Displacement Mapping, Bump Mapping, and Lighting

Margaux Masson

Winter 2018 - CS557 - Project 3

1 What I did and how it works

In this project we want to simulate a decaying cosine wave. In order to reproduce this shape using GLSL we need to implement a surface that spreads across X and Y, rising in Z, flowing in X, and decaying in X and Y separately according to this equation:

z = A * [cos(2Bx+C) * e-Dx] * [e-Ey]

So, the idea here is to implement a surface across x, y, and z which defined above.

- Vertex file:

First, I defined a variable z in the vertex file, its calculus derivatives, the tangent vectors and the normal as following:

```
  float x = gl_Vertex.x;
  float y = gl_Vertex.y;
  float z = uA * (cos(2*PI*uB*x+uC)*exp(-uD*x))*(exp(-uE*y)
      );

////////////// Getting the Normal ////////////
//  calculus derivatives
float dzdx = uA * (-sin(2*PI*uB*x+uC)*2*PI*uB*exp(-uD*x)+
     cos(2*PI*uB*x+uC)*(-uD)*exp(-uD*x))*(exp(-uE*y));
float dzdy = uA*(cos(2*PI*uB*x+uC)*exp(-uD*x))*(-uE*exp(-uE
    *y));

  // Tangent vectors
vec3 Tx = vec3(1., 0., dzdx);
vec3 Ty = vec3(0., 1., dzdy);

// Normal
vec3 normal = normalize(cross(Tx, Ty));
```

Then, I used the code from the Lighting course:

```
vec4 ECposition = gl_ModelViewMatrix * aVertex;
vNf = normalize(gl_NormalMatrix * normal);
vLf = eyeLightPosition - ECposition.xyz;
vEf = vec3(0., 0., 0.) - ECposition.xyz;

gl_Position = gl_ModelViewProjectionMatrix * aVertex;
```

and defined aVertex as following:

```
vec4 aVertex = vec4(x, y, z ,1.);
```

- Fragment file

I used the code providing in the Project 3 subject for the Bump-Mapping with

```
vMC = aVertex.xyz;
```

And for the lighting, instead of defining the normal as in the code from class:

```
Normal = normalize(vNf);
```

I used the function RotateNormal:

```
Normal = RotateNormal(angx, angy, vNf);
```

And finally, I implemented the lighting ambient, diffuse and specular properties of the lighting as seen in class using uKa, uKd, uShininess, uSpecularColor and uColor.

Therefore, by defining aVertex according to z, the surface's shape will follow the cosine equation on Z as well as the lighting thanks to the normal/tangents that use dzdx and dzdy.

