F20GA: 3D Graphics and Animation Report

Matthew Reilly

30/11/18

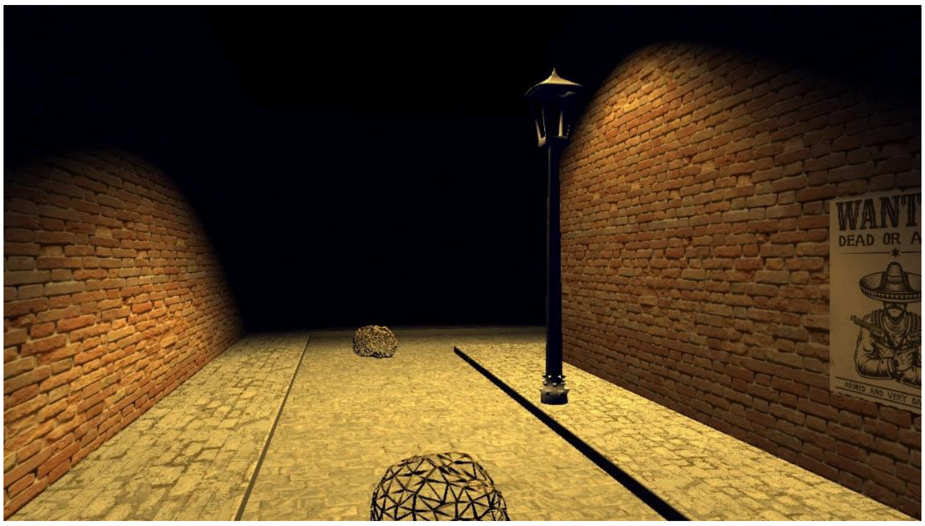


Table of Contents

Modelling and Instancing 3
Lighting and Materials 3
Animation, Interaction and Camera System 4
Appendix’s 5
- Appendix 1 5
- Appendix 2 6
- Appendix 3 6
- Appendix 4 6
- Appendix 5 7
- Appendix 6 7
- Appendix 7 7
- Appendix 8 8

Modelling and Instancing

I created all the object in blender expect from the lamp which I sourced from (https://www.turbosquid.com/FullPreview/Index.cfm/ID/1025013) and the tumbleweed which I sourced from (https://sketchfab.com/models/aba8dfd0195b4f61a761f42a34383e58). I had a prototype (Appendix 1) of the project in blender by I found that the complexity of the lamp and the car where much for the capabilities of the OpenGL implementation, so I change the car to a simple tumbleweed and the lamp to a more basic lamp (Appendix 2).

For the implementation in OpenGL all the relevant information about each object in the scene is stored in a Struct.

The loading of multiple objects works by having a start-up function that is called for each one of the objects(Appendix 3).

After each of the models are loading into the struct a loop will begin and will repeat until the application is closed. Within this loop I call the render function (Appendix 4) for each of the objects which will render each of the objects within the 3d space. For all the objects except of the tumbleweed I do not have to transform them into the correct place as I have already positioned each of the object in the blender scene that I created.

For instancing I created 2 tumbleweeds, which I done by creating a for loop (Appendix 5) that will repeat the translating and rotation and drawing of the object. Each instance of the object has its own model position and model rotation, so it can be positioned in the 3d space independently.

Lighting and Materials

The lighting is applied to each object within the render function (Appendix 6).

I have created a spot light that shines from the top of the lamp down onto the scene. For the light I used lightSpotCutOff and lightSpotOuterCutOff to create a soft edge on the light to make it look more atmospheric.

I used Phong's algorithm to calculate the light within the fragment shader. (Appendix 7)

$$I_p = k_a i_a + \sum_{lights} k_d i_d(\hat{L} \cdot \hat{N}) + k_s i_s(\hat{R} \cdot \hat{V})^{shininess}$$

This works by adding together the ambient light of the world and the diffuse and specular light of each light.

For the materials of my objects I mapped a texture to an object using uv mapping within blender then exported the objects with the materials mapped to it. I sourced all my textures from http://texturelib.com/ .

