Project #3: Interactive Noisy Elliptical Polka-dots

Haoxiang Wang; Student ID: 932359049

January 31, 2016

This project is the third project I have done in this class, and it is the first project that working with GLSL. The effects we are required to implement are pretty like what we did in the last project with RenderMan. Besides the effects have been implemented, the tolerance and two extra effects are also added into the project. This project takes me around half a day to get everything done and fit to all the requirements. 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. Another .obj file is added into the project in order to create a fun object. The specific filenames and usages are listed below.

ovalnoise.glib — Handle the user interface and object

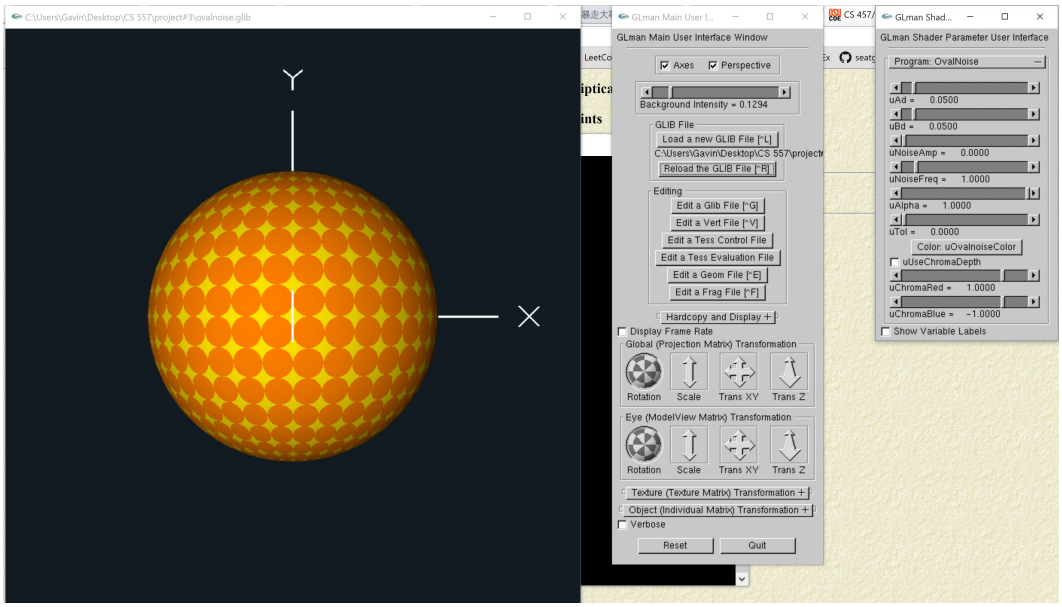
ovalnoise.vert — Handle the vertex shader

