Project #4: Displacement Mapping, Lighting, and Bump Mapping

Haoxiang Wang; Student ID: 932359049

February 17, 2016

This project is the fourth project I have done in this class, and it is the second project that working with GLSL. Since we gained the experience of *GLman* and *GLSL* from last project, this project seems easy to deal with, and the result turned out that the most part of the project was as easy as I expected except one problem I spent days to fix. The main part of the project takes me around half a day to get everything done and fit to all the requirements, and the problem takes me two days to fix it. The source listing, the results images and the explanation of how code works will be described in the after section.

1 Source Listing

In this project, three files are necessary. They are .glib file, .vert file, and .frag file. No .obj file is needed in this project. The specific filenames and usages are listed below.

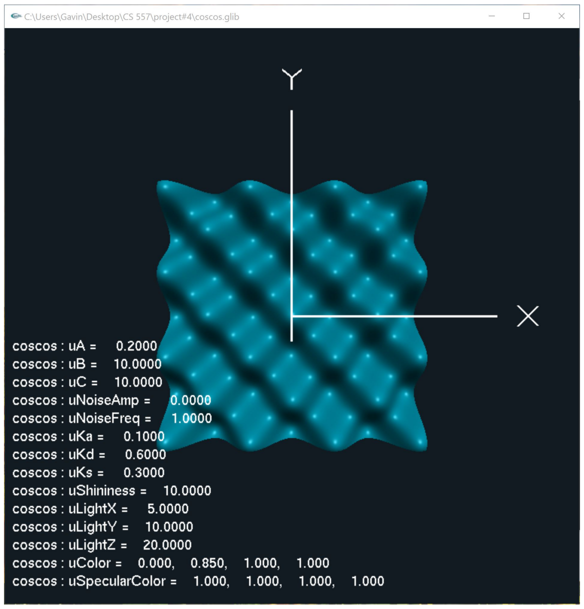
coscos.glib — Handle the user interface and object