Animation, Interaction and Camera System

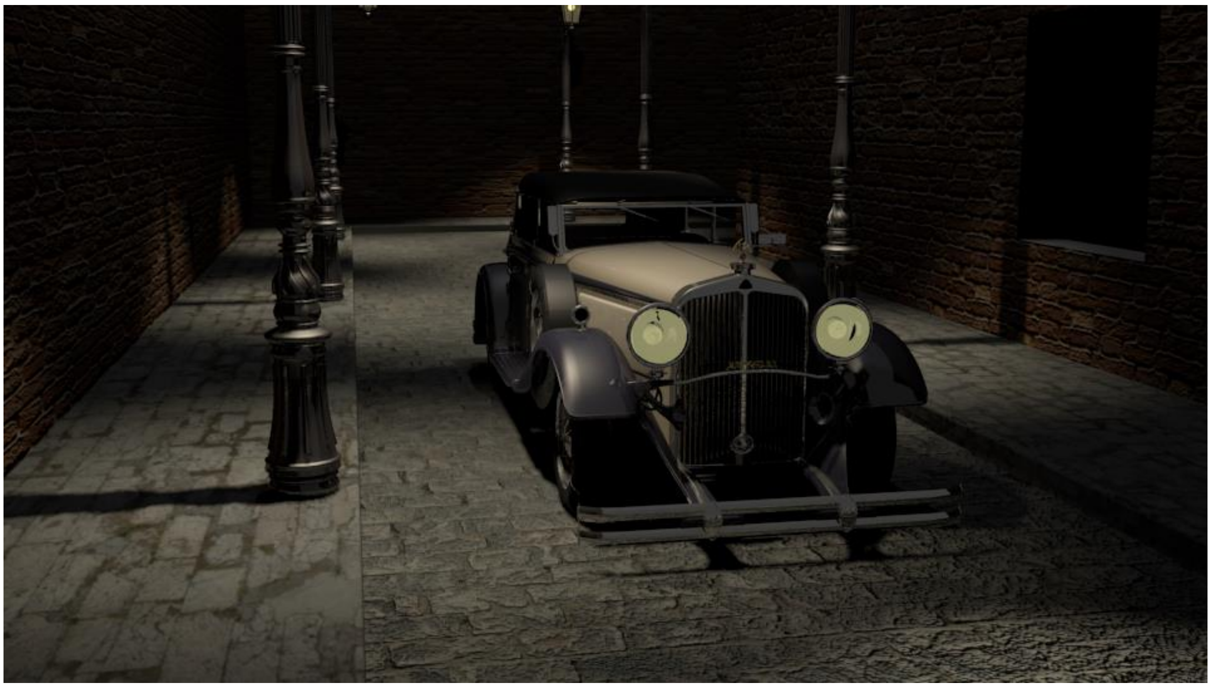
For animation I have applied translation and rotation (Appendix 8) to the instances of the tumbleweed within in render of the tumbleweed object so that they roll along the z axis. When the tumbleweed reaches a certain point it will reset back to its original starting location to repeat going through the scene again.

The interaction of my animation is that the user can rotate the camera, so they can look around the environment, and the user can click the left mouse button in to turn off the light.

The automated camera system works by moving the camera position along the z axis slowly until it hits the end of the street then it will move backwards through the screen until it reaches its original location, it will then repeat this process.

