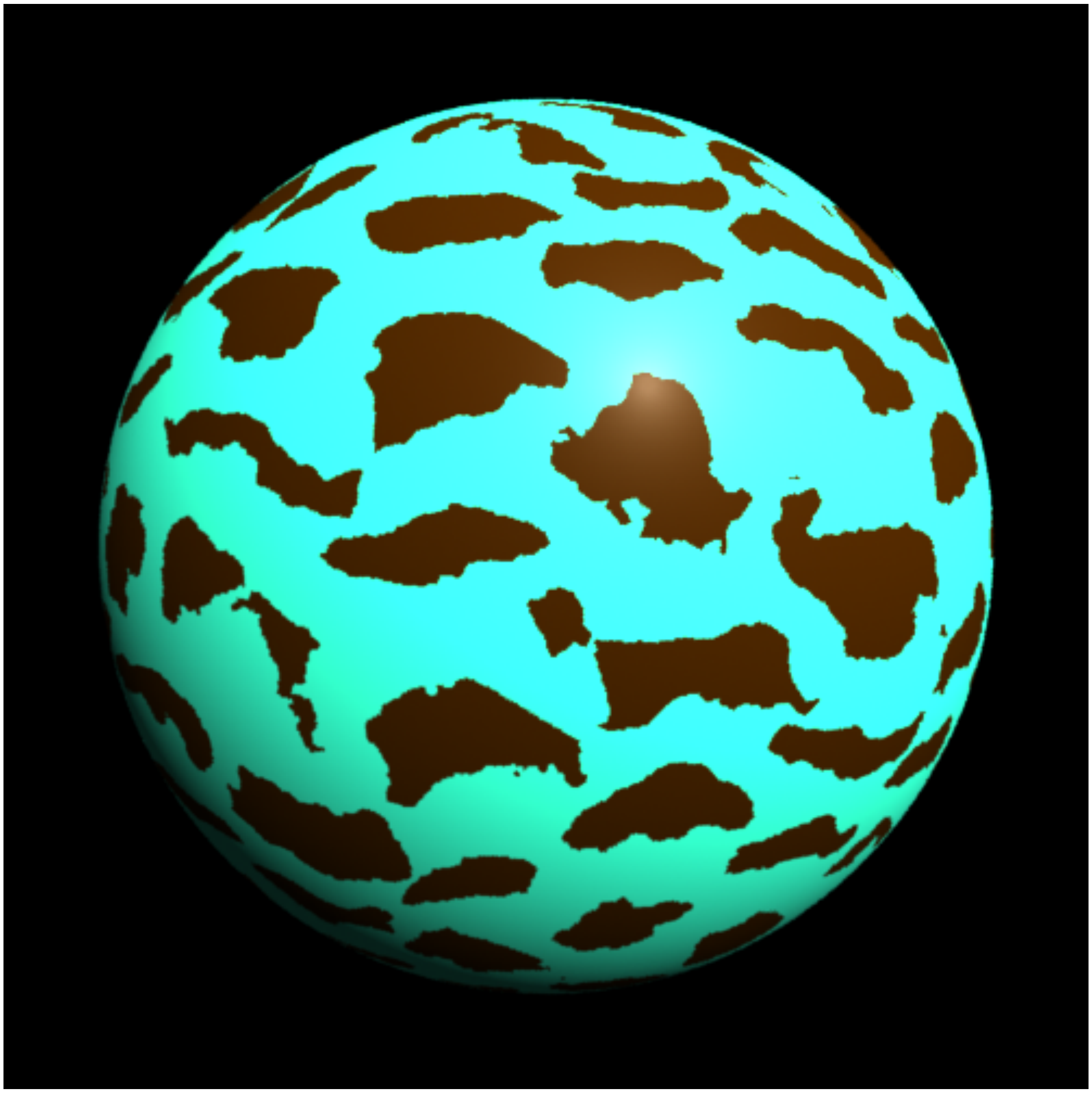
```
//normal n = normalize(N);
//P = P + disp * n;
//N = calculatenormal(P);
```