coscos.vert — Handle the vertex shader

coscos.frag — handle the fragment shader

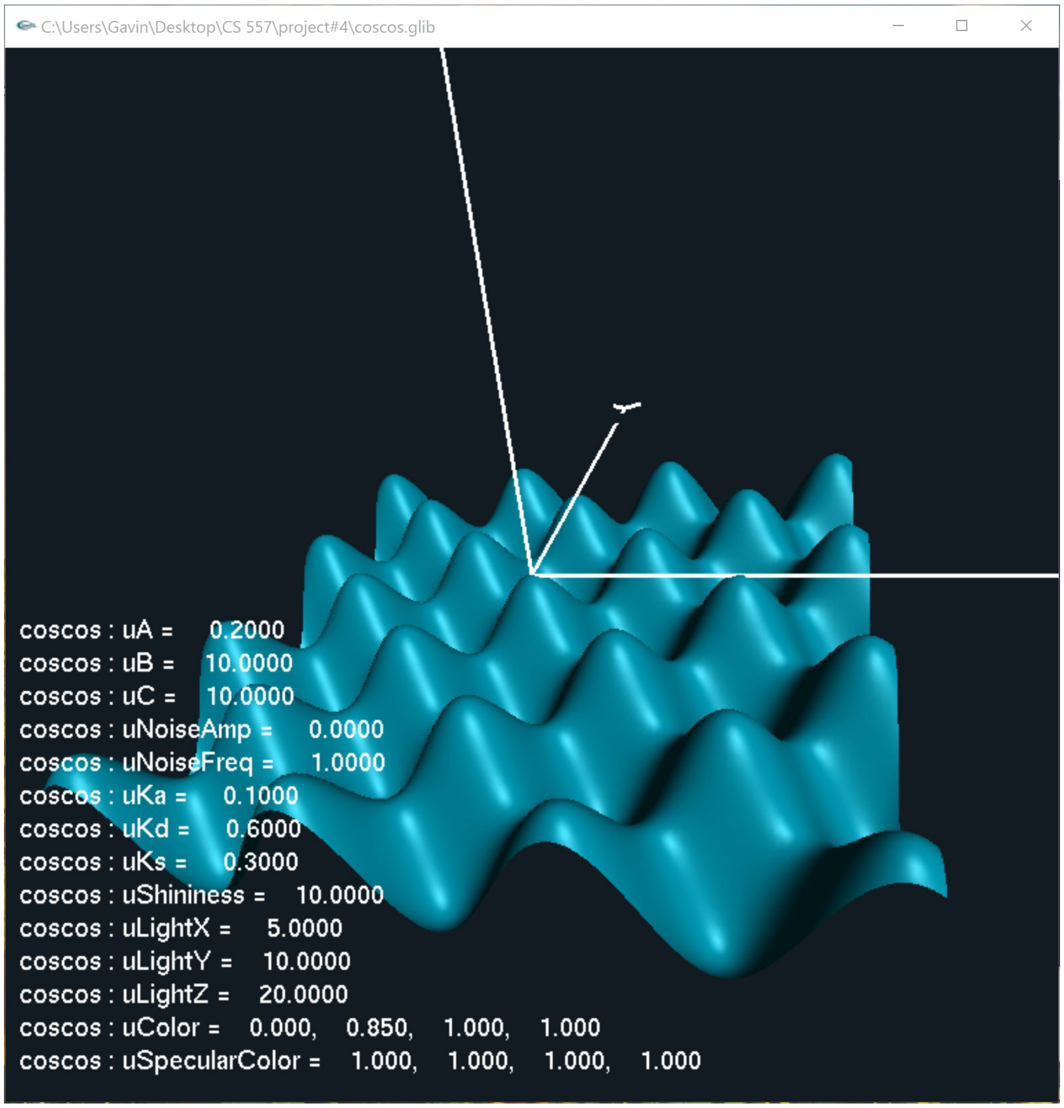
2 Result Images and Explanation

The requirements of this project can be simply divided into two parts. The first one is to put the real displacement on a flat plane in order to create a $cos * cos$ pattern. The second part is to implement the bump mapping with noise on the displaced surface. We also need to implement the fragment lighting to deliver the sense that our displacement and the bump mapping is correct. We have already done something with displacement and bump mapping with *RenderMan* in project 2, when implementing them again with GLSL, the situation differs, the biggest difference will be introduced in the following section. The following image shows that original image and the variable values when the image is rendered.



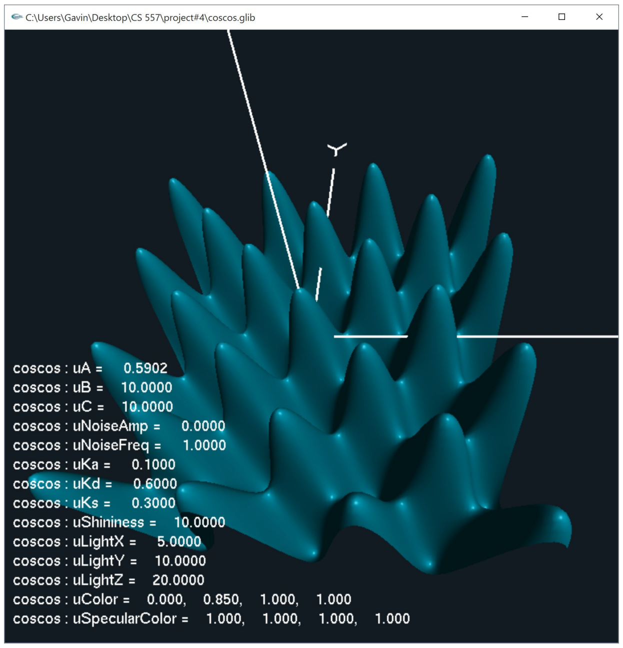
```
coscos : uA =     0.2000
coscos : uB =    10.0000
coscos : uC =    10.0000
coscos : uNoiseAmp =     0.0000
coscos : uNoiseFreq =     1.0000
coscos : uKa =     0.1000
coscos : uKd =     0.6000
coscos : uKs =     0.3000
coscos : uShininess =    10.0000
coscos : uLightX =     5.0000
coscos : uLightY =    10.0000
coscos : uLightZ =    20.0000
coscos : uColor =    0.000,    0.850,    1.000,    1.000
coscos : uSpecularColor =    1.000,    1.000,    1.000,    1.000
```