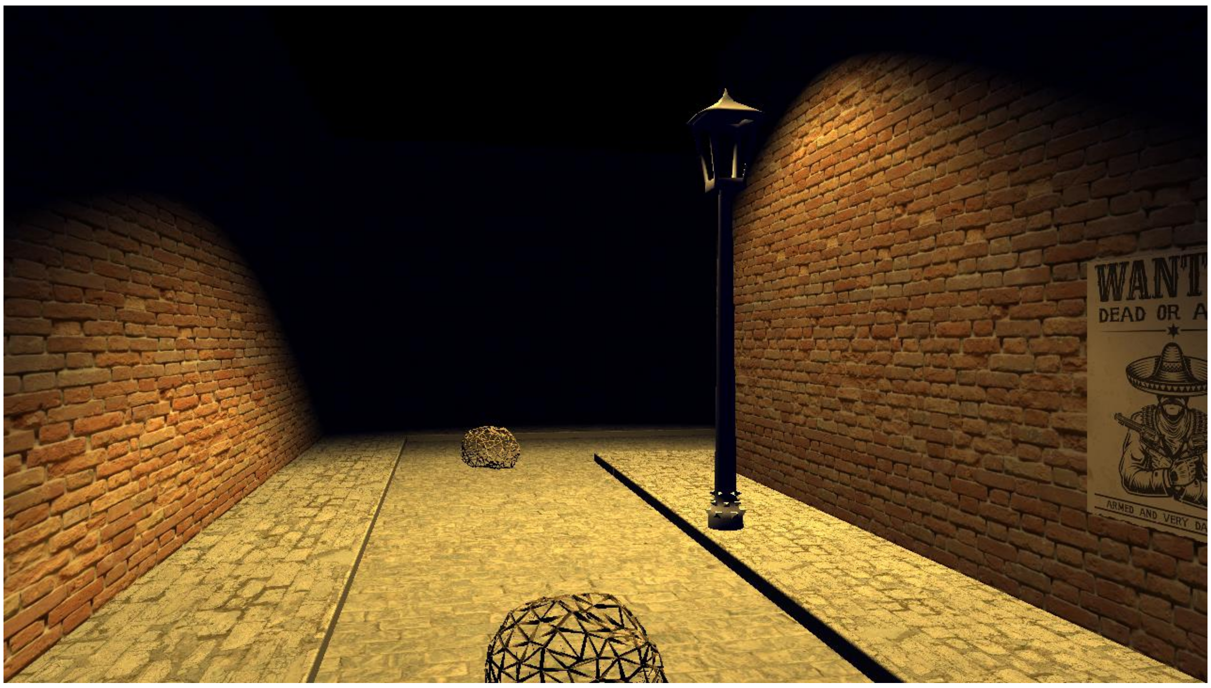
Appendix’s

Appendix 1



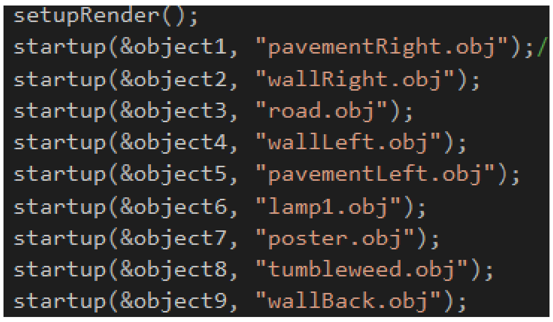
Prototype made within blender.

Appendix 2



OpenGL render

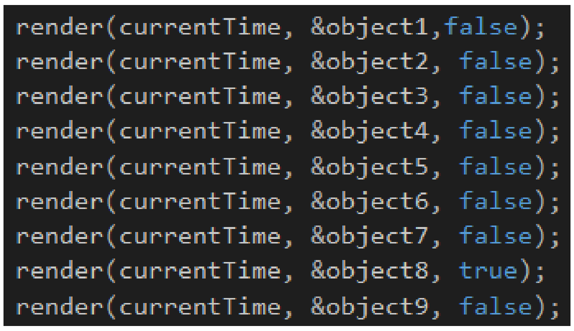
Appendix 3



```
setupRender();
startup(&object1, "pavementRight.obj");/
startup(&object2, "wallRight.obj");
startup(&object3, "road.obj");
startup(&object4, "wallLeft.obj");
startup(&object5, "pavementLeft.obj");
startup(&object6, "lamp1.obj");
startup(&object7, "poster.obj");
startup(&object8, "tumbleweed.obj");
startup(&object9, "wallBack.obj");
```

Start-up of objects.