ovalnoise.frag — handle the fragment shader

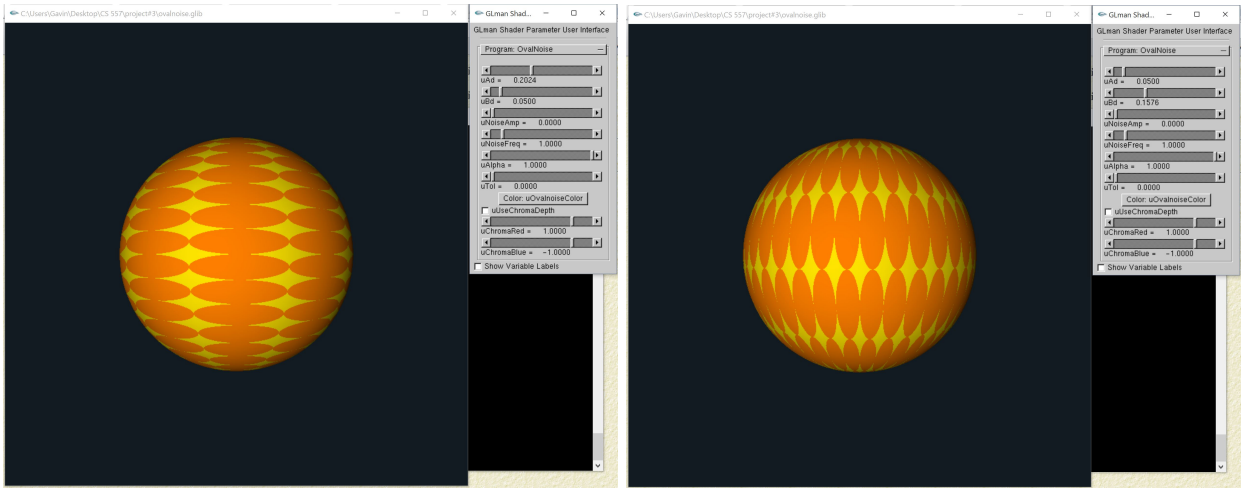
Trex.OBJ — A huge T-rex object

2 Result Images and Explanation

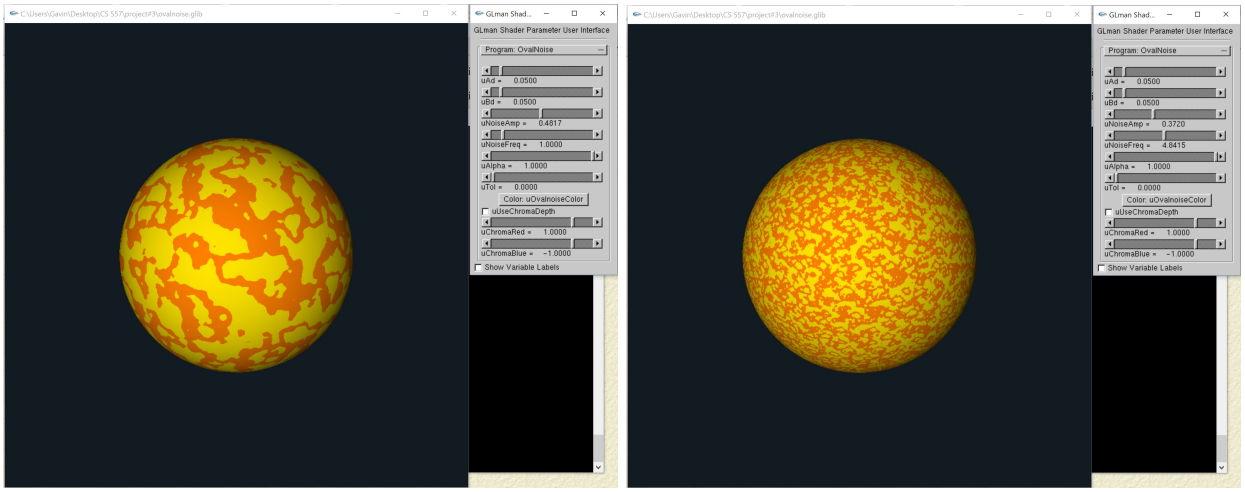
Since this is the first project working with GLSL, and this project is doing the similar jod as the last project. So the basic work of this project is to transform the RenderMan code into the GLSL, and add several new functions into the code. The fillowing image shows the original result that has all parameters set to defult values. It looks pretty similar to the result we have in the previous two projects.



The *uAd* and *uBd* sliders control the two diameters of the ellipse dots. In the RenderMan, we have to change these two values manually in the files, but thanks to the GLSL, we can control them by using two sliders. The first two sliders control these two parameters. The following two images show the examples of what will happen when manipulating these two sliders.