2 Side-by-side images showing different values for the input parameters

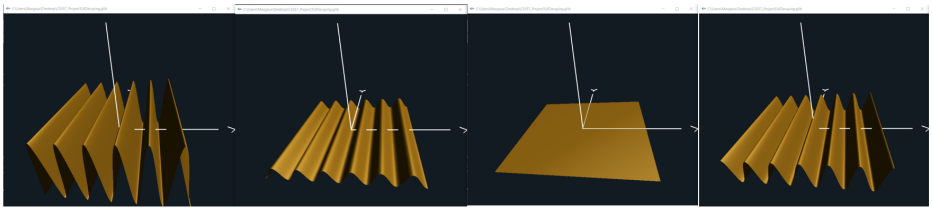

Figure 1: Parameter uA: -0.5, -0.1, 0, 0.2

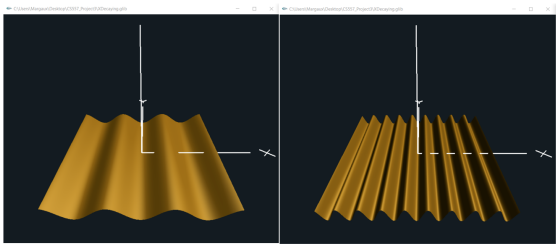

Figure 2: Parameter uB: 1.46 and 4.4

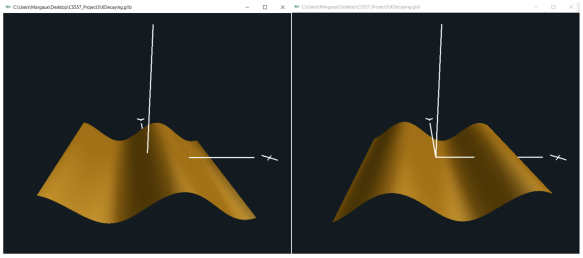

Figure 3: Parameter uC: 2 and 5.36

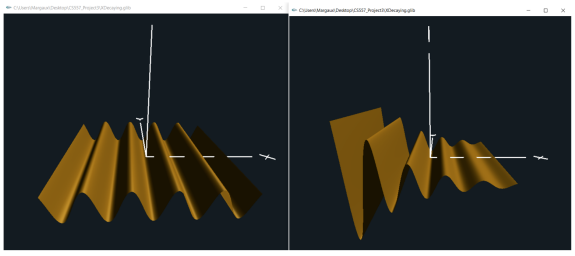

Figure 4: Parameter uD: 0 and 1.76

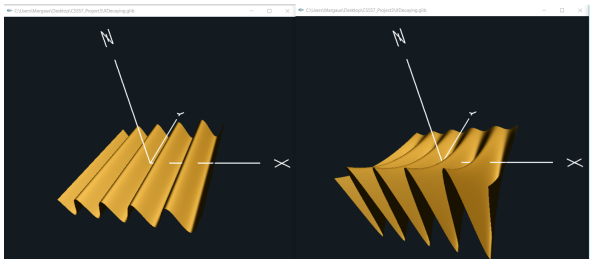

Figure 5: Parameter uE: 0 and 1.4

3 Per-fragment lighted image(s)

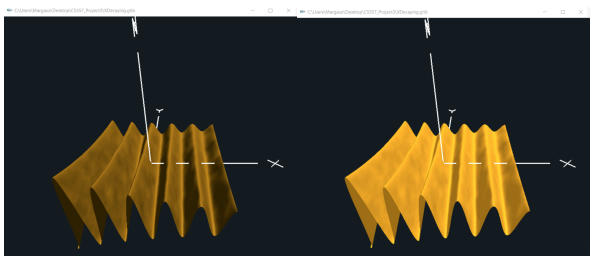

Figure 6: Parameter uKa:0.18 and 0.62

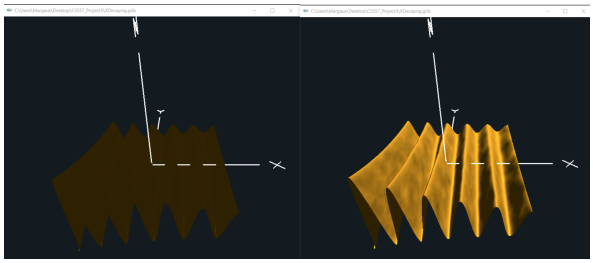

Figure 7: Parameter uKd:0 and 1

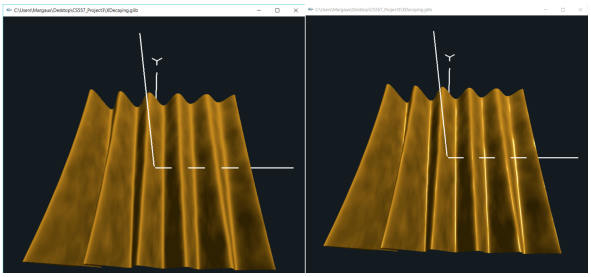

Figure 8: Parameter uKs:0 and 1

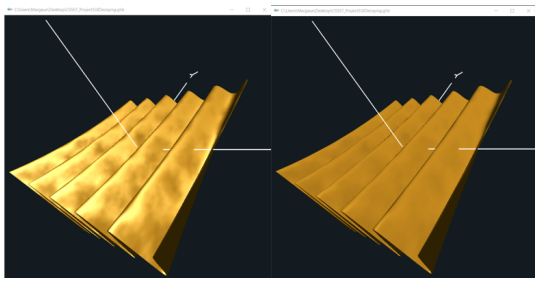

Figure 9: Parameter ushininess:3 and 40

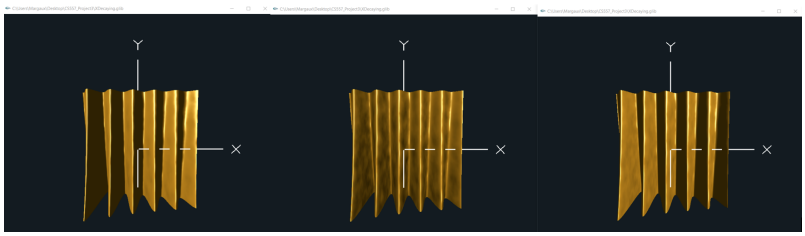

Figure 10: Parameter ulightX:-20, 0 and 20

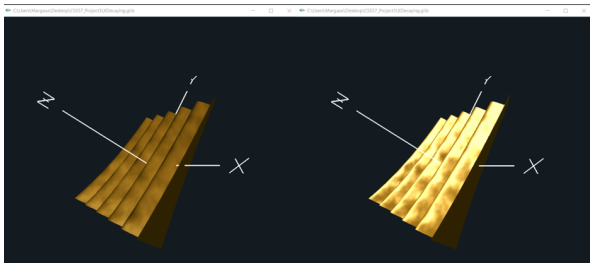

Figure 11: Parameter ulightY:-20 and 20