The original image shows the eye looking at the plane through a vertical direction, which makes the lighting looks not really straightforward. By rotating the whole plane with an angle, the correctness of fragment lighting could be examined. The biggest difference between this project and the project 2 is that GLSL doesn't have the normal calculate function like RenderMan does. In order to implement the correct lighting, we need to calculate the normal manually in the code. The way is to take two tangent value from x and y direction, and them apply a cross product on them. By doing this with every fragment, the correct normal will be produced. The following image shows the fragment lighting I have in in the project.

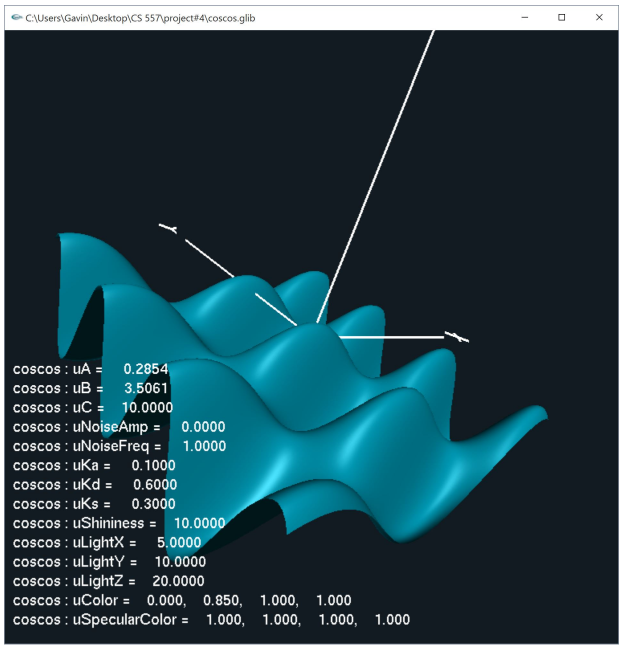


```
coscos : uA =     0.2000
coscos : uB =    10.0000
coscos : uC =    10.0000
coscos : uNoiseAmp =     0.0000
coscos : uNoiseFreq =     1.0000
coscos : uKa =     0.1000
coscos : uKd =     0.6000
coscos : uKs =     0.3000
coscos : uShininess =    10.0000
coscos : uLightX =     5.0000
coscos : uLightY =    10.0000
coscos : uLightZ =    20.0000
coscos : uColor =    0.000,    0.850,    1.000,    1.000
coscos : uSpecularColor =    1.000,    1.000,    1.000,    1.000
```

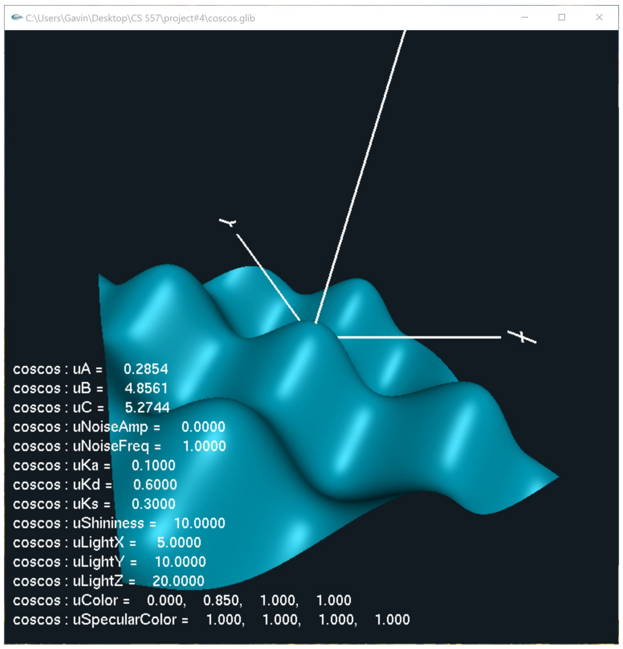
The project has tree sliders named "uA, uB, uC". These three sliders control the displacement of the plane. The uA controls the "height" fo the displacement. The "uB" and "uC" controls the number of the "waves" in the x and y direction. The "height" of the "waves" is actually calculated by the equation $Z = uA * cos(uB * x) * cos(uC * y)$, and this is exactly how these three sliders affect the image. The following images show the changes these three sliders make to the plane. The first is changing the "uA" slider. The second is changing the "uB" slider and the third is changing the "uC" slider.



```
coscos : uA =     0.5902
coscos : uB =    10.0000
coscos : uC =    10.0000
coscos : uNoiseAmp =     0.0000
coscos : uNoiseFreq =     1.0000
coscos : uKa =     0.1000
coscos : uKd =     0.6000
coscos : uKs =     0.3000
coscos : uShininess =    10.0000
coscos : uLightX =     5.0000
coscos : uLightY =    10.0000
coscos : uLightZ =    20.0000
coscos : uColor =    0.000,    0.850,    1.000,    1.000
coscos : uSpecularColor =    1.000,    1.000,    1.000,    1.000
```

