Trouble With The Law

IN3005 - Computer Graphics

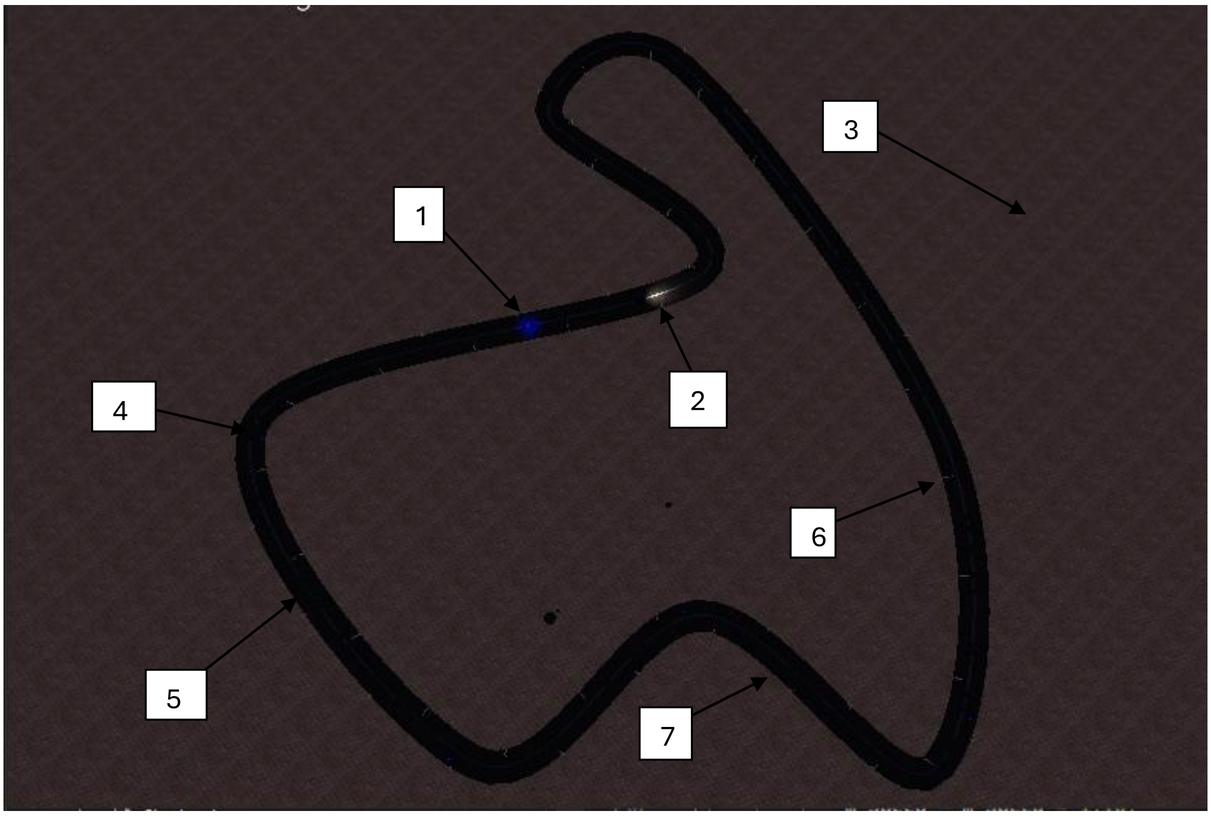
Demetri Michael

Project Concept:

The name of the game is "Trouble With The Law", and the concept is that you are a criminal who has been involved in various crimes and your only hope is to escape the police in your BMW M3. The general idea is the player needs to survive the police chase as long as possible. The longer the player lasts the higher the score they get.

As time progresses in the game the police car gets faster and faster and the only way to speed up is to collect the bright diamonds, so keep collecting them to keep your speed faster than the police car, if he catches up to you it's game over. Watch out so that you don't hit the rocks because they will slow you down and the police car will catch up easier.

Screen Annotation:



The screenshot above displays the path itself, which is draw along a Catmull rom spline and all the **visible** attributes on the path.

1. Shows the police car, has flashing red and blue lights.
2. The player car with a white spotlight.
3. Textured plane
4. Hard to see in screenshot, but we have a speed pickup which increases the speed of the player when hit.
5. Hard to see in screenshot, but we have a rock that slows the player down when hit.
6. Lampposts which are drawn along the track.
7. The track itself is textured and follows the spline.