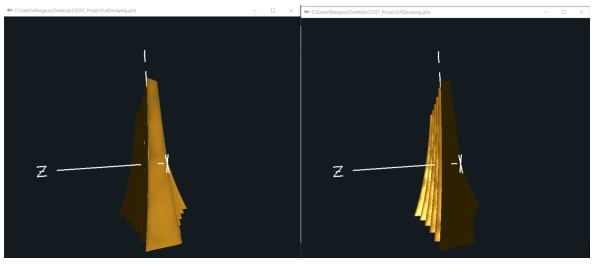

Figure 12: Parameter ulightZ:-20 and 20

4 Extra Credits

The idea now is to decay in a radius around a center point instead of decaying in X and Y separately. The shape wanted looks like the ripples you get from throwing a rock in a pond. So now we use this equation toward z:

```
z = uA * cos(2*PI*uB*r+uC)*exp(-uD*r)
```

with:

```
float r = sqrt(x*x+y*y);
```

So now the calculus derivatives are:

```
float dzdx = uA*2*PI*uB*drdx*(-sin(2*PI*uB*r+uC))*exp(-uD*r)

+uA*cos(2*PI*uB*r+uC)*(-uD*drdx)*exp(-uD*r);

float dzdy = uA*2*PI*uB*drdy*(-sin(2*PI*uB*r+uC))*exp(-uD*r)

+uA*cos(2*PI*uB*r+uC)*(-uD*drdy)*exp(-uD*r);
```

with

```
float drdx = x / r;
float drdy = y / r;
```

By using this z, dzdx and dzdy in the previous code, we obtain the result desired both for the shape and lighting:

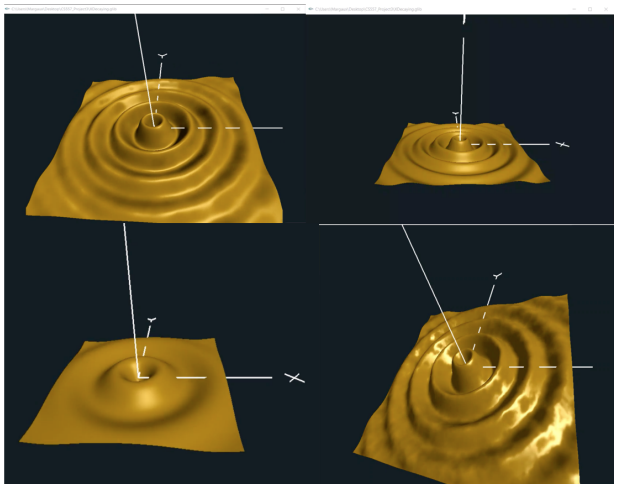

Figure 13: Some screenshots of the new surface

5 Link to the video

https://media.oregonstate.edu/media/t/0$_e$$ojb7l73$