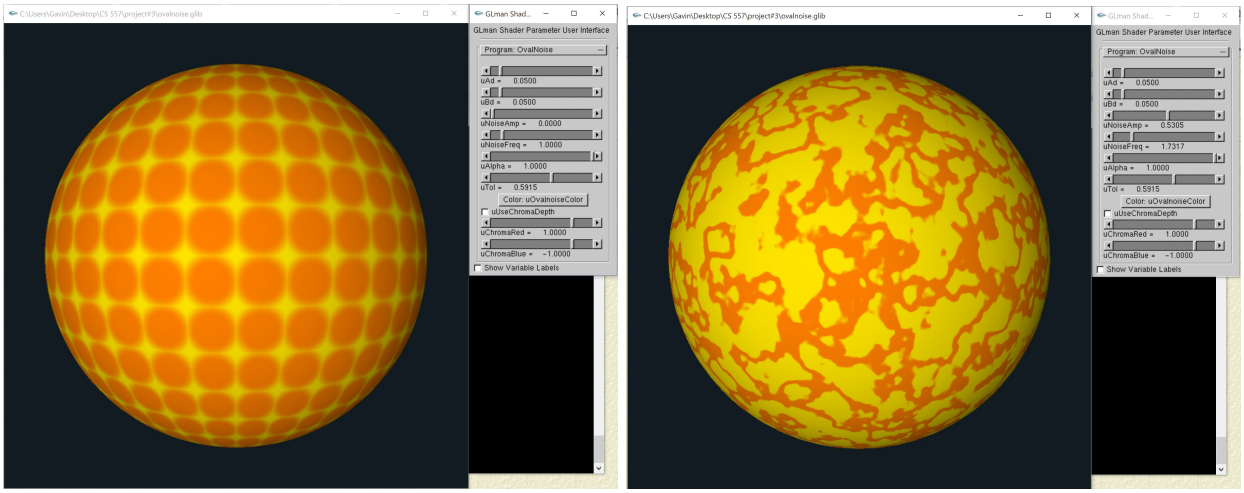


The noise is also applied to this project. The way to implement the noise using GLSL, is also pretty similar to the RenderMan noise. By following the equation $N = NoiseMag * noise(NoiseFreq * PP)$, the noise can be implemented easily. The two coefficients in this equation is controlled by two sliders. The $NoiseMag$ is replaces by $uNoiseAmp$, which is the third slider on the window. The $NoiseFreq$ is controlled by $uNoiseFrep$ which is the forth slider. The $uNoiseAmp$ controls the noise amplitude and the result is shown in the below left image. The $NoiseFreq$ controls the noise frequency and the result is shown in the below right image.



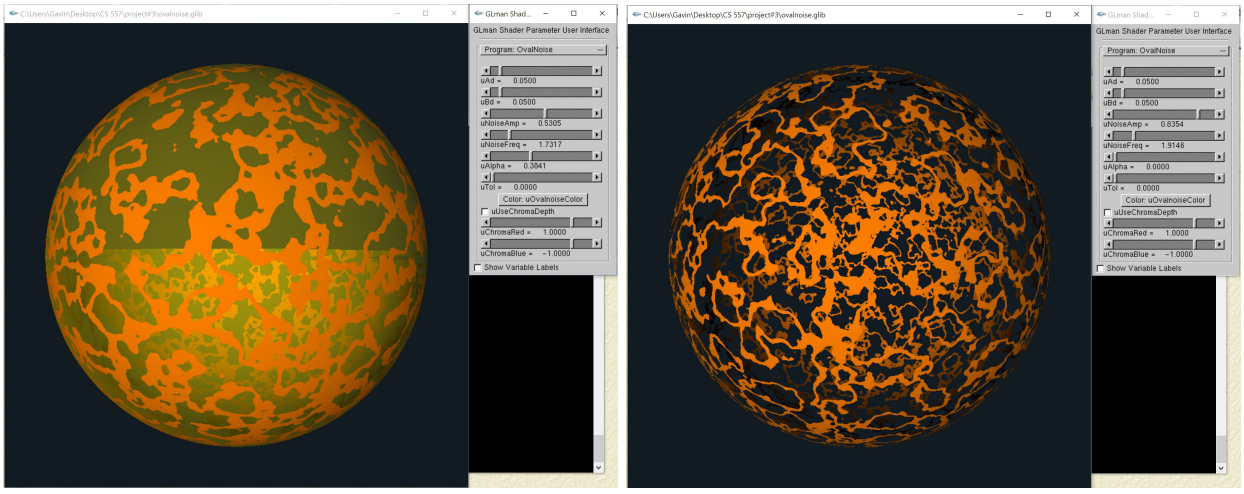
One new feature that has been added into this project is the tolerance. The tolerance has been implemented for both situations that with or without noise. The parameter that controls the tolerance is changed by a slider named "uTol". To implement tolerance, as for this project, I just added and subtracted tolerance value to the boundary value 1, and put these two values into a *smoothstep* function in order to set the range for the colors to get mixed together. Then by putting the range and the two colors together into the *mix* function,

the smooth change in between two colors around the edge will be created. The following two images show the results for adding the tolerance to the pattern on the sphere with and without noise.