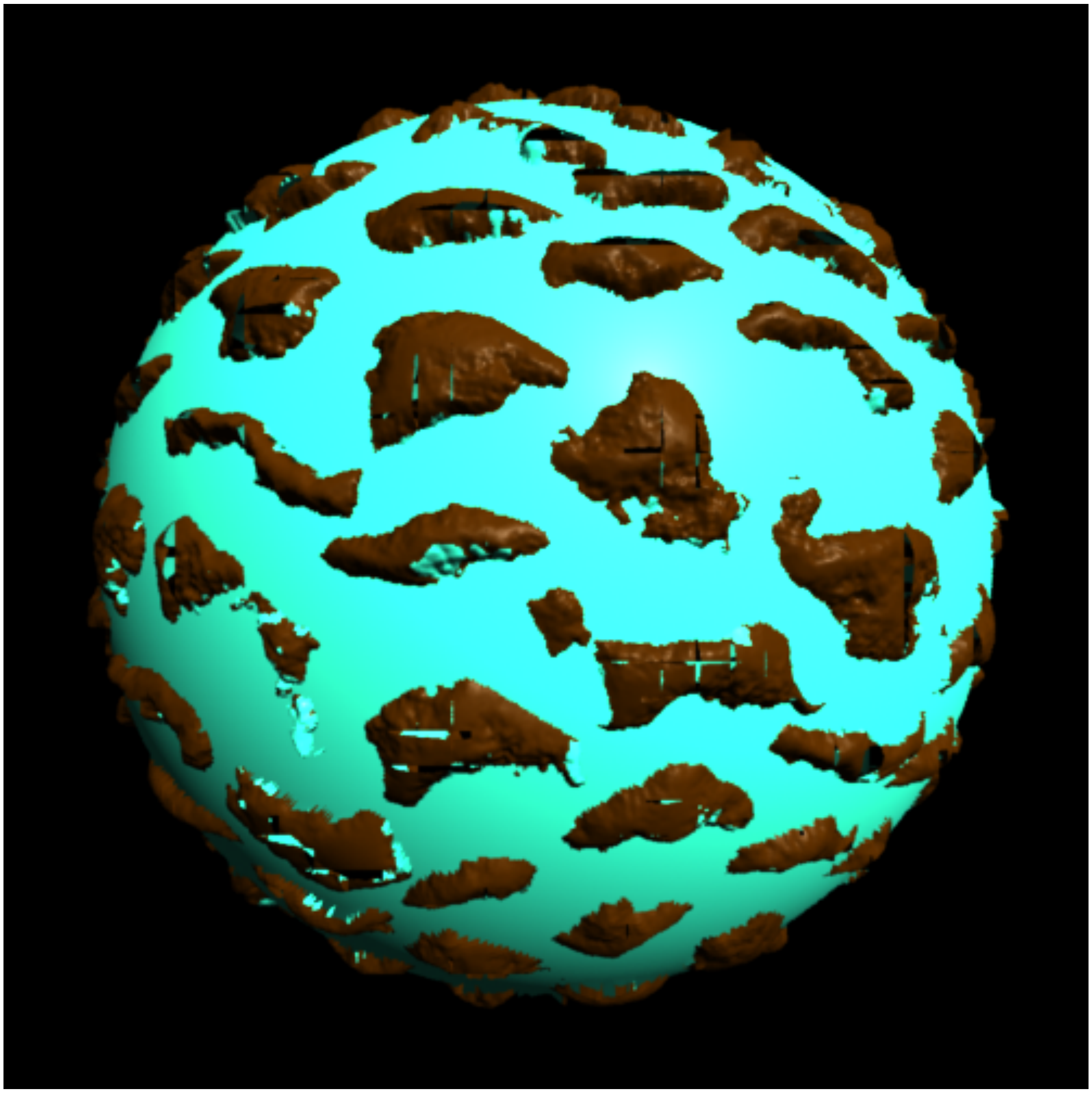
This will calculate the normal and then replace the normal.

