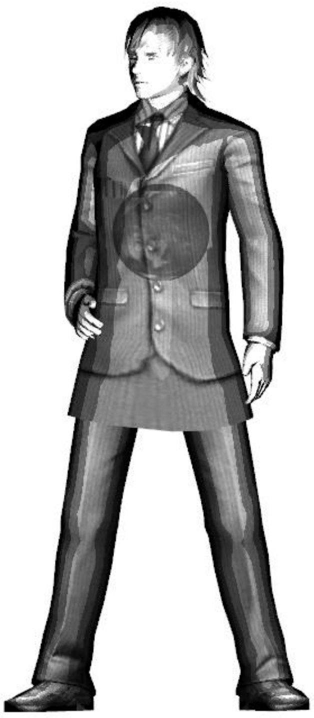
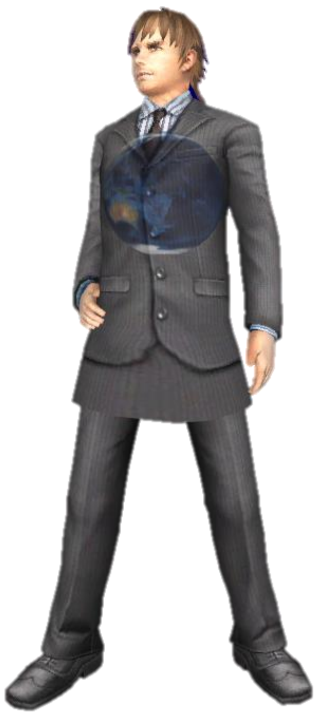
Tommy Chen Cosc342 5277929

Assignment 2: Report





How to use the render:

Arrow keys for movement, press 1 for Comic Shader, press 2 for MTL Shader and press b for enable or disable the opacity.

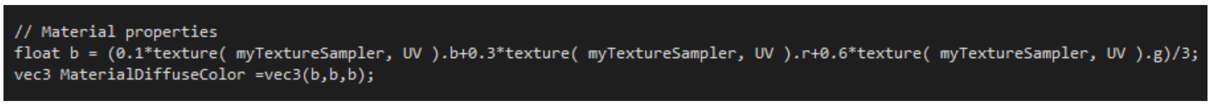
How was the control input implemented:

The float value rendermode is passed from control to scene to object's derivative class group.

Comic Shader:

Comic shader is the combination of Black and white, edge detection, cell shading and Model Flattening.

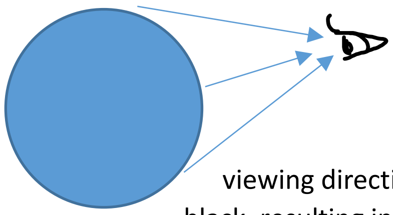
Black and white:



```
// Material properties
float b = (0.1*texture( myTextureSampler, UV ).b+0.3*texture( myTextureSampler, UV ).r+0.6*texture( myTextureSampler, UV ).g)/3;
vec3 MaterialDiffuseColor =vec3(b,b,b);
```

Since this model's texture is mostly blue, `texture( myTextureSampler, UV ).bbb` was used at the start to achieve Black and white by loading the Blue value for all 3 colours of material diffuse. However, at the end, 10% of blue, 30% read and 60% green was used to achieve a truer grayscale image.

Edge detection:



Edge detections takes the angle between the angle between the viewing direction of the camera and the normal. Any angle less than 0.3 is painted black, resulting in the black border around the edges of the model.

Cell shading:

```
int levels = 5;
float scaleFactor = 1.0 / levels;
```

```
vec3 diffuseComponent =  MaterialDiffuseColor * LightPower*LightColor*scaleFactor* pow(floor(cosTheta * levels), 2);
```

The cell shader works by rounding the light intensity down to split up the gradient shift, then scale it down by a factor of 5. the rounding down and the light power is doubled to make the looks more like traditional comic style (more intense);

```
float specMask = (pow(cosTheta, Ns) > .45) ? 1 : 0;
```

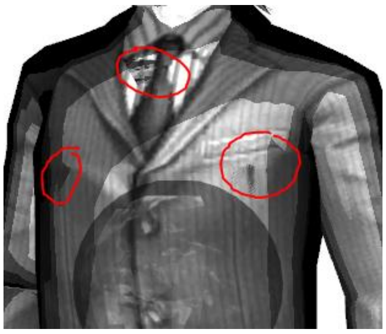
Specular also needs a mask to restrict the shininess, but this model doesn't have any shiny object so this is not needed.

Model Flattening:

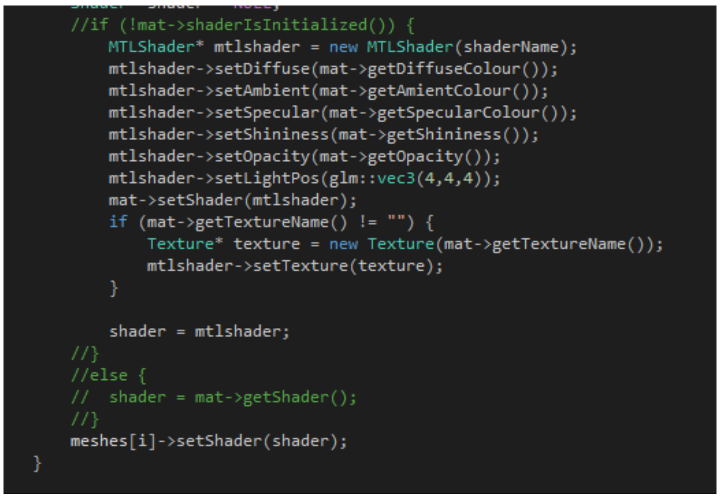
```
// Output position of the vertex, in clip space : MVP * position
vec3 v = vertexPosition_modelspace;
//flatten the z coord of all vertices
v.z = v.z/50;
gl_Position =  MVP * vec4(v,1);
```

To convert a 3D model to 2D, each vertices' z coordinates is set to 0 or closed to 0. In the beginning, the z coord was just set to 0. However, it caused some horrific z-fighting since the render has no idea what vertex is in front of what. To counter that, the z value is divided to get a value close to 0 so the shader can calculate which part is closer to the camera.

Problems encountered:



Even though I have divided the z coord, the model still has some unfixable z fighting.



```
    //if (!mat->shaderIsInitialized()) {
        MTLShader* mtlshader = new MTLShader(shaderName);
        mtlshader->setDiffuse(mat->getDiffuseColour());
        mtlshader->setAmbient(mat->getAmientColour());
        mtlshader->setSpecular(mat->getSpecularColour());
        mtlshader->setShininess(mat->getShininess());
        mtlshader->setOpacity(mat->getOpacity());
        mtlshader->setLightPos(glm::vec3(4,4,4));
        mat->setShader(mtlshader);
        if (mat->getTextureName() != "") {
            Texture* texture = new Texture(mat->getTextureName());
            mtlshader->setTexture(texture);
        }

        shader = mtlshader;
    //}
    //else {
    //  shader = mat->getShader();
    //}
    meshes[i]->setShader(shader);
}
```

To make the shader switch shader on the spot, the if statement that optimise the code by avoiding it from loading the mtl values over and over was taken out. But since this assignment emphasize on graphics not program efficiency a fix is not needed.

Pressing the same shader mode over and over will just continuously load the value, but it's not important.