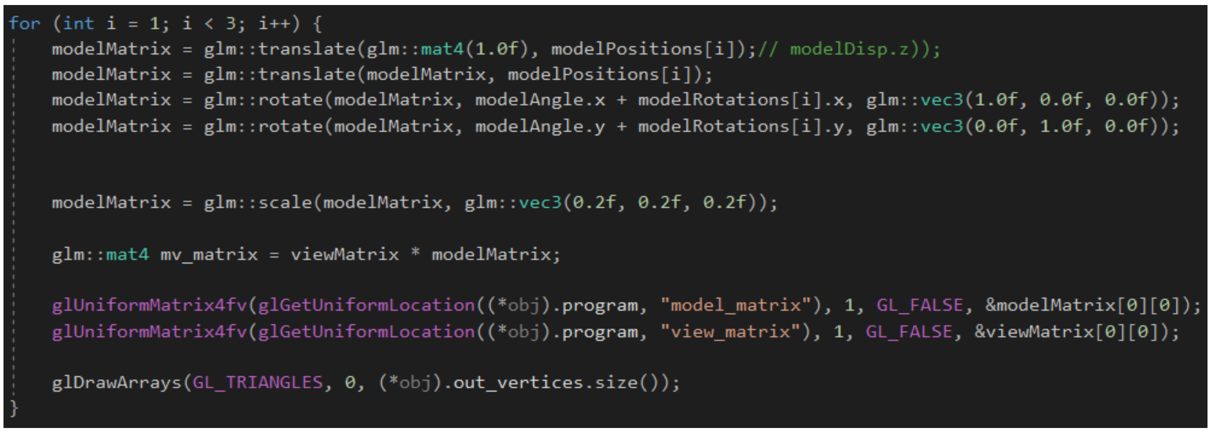
Appendix 4



```
render(currentTime, &object1,false);
render(currentTime, &object2, false);
render(currentTime, &object3, false);
render(currentTime, &object4, false);
render(currentTime, &object5, false);
render(currentTime, &object6, false);
render(currentTime, &object7, false);
render(currentTime, &object8, true);
render(currentTime, &object9, false);
```

Render of objects

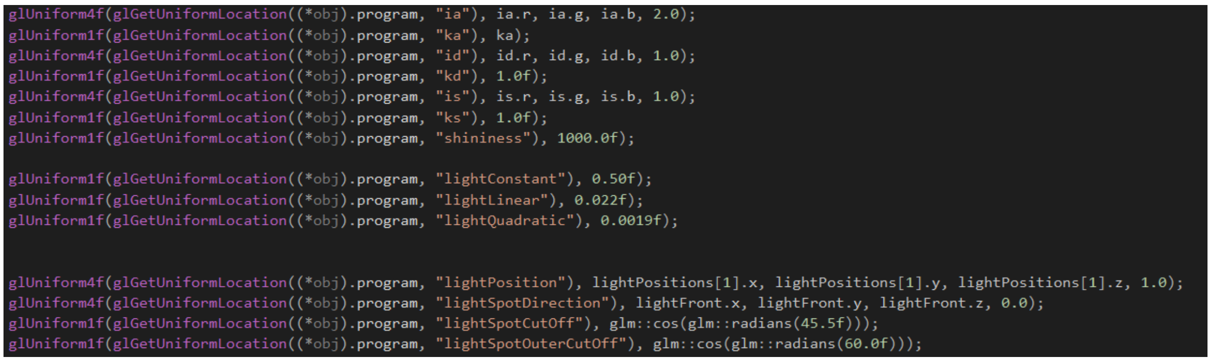
Appendix 5



```
for (int i = 1; i < 3; i++) {
    modelMatrix = glm::translate(glm::mat4(1.0f), modelPositions[i]);// modelDisp.z));
    modelMatrix = glm::translate(modelMatrix, modelPositions[i]);
    modelMatrix = glm::rotate(modelMatrix, modelAngle.x + modelRotations[i].x, glm::vec3(1.0f, 0.0f, 0.0f));
    modelMatrix = glm::rotate(modelMatrix, modelAngle.y + modelRotations[i].y, glm::vec3(0.0f, 1.0f, 0.0f));


    modelMatrix = glm::scale(modelMatrix, glm::vec3(0.2f, 0.2f, 0.2f));

    glm::mat4 mv_matrix = viewMatrix * modelMatrix;

    glUniformMatrix4fv(glGetUniformLocation((*obj).program, "model_matrix"), 1, GL_FALSE, &modelMatrix[0][0]);
    glUniformMatrix4fv(glGetUniformLocation((*obj).program, "view_matrix"), 1, GL_FALSE, &viewMatrix[0][0]);

    glDrawArrays(GL_TRIANGLES, 0, (*obj).out_vertices.size());
}
```

Instancing of the tumbleweed.

