Week 8 Tutorial Solutions

Question 1: Texturing

a. What is a reason to use texture mapping rather than lots of little polygons? Are the two representations functionally equivalent? What are the differences?

Textures can be used to give the illusion of complex geometry while keeping the actual geometric complexity low. Rather than using textures, we could just add lots of polygons to the figure to model the extra details, but this would slow down drawing, and it would be a lot of work to figure out the extra points and faces that we want to add.

They are not functionally equivalent. Textures can be used to simulate small features that are not actually there in the geometry so will not respond to lighting or in the same way. For example, if we shine light nearly parallel to a given face on the golf ball, one side of the dimple should be light and the other should be dark, but this won't happen if we're using textures. Textures also have resolution and aliasing issues.

b. What is difference between the following 2 texture filter settings?

```
gl.glTexParameteri(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER, GL.GL_NEAREST);

gl.glTexParameteri(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER, GL.GL_LINEAR);
```

These are 2 different filter settings for magnification. If we are in a situation where one texel gets mapped to many pixels (like zooming in or enlarging a photo too much) we have 2 choices of filters.

GL.GL_NEAREST just chooses the texel closest to the texture coordinate value output by the rasteriser. This is known as point sampling and is most subject to visible aliasing.

GL.GL_LINEAR is a filter that for 2D textures performs bilinear interpolation across the 4 nearest texels [2 texels in S direction and 2 Texels in T direction] to create a smoother (sometimes blurrier) image This approach is more expensive to compute.

c. What are Mip Maps? Why are they used?

They are pre-calculated, sequences of textures, each of which is a progressively lower resolution representation of the same image. The height and width of each image, or level, in the mipmap is a power of two smaller than the previous level where each texel in the next level is calculated by averaging 4 of the parent texels.

These can help with minification aliasing problems which can arise when more than one texel is mapped to one pixel (like zooming out). Without mip-mapping you can use similar filters for minification as for magnification - GL_NEAREST and GL_LINEAR. WITH mip mapping you can use GL_NEAREST_MIPMAP_NEAREST which returns the nearest texel on the nearest mip map or GL_LINEAR_MIPMAP_LINEAR which is trilinear filtering where bilinear filtering is used on 2 of the nearest mipmaps and then interpolated. There are also other filter settings.

d. What do the following settings do?

```
float flargest[] = new float[];
gl.glGetFloatv(GL.GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, fLargest[0]);
gl.glTexParameterf(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAX_ANISOTROPY_EXT,fLargest[0]);
```

This code is turning on anisotropic filtering. Like trilinear filtering, anisotropic filtering is used to eliminate aliasing effects, but improves on trilinear filtering by reducing blur and preserving

detail at extreme viewing angles. Anisotropic can take a different number of samples in the horizontal vs vertical directions depending on the projected shape of the texel. Where as isotropic (trilinear/mipmapping) always filters the same in each direction.

Different degrees or ratios of anisotropic filtering can be applied during rendering and current hardware rendering implementations set an upper bound on this ratio. This code finds out the maximum level of filtering for the current implementation and sets the filter to this max level.

Like trilinear filtering, anisotropic filtering is used to eliminate aliasing effects, but improves on trilinear filtering by reducing blur and preserving detail at extreme viewing angles. Anisotropic can take a different number of samles in the horizontal vs vertical directions deeding on the rojected shape of the texel. Where as isotroic (trilinear/mipmapping) always filters the same in each direction.

For best results, combine anisotropic filtering with a GL_LINEAR_MIPMAP_LINEAR minification filter.

Question 2: Texture Co-ordinates

Assuming texture co-ordinates start at 0,0 in the bottom left of the image, define texture co-ordinates for the following situations.

a. A quad with the following vertices, where you want to map the whole texture to the quad.

```
(-1, -1, -1), (1, -1, -1), (1, 1, -1), (-1, 1, -1)
```

(0, 0), (1, 0), (1, 1), (0, 1)

b. The same quad but where you want to map the whole texture to the quad in the y-direction, but have 3 repeated copies in the x direction. Assuming we are using GL2.GL_REPEAT mode.