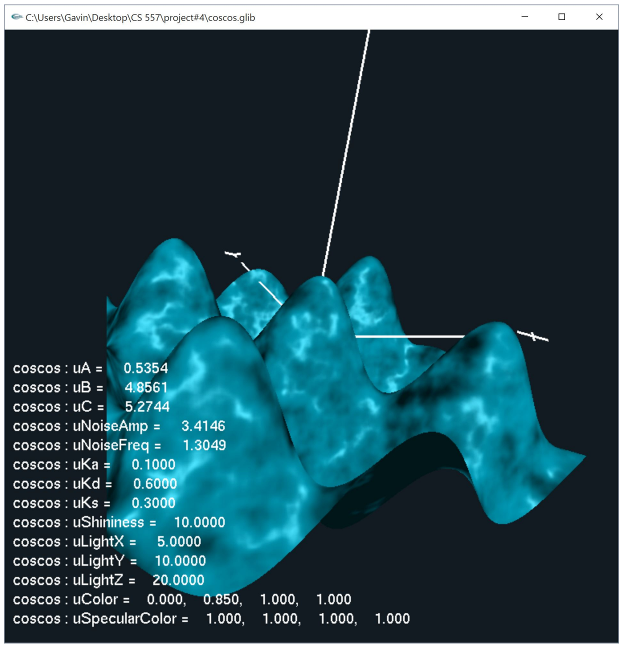


```
coscos : uA =     0.2854
coscos : uB =     3.5061
coscos : uC =    10.0000
coscos : uNoiseAmp =     0.0000
coscos : uNoiseFreq =     1.0000
coscos : uKa =     0.1000
coscos : uKd =     0.6000
coscos : uKs =     0.3000
coscos : uShininess =    10.0000
coscos : uLightX =     5.0000
coscos : uLightY =    10.0000
coscos : uLightZ =    20.0000
coscos : uColor =    0.000,    0.850,    1.000,    1.000
coscos : uSpecularColor =    1.000,    1.000,    1.000,    1.000
```

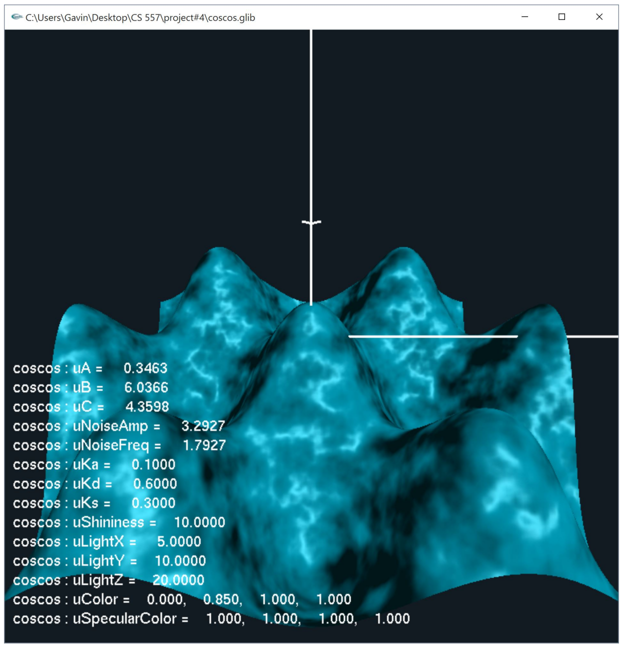


```
coscos : uA =     0.2854
coscos : uB =     4.8561
coscos : uC =     5.2744
coscos : uNoiseAmp =     0.0000
coscos : uNoiseFreq =     1.0000
coscos : uKa =     0.1000
coscos : uKd =     0.6000
coscos : uKs =     0.3000
coscos : uShininess =    10.0000
coscos : uLightX =     5.0000
coscos : uLightY =    10.0000
coscos : uLightZ =    20.0000
coscos : uColor =    0.000,    0.850,    1.000,    1.000
coscos : uSpecularColor =    1.000,    1.000,    1.000,    1.000
```

