Cube Mapping Reflective and Refractive Bump-mapped Surfaces

Margaux Masson

Winter 2018 - CS557 - Project 4

1 What I did and how it works

In this project we want to use cube-mapping to create a reflective and refractive display of a bump-mapped math function like the one we implemented in the Project 3 (decaying cosine wave or Polar Ripples).

- Vertex file

First, I re-used the decaying cosine wave equation:

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

with the corresponding calculus derivatives, the tangent vectors and normal all calculated in the vertex shader. Then, we want to pass the normla vector following to the fragment shader in which the Bump-Mapping is implemented:

```
vec3 newNormal=normalize(gl_NormalMatrix*cross( Tx, Ty));
vNs = newNormal;
```

- Fragment file

The fragment file is where the Bump-Mapping is implemented, exactly with the same code as in Project 3

```
vec4 nvx = texture( Noise3, uNoiseFreq*vMC );
float angx = nvx.r + nvx.g + nvx.b + nvx.a - 2.;
angx *= uNoiseAmp;
vec4 nvy = texture( Noise3,uNoiseFreq*
vec3(vMC.xy,vMC.z+0.5) );
float angy = nvy.r + nvy.g + nvy.b + nvy.a - 2.;
angy *= uNoiseAmp;
```

with, passed by the vertex shader:

```
vMC = gl_Vertex.xyz;
```

Then, we get the normal vector passed by the vertex shader as well and apply the Bump-Mapping:

```
vec3 Normal = normalize(vNs);
Normal = normalize( RotateNormal( angx, angy, Normal ));
```

Once the final Normal vector is obtained, the reflection and refraction can be done as following:

```
vec3 reflectVector = reflect( vEC, Normal );
vec3 reflectcolor = textureCube(uReflectUnit,
reflectVector ).rgb;
vec3 refractVector = refract( vEC, Normal, uEta );
//uEta = refraction's factor
vec3 refractcolor = textureCube( uRefractUnit,
refractVector).rgb;

refractcolor = mix( refractcolor, WHITE, 0.30 );
gl_FragColor = vec4( mix( reflectcolor, refractcolor, uMix )
    , 1. );
```

with uRefractUnit and uReflectUnit defined in the .glib file. The mix function used in the last line allows the user to blend between the reflaction and the refraction.

