**COMP 4490 PROJECT**

**KIIBATI ADEOJO**

**7761684**

**PROJECT TITLE: CEL-SHADING, SKYBOX, PERLIN NOISE**

My project would be considering and implementing the following concepts or techniques namely:

* CEL SHADING
* SKYBOX
* PERLIN NOISE

These ideas are inspired by my passion for making video games and how much fun I have learning new interesting and powerful techniques

**CELSHADING**

This is a non-photorealistic shading model that is also known as toon shading. This technique is meant to give a cartoon or comic book like look to a game or object. Compared to photorealistic shading models, cel-shading aims at using less shading color levels.

In cel-shading technique that I would be examining, there is a limit in the number of colors used on an object because the aim is to make the object look flat but at the same time possibly giving a 3D look. A good game that does this well is the Borderlands game. It has a specific art direction that uses cell shading as the main shading technique. This technique totally dictates both the feel of the game and the look of the game.

The color of a cel-shaded object could be determined by a lighting source or rather be the same throughout a scene no matter the lighting applied to it. The advantage of this is that since cel-shading is meant to be non-photorealistic, calculating the right lighting for every pixel or fragment is sometimes not necessary and by doing this computation speed might increase. This is similar to the advantage of using low poly techniques for terrains.

Since a lot of computations can be escaped, one of the ways of implementing cell shading is to clamp all the color values that are within the same range to output the same color.

**CEL SHADING IMPLEMENTATION**

In this project, I would be applying and calculating the ambient, diffuse and specular of a box using a light. After the light has been applied, I would then clamp the color in several ranges(levelS). This would be done in the fragment shader where I would calculate the color of the fragment depending on the lighting and calculate the final color of the fragment depending on the number of levels I would assign in the fragment.

**Fragment Shader Code:**

float brightness =  (fragCol.x + fragCol.y + fragCol.z) / 3;

int lightLevels = 4;  “Dictates amount of difference we want between colours”

float pointLevel = floor(brightness \* levels); “Clamp down value between integer 0 - 4”

brightness = pointLevel / lightLevels;

outColour = fragCol \* brightness; “Returns the final colour”

**SKYBOX**

A skybox is a method of creating a background that to make a scene or level look bigger than it actually is. The method used to create a skybox is called an environment map or in this case, a cube map. A skybox is a (large) cube that encompasses the entire scene and contains 6 images of a surrounding environment. The surrounding environment is the image texture applied to the cube and this image texture could be any image we would like. The images used to make a skybox also influences an object and in a lot of cases, a skybox can be reflected or refracted onto an object.

To visualize the cube, we can use the same image on every side of the box to see the edges of the box.

**SKYBOX IMPLEMENTATION:** In OpenGL, since we are texturing 6 sides of a cube, we have to know which side we are applying which texture too. This is a list of how they are represented: -

* Positive X - Right
* Positive Y - Top
* Positive Z - Back
* Negative X - Left
* Negative Y - Bottom
* Negative Z – Front

This is basically the right coordinate system.

I would be using an external library to load my skybox images - The name is the stb\_image loader. After texturing the box, we can center the box but since we want the skybox to be always in the background of any other object in the scene. To do this, we would have to dsable the depth mask and then enable it after we have drawn the skybox - “glDepthMask(GL\_FALSE);”. After doing all this, we could encounter an issue with the movement because we are using the same view matrix for every object. What we could do to remove the translation from the homogeneous view matrix is that we have to remove the 4th column of the matrix. To do this, we can just change the matrix to a 3x3 matrix and then switch it back to a 4x4 matrix.

For the reflected and reflected box, I would modify the fragment and vertex shader after I have textured the boxes with the same texture I use on the skybox. Also, I have specified normals in the cube box vertice array which I would use for the reflection and refraction.

**Reflected Box:** For this, I would calculate the “view direction vector”, the “objects normal vector” specified in the vertices and then calculate the “reflection vector”.

**In the Vertex Shader:**

Since we have applied non-uniform scaling to our model matrix, our normal vector would be affected. To fix this I would need to get the inverse-transpose of the model matrix and then multiply it by the normal vector to get the new normal vector.