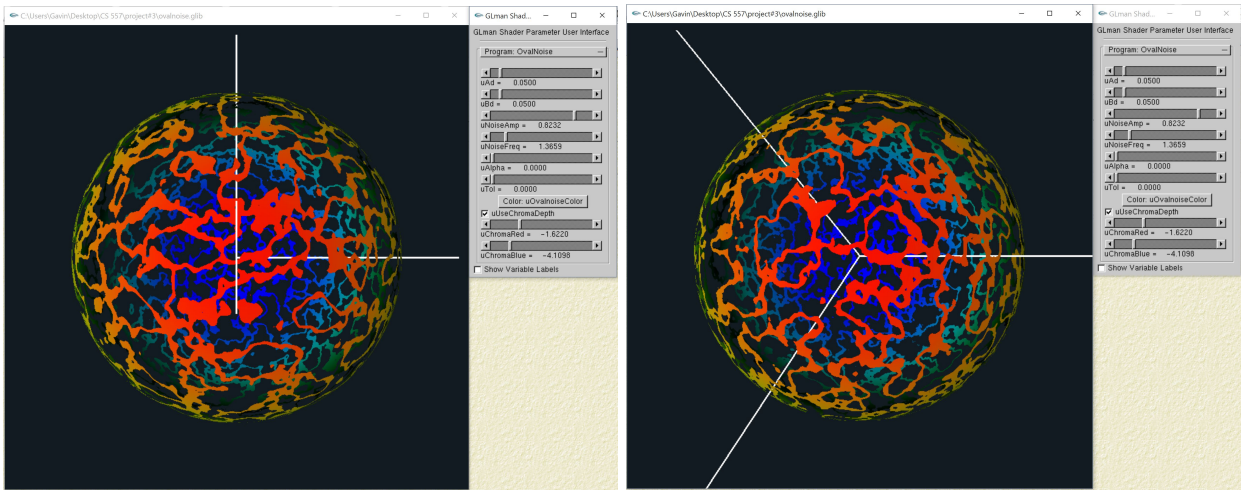
At this point, all of the requirements of the project have been accomplished and explained. The following part of the report will talk about two extra credits of the project.

The first extra credit is implementing the transparency on the sphere surface along with the noise. In the requirement of this exta credit, when the alpha value is between 0 and 1, we can simply add the alpha into account with the RGB, this will result in some part of the sphere cannot be shown properly. When the opacity is set to 0, then we should simply do the "discard" instead of setting the alpha to 0, and this will give us a good looking sphere without any problem. The images below show the result of that. The one on the left is doing with alpha value, and the back upper part of the sphere is black. The one on the right is using discard, and the back upper part turns out to be correct. The opacity is controlled by a slider named *uAlpha* in the interface.