(0, 0), (3, 0), (3, 1), (0, 1)

c. The triangle

```
(-10, -10, 0), (10, -10, 0), (0, -5, 0)
```

(0, 0), (1, 0), (0.5, 1)

Question 3: Textured meshes

1. Modify the cylinder example from last week with texture coordinates that wrap a texture all the way around the curved surface. The circles at either end should fit as much of a texture on them as they can without distorting or stretching it. When doing this, consider if any vertices that were shared before have to be duplicated.

```
private void makeCylinder(GL3 gl) {
    //An front circle
    List frontcircle = new ArrayList<>();
    List frontIndices = new ArrayList<>();
    List circleTexCoords = new ArrayList<>();
    float angleIncrement = (float) (2*Math.PI/NUM_SLICES);
    for (int i = 0; i < NUM_SLICES; i++) {
        float angle = i * angleIncrement;
        float x = (float) Math.cos(angle);
        float y = (float) Math.sin(angle);
        frontcircle.add(new Point3D(x, y, -1));
        circleTexCoords.add(new Point2D(x/2 + 0.5f, y/2 + 0.5f));

        //Generate indices that effectively create a triangle fan.
        frontIndices.add(i);
```

```
            if (i > 2) {
                frontIndices.add(0);
                frontIndices.add(i-1);
            }
        }


        TriangleMesh front = new TriangleMesh(frontcircle, frontIndices, true, circleTexCoords);

        // The back circle
        List backCircle = new ArrayList<>();
        for (Point3D p : frontcircle)
            backCircle.add(p.translate(0,0,-2));
        List backIndices = new ArrayList<>(frontIndices);
        Collections.reverse(backIndices);

        //Note that the texture on the back circle is mirrored
        TriangleMesh back = new TriangleMesh(backCircle, backIndices, true, circleTexCoords);

        // We want the sides to be smooth, so make sure the vertices are shared.
        List sides = new ArrayList<>();
        List sideIndices = new ArrayList<>();
        List sideTexCoords = new ArrayList<>();
        for (int i = 0; i <= NUM_SLICES; i++) {
            //The corners of the quad we will draw as triangles
            Point3D f = frontcircle.get(i % NUM_SLICES);
            Point3D b = backCircle.get(i % NUM_SLICES);

            sides.add(f);
            sides.add(b);

            float s = 1f*i/NUM_SLICES;

            sideTexCoords.add(new Point2D(s,0));
            sideTexCoords.add(new Point2D(s,1));

            if (i == NUM_SLICES) break;

            //Indices
            int j = i + 1;
            sideIndices.add(2*i);
            sideIndices.add(2*i + 1);
            sideIndices.add(2*j + 1);

            sideIndices.add(2*i);
            sideIndices.add(2*j + 1);
            sideIndices.add(2*j);
        }

        TriangleMesh sidesMesh = new TriangleMesh(sides, sideIndices, true, sideTexCoords);

        front.init(gl);
        back.init(gl);
        sidesMesh.init(gl);
        meshes.add(front);
        meshes.add(back);
        meshes.add(sidesMesh);
    }
```

In this solution, we duplicate the starting pair of vertices in the sides. This is because those vertices have different texture coordinates depending whether you are drawing the first or final quad. Note however that this introduces an unfortunate side effect of screwing up the normal calculations and creating a visible seam under lighting.

2. Modify the code again so that the texture only covers a third of the curved surface. What OpenGL command would stop the texture repeating? **Changing the two lines where the texture coordinates for the sides are set to**

```
sideTexCoords.add(new Point2D(s*3,0));
sideTexCoords.add(new Point2D(s*3,1));
```

in conjunction with setting the texture to CLAMP_TO_BORDER will achieve this effect.

```
gl.glTexParameteri(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL3.GL_CLAMP_TO_BORDER);
gl.glTexParameteri(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL3.GL_CLAMP_TO_BORDER);
```

The border colour can be set with this command:

```
float[] white = {1, 1, 1, 1};
gl.glTexParameterfv(GL.GL_TEXTURE_2D, GL3.GL_TEXTURE_BORDER_COLOR, white, 0);
```

Question 4: Assignment

If there is any time left, discuss final milestone of the assignment.