The second part of the project asks us to add noise to the surface using bump mapping. Like the previous project we did, the bump mapping is a fake displacement. It actually changes the normal instead of implement the real displacement. In this project, I did the same thing to the surface. The project has two sliders named "uNoiseAmp, uNoiseFreq", which stands for noise amplitude and frequency as previous project. Then the I used these two values to calculate two random angles on x and y direction. By applying these two angles to the fragment's normal, the normal will be rotated to another direction. By using this normal to calculate the lighing, it will result in some "dark marks" and "shiny spots", which deliver the sense of an irregularity surface but the truth is the surface is still smooth. The following two images show the result of bump mapping from different looking directions. With the fragment lighting, the correctness of bump mapping could be examined.

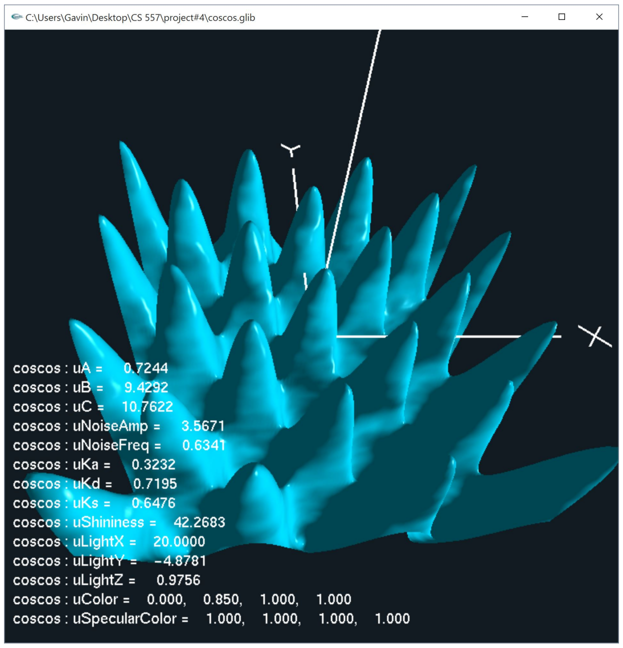


```
C:\Users\Gavin\Desktop\CS 557\project#4\coscos.glib
coscos : uA =     0.5354
coscos : uB =     4.8561
coscos : uC =     5.2744
coscos : uNoiseAmp =     3.4146
coscos : uNoiseFreq =     1.3049
coscos : uKa =     0.1000
coscos : uKd =     0.6000
coscos : uKs =     0.3000
coscos : uShininess =     10.0000
coscos : uLightX =     5.0000
coscos : uLightY =     10.0000
coscos : uLightZ =     20.0000
coscos : uColor =     0.000,     0.850,     1.000,     1.000
coscos : uSpecularColor =     1.000,     1.000,     1.000,     1.000
```