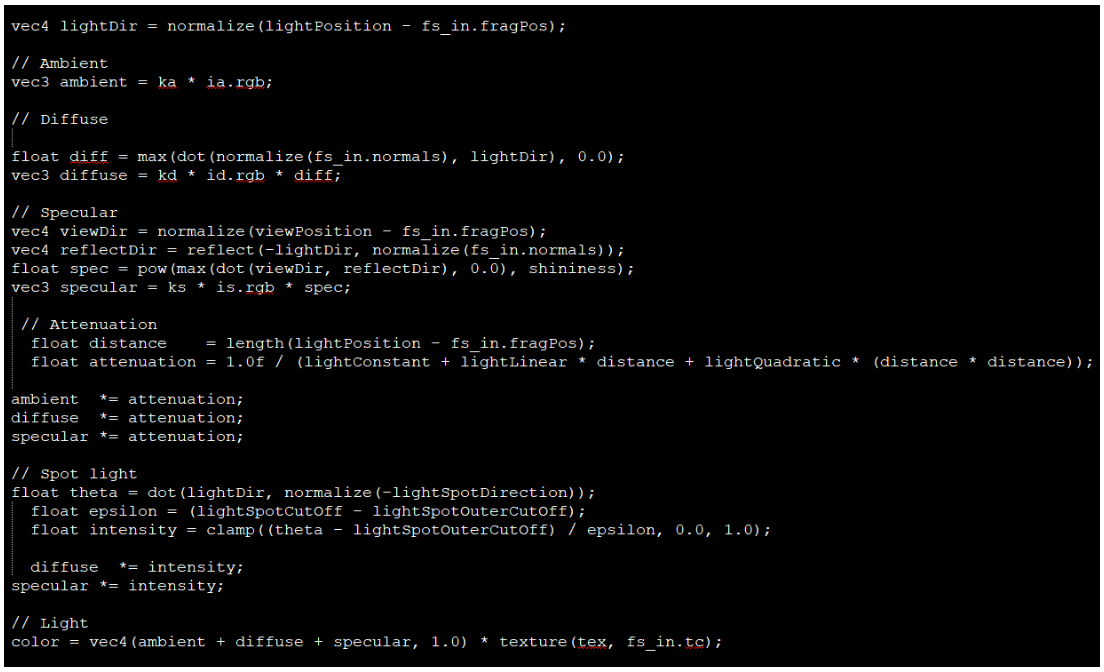
Appendix 6



```
glUniform4f(glGetUniformLocation((*obj).program, "ia"), ia.r, ia.g, ia.b, 2.0);
glUniform1f(glGetUniformLocation((*obj).program, "ka"), ka);
glUniform4f(glGetUniformLocation((*obj).program, "id"), id.r, id.g, id.b, 1.0);
glUniform1f(glGetUniformLocation((*obj).program, "kd"), 1.0f);
glUniform4f(glGetUniformLocation((*obj).program, "is"), is.r, is.g, is.b, 1.0);
glUniform1f(glGetUniformLocation((*obj).program, "ks"), 1.0f);
glUniform1f(glGetUniformLocation((*obj).program, "shininess"), 1000.0f);

glUniform1f(glGetUniformLocation((*obj).program, "lightConstant"), 0.50f);
glUniform1f(glGetUniformLocation((*obj).program, "lightLinear"), 0.022f);
glUniform1f(glGetUniformLocation((*obj).program, "lightQuadratic"), 0.0019f);


glUniform4f(glGetUniformLocation((*obj).program, "lightPosition"), lightPositions[1].x, lightPositions[1].y, lightPositions[1].z, 1.0);
glUniform4f(glGetUniformLocation((*obj).program, "lightSpotDirection"), lightFront.x, lightFront.y, lightFront.z, 0.0);
glUniform1f(glGetUniformLocation((*obj).program, "lightSpotCutOff"), glm::cos(glm::radians(45.5f)));
glUniform1f(glGetUniformLocation((*obj).program, "lightSpotOuterCutOff"), glm::cos(glm::radians(60.0f)));
```

Rendering the light for each object.