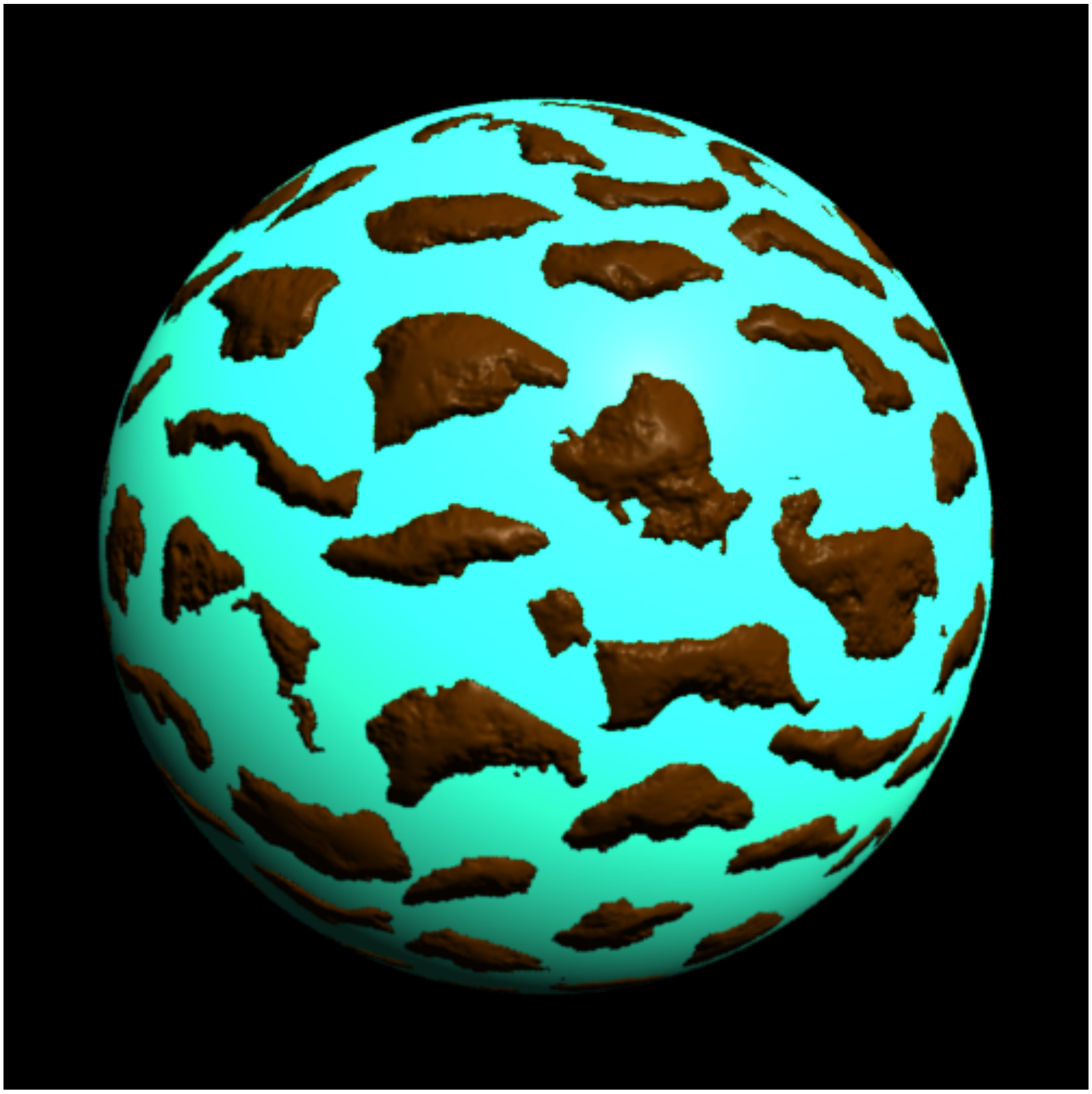
```
normal n = normalize(N);
N = calculatenormal( P + disp * n );
```

This will calculate the normal but not replace it, so this is the bump displacement.

2. Results


Surface only


Surface and displacement


Surface and bump displacement