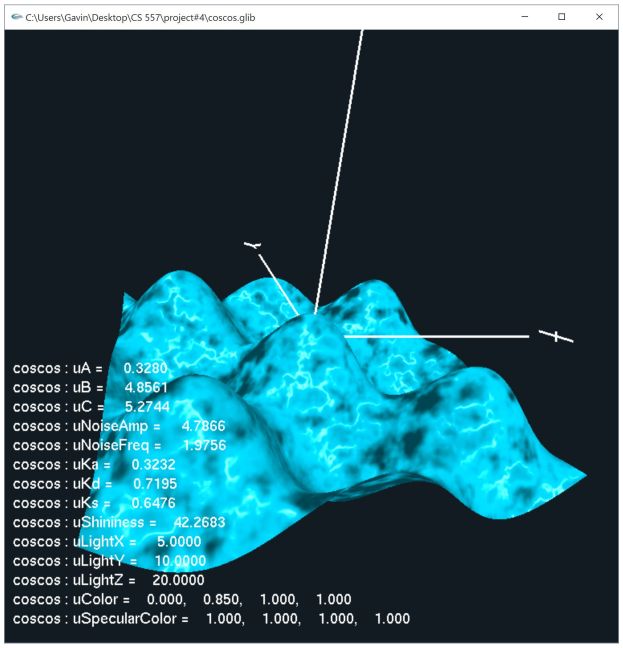


```
C:\Users\Gavin\Desktop\CS 557\project#4\coscos.glib
coscos : uA =     0.3463
coscos : uB =     6.0366
coscos : uC =     4.3598
coscos : uNoiseAmp =     3.2927
coscos : uNoiseFreq =     1.7927
coscos : uKa =     0.1000
coscos : uKd =     0.6000
coscos : uKs =     0.3000
coscos : uShininess =     10.0000
coscos : uLightX =     5.0000
coscos : uLightY =     10.0000
coscos : uLightZ =     20.0000
coscos : uColor =     0.000,     0.850,     1.000,     1.000
coscos : uSpecularColor =     1.000,     1.000,     1.000,     1.000
```

The project also has three sliders named "uLightX, uLightY, uLightZ". They controls the position of the light. By changing these three sliders, the light source position will be changed. The following image on the left shows the result that the light source position has been changed. The project also has three sliders named "uKa, uKd, uKs". These three sliders control the ambient color, diffuse color, and specular color of the light. There also exists one slider named "uShininess", which controls the shiny spot on the surface. By changing the four sliders mentioned above, the sense of different materials of the surface will be dilivered. The following image on the right shows the effect that changing these four sliders.



```
C:\Users\Gavin\Desktop\CS 557\project#4\coscos.glib
coscos : uA =     0.7244
coscos : uB =     9.4292
coscos : uC =     10.7622
coscos : uNoiseAmp =     3.5671
coscos : uNoiseFreq =     0.6341
coscos : uKa =     0.3232
coscos : uKd =     0.7195
coscos : uKs =     0.6476
coscos : uShininess =     42.2683
coscos : uLightX =     20.0000
coscos : uLightY =     −4.8781
coscos : uLightZ =     0.9756
coscos : uColor =     0.000,     0.850,     1.000,     1.000
coscos : uSpecularColor =     1.000,     1.000,     1.000,     1.000
```



```
C:\Users\Gavin\Desktop\CS 557\project#4\coscos.glib
coscos : uA =     0.3280
coscos : uB =     4.8561
coscos : uC =     5.2744
coscos : uNoiseAmp =     4.7866
coscos : uNoiseFreq =     1.9756
coscos : uKa =     0.3232
coscos : uKd =     0.7195
coscos : uKs =     0.6476
coscos : uShininess =     42.2683
coscos : uLightX =     5.0000
coscos : uLightY =     10.0000
coscos : uLightZ =     20.0000
coscos : uColor =     0.000,     0.850,     1.000,     1.000
coscos : uSpecularColor =     1.000,     1.000,     1.000,     1.000
```

3 Summary

This project itself is not hard at all. However, I did get caught by one problem. I timed the noise amplitude to the moise before mapped it to range of $[-1, 1]$, which will result in situation that the whole rotate angle range shifted away. The result image is like the whole surface normal turned around when changing the uNoiseAmp value. Thanks to Prof. Bailey helping me solve the problem so that I could hand in a completed project. Dealing with the problems like this is also fun to me, I would like to do more things about shader and I'm looking forward to the following shader projects and willing to do a lot more on them.