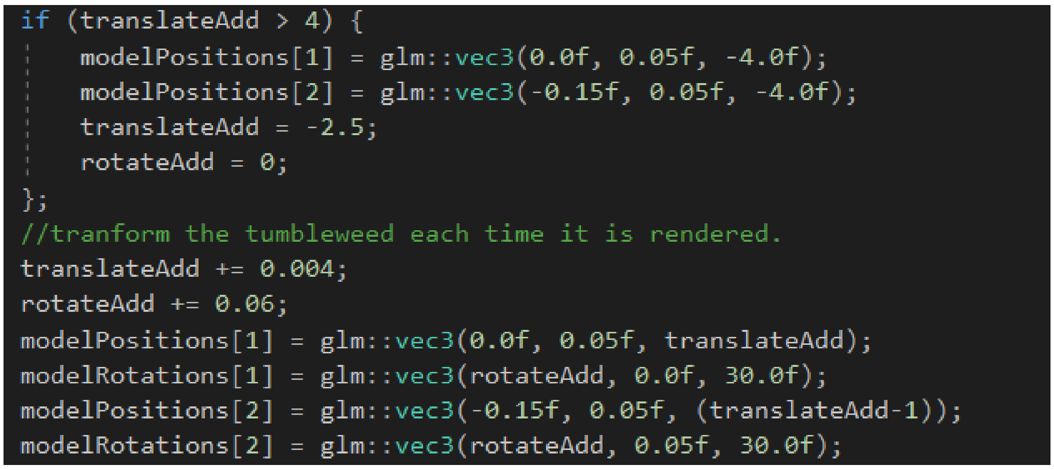
Appendix 7



```
vec4 lightDir = normalize(lightPosition - fs_in.fragPos);

// Ambient
vec3 ambient = ka * ia.rgb;

// Diffuse

float diff = max(dot(normalize(fs_in.normals), lightDir), 0.0);
vec3 diffuse = kd * id.rgb * diff;

// Specular
vec4 viewDir = normalize(viewPosition - fs_in.fragPos);
vec4 reflectDir = reflect(-lightDir, normalize(fs_in.normals));
float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
vec3 specular = ks * is.rgb * spec;

 // Attenuation
  float distance    = length(lightPosition - fs_in.fragPos);
  float attenuation = 1.0f / (lightConstant + lightLinear * distance + lightQuadratic * (distance * distance));

ambient  *= attenuation;
diffuse  *= attenuation;
specular *= attenuation;

// Spot light
float theta = dot(lightDir, normalize(-lightSpotDirection));
  float epsilon = (lightSpotCutOff - lightSpotOuterCutOff);
  float intensity = clamp((theta - lightSpotOuterCutOff) / epsilon, 0.0, 1.0);

  diffuse  *= intensity;
specular *= intensity;

// Light
color = vec4(ambient + diffuse + specular, 1.0) * texture(tex, fs_in.tc);
```

Phong's algorithm within the fragment shader

Appendix 8



```
if (translateAdd > 4) {
    modelPositions[1] = glm::vec3(0.0f, 0.05f, -4.0f);
    modelPositions[2] = glm::vec3(-0.15f, 0.05f, -4.0f);
    translateAdd = -2.5;
    rotateAdd = 0;
};
//tranform the tumbleweed each time it is rendered.
translateAdd += 0.004;
rotateAdd += 0.06;
modelPositions[1] = glm::vec3(0.0f, 0.05f, translateAdd);
modelRotations[1] = glm::vec3(rotateAdd, 0.0f, 30.0f);
modelPositions[2] = glm::vec3(-0.15f, 0.05f, (translateAdd-1));
modelRotations[2] = glm::vec3(rotateAdd, 0.05f, 30.0f);
```

Animation of the tumbleweeds