Figure 1: Reflection

Figure 2: Refraction with different uEta inputs

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

1. Reflection

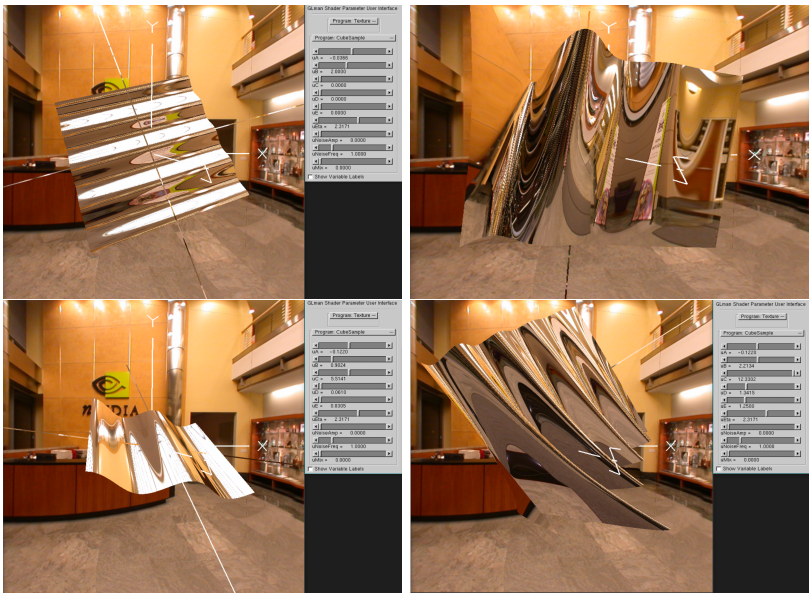

Figure 3: Reflection - Parameter with different values for the input parameters

2. Refraction

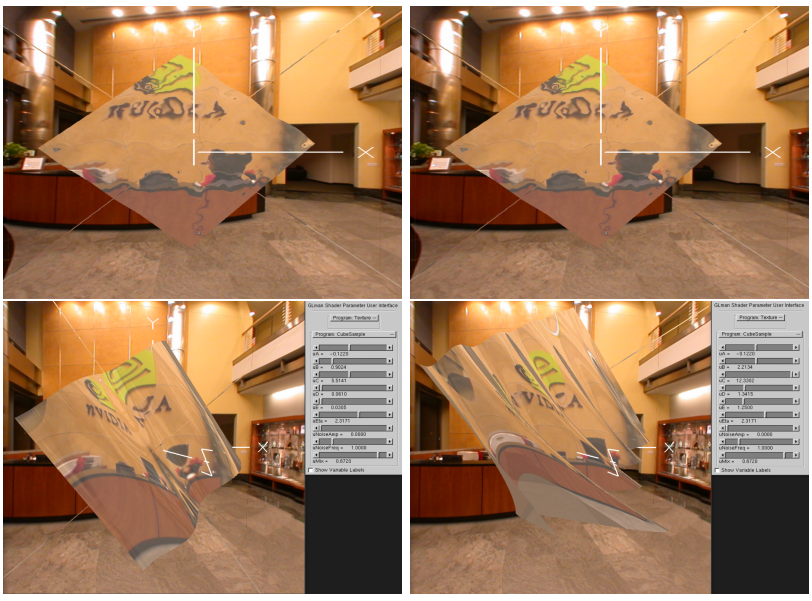

Figure 4: Refraction - Parameter with different values for the input parameters

3 Image(s) showing that your bump-mapping is correct

Figure 5: Bump-Mapping - Noise for the Reflection and Refraction effects

4 Mix the reflective and refractive outputs

Figure 6: Mix Refraction and Reflection

5 Link to the video

https://media.oregonstate.edu/media/t/0_ct4xz2l7

6 Some extra pictures using the Polar Ripples

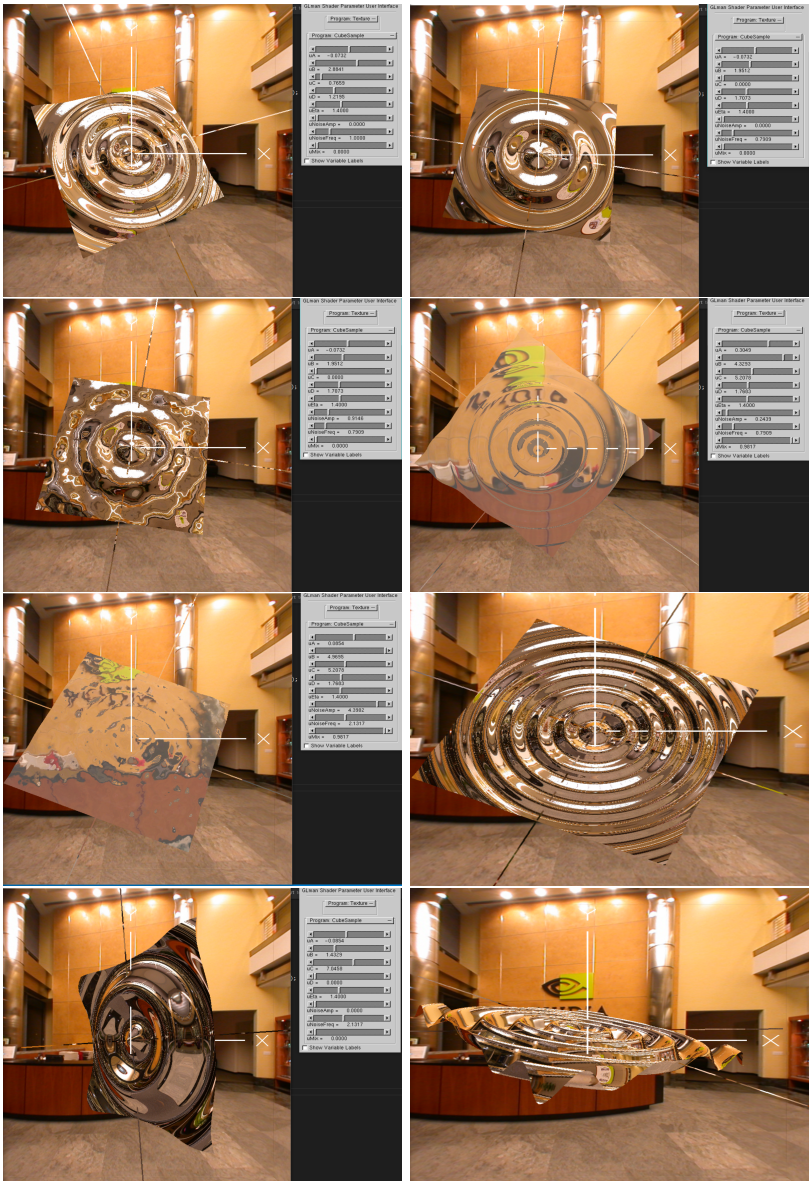

Figure 7: Polar ripples