CS4102 - P2

Graphics Implementation

200007413

April 2024

1 Basic tasks

The shape created for this first part was a diamond, based on the photo in Figure 1. I decided to make the shape thinner in the z-axis, and have only 12 faces (skipping the side center faces).

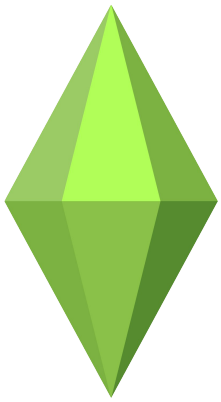

Figure 1: Reference image used to create the diamond geometry.

I wanted each face to be eventually textured, and shaded per-face, so I separated the vertices. As all faces are triangles, requiring 3 vertices, this meant a total of $12 \times 3 = 36$ vertices were required.

Starting from the bottom left face, I defined the vertices required from each face, moving around the bottom anti-clockwise, and then starting again at the top left in the image

moving around anti-clockwise again. This defined the order of faces for the indices, color, and (eventually) normal arrays.

The colors were then defined for each vertex. This was set as a constant green color (not used when textured).

The texturecoords are on a 0 to 1 scale of the square texture image. I set the texture to repeat on the front and back, stretching across to not be too stretched. As the shape is thinner (x ranges from -0.4 to 0.4) , I calculated the aspect ratio, and used the same aspect ratio in the amount of the texture to use. The height is 2 (range -1 to 1), and width 0.8, and so it is 40% as wide as tall. The texcoords therefore range in the x from 0.3 to 0.7 (40% of 1).

The `vertCode` defines the code for the vertex shader. This takes the shape definition as aPosition, and the position changes (translations, rotations, etc), as the uModelMatrix. It also takes a uViewMatrix for the camera position (skewed for oblique projection). By multiplying each of these, the final position of a vertex can be calculated.

This, and the texture coordinates, and a constant ambient light object was sent to the fragment shader, which defines the color of the pixels. This takes the texture stored previously, and multiplies the color at the texcoord with the ambient light vector to get the final color of the pixel.

The fragment shader also has the useTexture boolean. If no texture is used, the color array is multiplied by the ambient lighting to get the final colour.

The texture itself is created and stored. When an object is drawn, the texture is loaded and sent to the fragment shader as uTexture. When applying a texture, this is when the useTexture boolean is set to be true or false depending on if the supplied texture is null.

The oblique perspective was created by using a model view matrix `M_model`. This positioned the object, and scaled it down to fit in the clipping volume. I then created a projection matrix, which skewed the model view matrix by an amount determined by an angle and scale. This model view matrix is then given to the vertex shader as `uViewMatrix`.

To achieve the rotation, a model matrix was created. The model matrix, view matrix. and vertex position definitions are multiplied to get the final position of each vertex.

The model matrix defines the movement of each object separately (as opposed to the view matrix applying global transformations). A directions array defines the direction and speed of the rotations for each object in the scene. These values are set as the X, Y, and Z rotation values for the model matrix, and then sent to the vertex shader.

Using a render function, the rotation direction is multiplied by the now parameter given by renderAnimationFrame, which allows a constant increase in rotation amount. This render function then calls renderAnimationFrame to create an infinite render loop.

2 Standard Tasks

To add directional lighting, normals were required to be calculated to determine how light would reflect. Each vertex requires a normal set to that of the average normal of the faces it is involved in. As the vertices are repeated to ensure they are only used on once face, this means the three vertex normals are just the same normal to the face.

These normals could be calculated by looping through each face, and fetching the three points involved A, B, and C. These could be used to get two vectors representing the edges of the triangle by subtracting A from B, then C from B. Then calculating the cross product of these two vertices gets the vector perpendicular to the face (i.e. the normal). These were normalized to between 1 and -1, and added three times to the normals matrix (faces use three unique vertices). This function can be found in the `getNormals.js` file.

After calculating the normals, I stored these in the `diamond()` function as they are fixed for this shape.

I then added global movement by adjusting the idependent movement code from the first part. Instead of each item having it's own direction, and using their origins as the center of movement, I defined a global rotation origin and axis to define the rotation, and multiplied this value by a speed constant for each object. To apply it to each object, the rotation origin had to be centered by translating the movement matrix M by the inverse of the origin vector, applying the rotation, and then translating by the origin vector to get back to the correct position.

I then had to add directional lighting to the vertex shader code. This was done by adding in lighting color and direction vectors, which would be dot multiplied with the normals to detect the amount the light bounced.

Because the objects are moving (rotating), the normals must be updated. A separate attribute uNormalMatrix was created, which took the overal movement of the object M, and then inverted and transposed it to get the normal's movement. This is multiplied in the vertex shader by the objects original aNormal matrix, to get the adjusted normals.

3 Advanced Tasks

I attempted, but did not complete this task. I attemtped to create a skybox, which would be reflected by an object on the screen. This was a simple cube surrounding the camera, which would have one texture applied to each face separately. This required a different texture type, and I had to create a separate shader program for the skybox, with a separate vertex and fragment shader, ensuring.

To prevent distortions in the skybox image, the perspective transformations from the

camera have to be ignored, so only the rotation affects what is rendered.

However, despite being able to create the object and shader without error, the skybox didn't display. The images have loaded, but I am unsure if the positioning is incorrect.