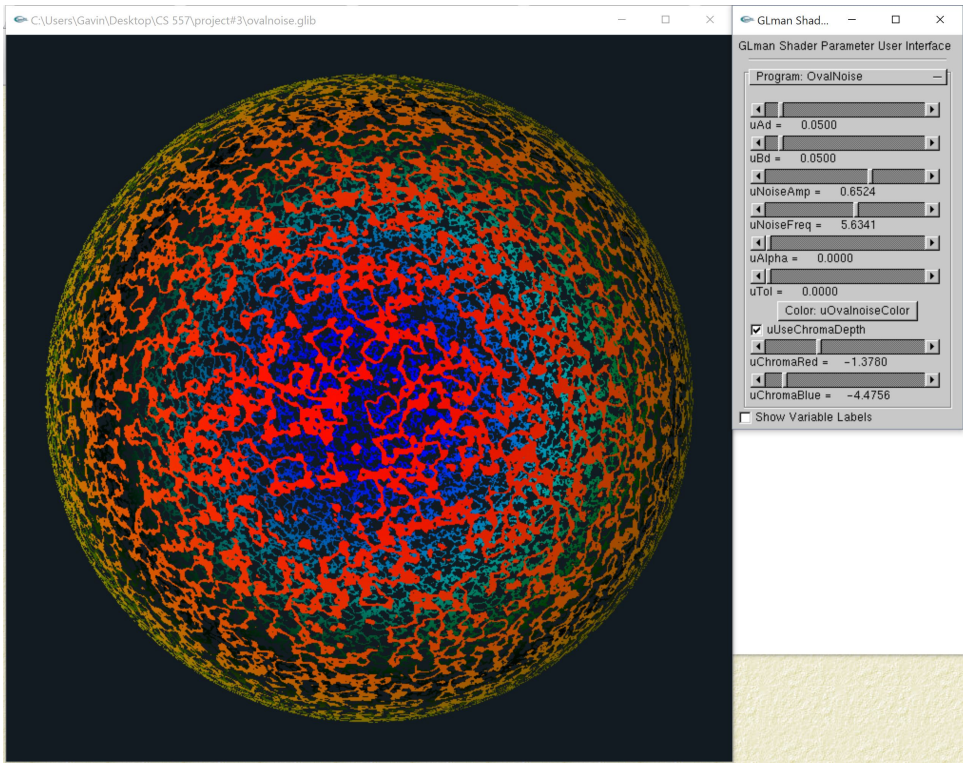


The second extra credit is implementing the "ChromaDepth" feather into the project. The ChromaDepth does the job that rendering the closer part of the object to red and the further part of the object to blue. So that we need to keep the closer part always be rendered as red no matter how the object id rotated of

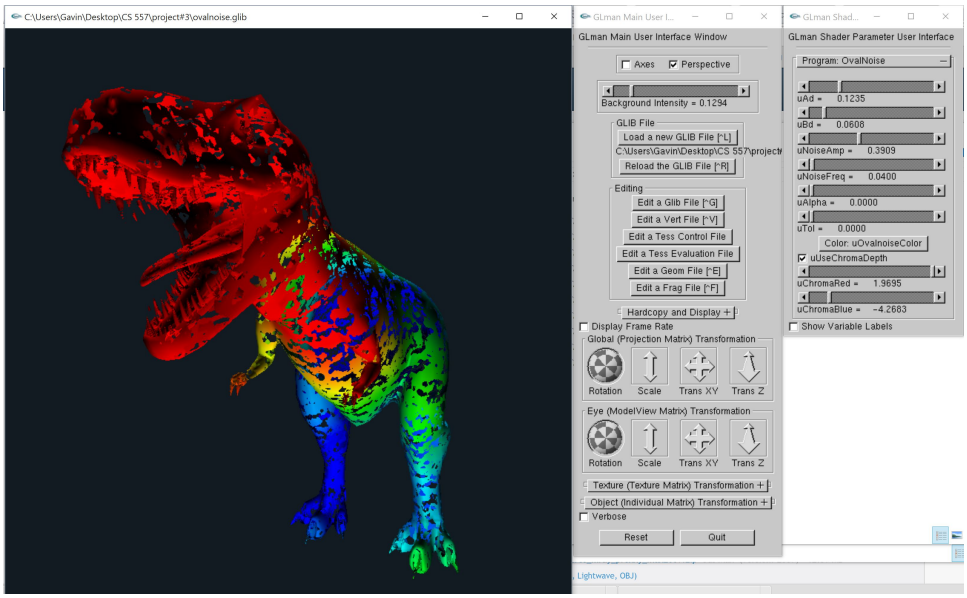
transformed. In order to implement this, we need to change the model coordinate into world coordinate. This is easy to do, just multiply the "gl_ModelViewMatrix" with the model coordinate in the vertex shader will do the trick. The red range and the blue range in the image are controlled by two sliders named *uChromaRed* and *uChromaBlue*. In the following images, the axes is turned on in order to deliver the idea that the sphere has been rotated but the rad-blue range stays the same.



The ChromaDepth could work with a pair of special-made glasses to create a 3D effect. With the glasses on. the red part of the object will become closer to the viewer and the blue part will become further. This effect relates to the face that the glasses refract different colors' wave with different degrees. So looking into the sphere listed below with glasses on will result in a 3D scene.



Finally, by changing the object we put into the scene, we could have something really fun!



3 Summary

This project includes a lot of the knowledge that taught in the class and is really interesting. By doing this project, I got a pretty good review of the knowledge that need to be used in it and gained a better and deeper understanding on them. Playing with noise and some object files is also pretty fun and makes this project a lot more interesting. I'm looking forward to the following shader projects and willing to do a lot more on them.