**In the Fragment Shader :**

* The “view direction vector” is simply the camera or view position subtracted from the cube’s position.
* The “reflection vector” is simply using the reflect function in GLSL to reflect the view direction vector and the normal vector.
* After all this, the out variable which is the fragment color is basically the skybox textured with the “reflection vector”

**Refracted or Transparent Box:**

**In the Fragment Shader :**

In theory, the refraction of a surface only deals with the color of the surface. Therefore, I won’t be changing the fragment shader to produce a refracted or transparent surface. Using Snell’s law, I can easily get how I want the box to be refracted. All I need to do is give an “index of refraction (ior)”. I would just pick some low decimal number.

**PERLIN NOISE**

A good random number generator would generate numbers that have no relationship to each other. This is a good thing when a random number is good. However, It would be nice to generate some sort of smooth randomness whereby although, the movement or output is random or in this case, looks random but also has some sort of relation to the last random number. For example, a real-life fire moves randomly but there is some sort of smoothness to the way the fire randomly moves.

It was developed by Ken Perlin in the 1980s and has been used in graphical applications to generate procedural textures, shapes, terrains, and other seemingly organic forms.

In this project, I would be examining making the idea of using the Perlin noise algorithm to make a PPM heightmap image and then show how I can use that PPM image in OpenGL to create a terrain from this heightmap image. To Ken Perlin for the development of Perlin Noise, a technique used to produce natural appearing textures on computer generated surfaces for motion picture visual effects. The development of Perlin Noise has allowed computer graphics artists to better represent the complexity of natural phenomena in visual effects for the motion picture industry.

Using processing, I have implemented a simple Perlin noise program that easily uses the noise() function given by processing. In this case, I am taking a plane with an x, y, z coordinate and then rotating it to make it look like a floor. I am then using noise() to generate a z value depending on the x and y value of that pixel. After all this is done, I draw a TRIANGLE\_STRIP using this x, y, z coordinate.

Finally, Perlin noise is an elegant and amazing algorithm that helps in generating a good amount of randomness.

Using a heightmap, the brightness of a pixel’s color would control the height of that pixel in the terrain. The height of the terrain at each pixel can also be known as the displacement. Therefore, it's a good idea to make the heightmap image black and white where our values are between 0 and 1, therefore, we can specify the height of our terrain depending on this value. In most cases the brighter the pixel’s colour, the higher the height of the displacement in the terrain. But this can be changed to be the other way around.

**Making Heightmap Image Implementation:**

To create this 2d image. I would be making the image into a ppm format because it is a easy image format to create.

* Consider a 2d grid of image.
* Divide this image grid into squares and examine each vertices of every square.
* For every square assign a random gradient vector or I could use a gradient table that is randomly generated. Let’s name the gradients G.
* After this, for every point P which is a pixel inside a square, draw 4 vectors from the pixel to each vertex of the square. Let’s name that S.
* Take the dot product of the G and S of each vertex. **C = G.S**.
* I then take the linear interpolation of the two top vertexes and the two bottom vertexes. Let’s name these linear interpolations as n0 and n1.
* To get the color value of that pixel in the image, we take the linear interpolation of n0 and n1 that gives us our grayscale color value.

**Loading Heightmap Image to make a terrain:**

To load this, as a terrain I would be make a plane and then the brightness of each pixel would depend on this height map.

* First, I make a plane without a z-value which would make it completely flat.
* Then run through each pixel in an image and use this height map colour to pick the z value of that triangle.
* After these vertices are made, draw the triangles with TRIANGLE\_STRIP or LINE\_STRIP.

In conclusion, I enjoyed reading, learning and implementing these new techniques. There are always improvements coming up in computer graphics and in the case of the features, there are several ways to implement or even optimize them. I learned some extra tools like loading and reading ppm images. I chose ppm images because it is one of the most available image formats out there with a lot of resources.

**REFERENCES**

* <https://en.wikipedia.org/wiki/Cel_shading>
* <https://en.wikipedia.org/wiki/Skybox_(video_games)>
* <http://ogldev.atspace.co.uk/www/tutorial25/tutorial25.html>
* <https://en.wikipedia.org/wiki/Perlin_noise>
* <https://medium.com/100-days-of-algorithms/day-88-perlin-noise-96d23158a44c>
* <https://flafla2.github.io/2014/08/09/perlinnoise.html>
* <https://www.youtube.com/watch?v=Or19ilef4wE&t=1s>
* <https://www.youtube.com/watch?v=MJ3bvCkHJtE>

**DRAWINGS TO EXPLAIN PERLIN NOISE**