Controls:

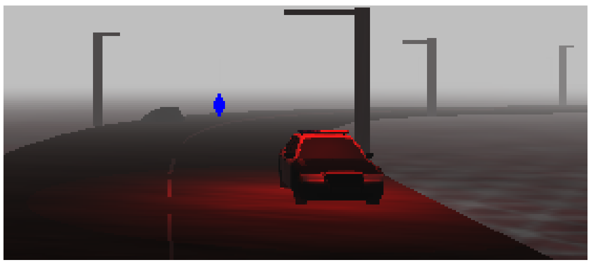
Key Bind	Action
CAPS LOCK	Changed the view of the camera from 3rd person to rear facing camera so that you can see the police car approaching.
Left arrow key (hold or tap)	Moves the players car left
Right arrow key (hold or tap)	Moves the players car right
ESC	Close game
CTRL	Changes to debug/free camera
UI Element	**Explanation**
Score	Displays the players score, the longer time progresses the higher the score will go. Hitting rocks will negatively affect your score.
FPS	Shows you fps
Controls Display	Displays they key binds that are not self-explanatory.
Top Score	Show your score after getting caught by the police

Route and Camera:

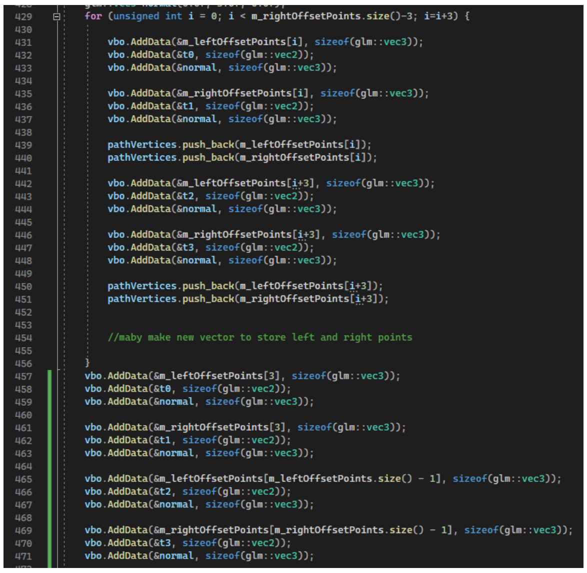
The path was generated along using a Catmull rom spline, as you can see from the annotation on page 2. It is nonlinear however it does not have any verticality to it, and only changes direction on the x,y axis.

The path is correctly textured using a road texture, and triangles were used to form the squares used to texture the path. Additionally, the track correctly responds to the various dynamic lights used in the scene.

The camera views that were created by the author are, 3rd person camera and the rear-view camera. The free camera was also retained. You can use the rear camera to see how close the police car is to catching you.



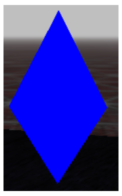
The code snippet below is from the CatmullRom class, shows how the author linked up the track to ensure that the texture did not cut out where the start of the track meets the end of the track. Also note that the author does i+3 so that the texture is drawn across 3 offset points instead of just 1, it makes the texture look smoother and not stretched.



```
for (unsigned int i = 0; i < m_rightOffsetPoints.size()-3; i=i+3) {

    vbo.AddData(&m_leftOffsetPoints[i], sizeof(glm::vec3));
    vbo.AddData(&t0, sizeof(glm::vec2));
    vbo.AddData(&normal, sizeof(glm::vec3));

    vbo.AddData(&m_rightOffsetPoints[i], sizeof(glm::vec3));
    vbo.AddData(&t1, sizeof(glm::vec2));
    vbo.AddData(&normal, sizeof(glm::vec3));

    pathVertices.push_back(m_leftOffsetPoints[i]);
    pathVertices.push_back(m_rightOffsetPoints[i]);

    vbo.AddData(&m_leftOffsetPoints[i+3], sizeof(glm::vec3));
    vbo.AddData(&t2, sizeof(glm::vec2));
    vbo.AddData(&normal, sizeof(glm::vec3));

    vbo.AddData(&m_rightOffsetPoints[i+3], sizeof(glm::vec3));
    vbo.AddData(&t3, sizeof(glm::vec2));
    vbo.AddData(&normal, sizeof(glm::vec3));

    pathVertices.push_back(m_leftOffsetPoints[i+3]);
    pathVertices.push_back(m_rightOffsetPoints[i+3]);


    //maby make new vector to store left and right points

}
vbo.AddData(&m_leftOffsetPoints[3], sizeof(glm::vec3));
vbo.AddData(&t0, sizeof(glm::vec2));
vbo.AddData(&normal, sizeof(glm::vec3));

vbo.AddData(&m_rightOffsetPoints[3], sizeof(glm::vec3));
vbo.AddData(&t1, sizeof(glm::vec2));
vbo.AddData(&normal, sizeof(glm::vec3));

vbo.AddData(&m_leftOffsetPoints[m_leftOffsetPoints.size() - 1], sizeof(glm::vec3));
vbo.AddData(&t2, sizeof(glm::vec2));
vbo.AddData(&normal, sizeof(glm::vec3));

vbo.AddData(&m_rightOffsetPoints[m_rightOffsetPoints.size() - 1], sizeof(glm::vec3));
vbo.AddData(&t3, sizeof(glm::vec2));
vbo.AddData(&normal, sizeof(glm::vec3));
```

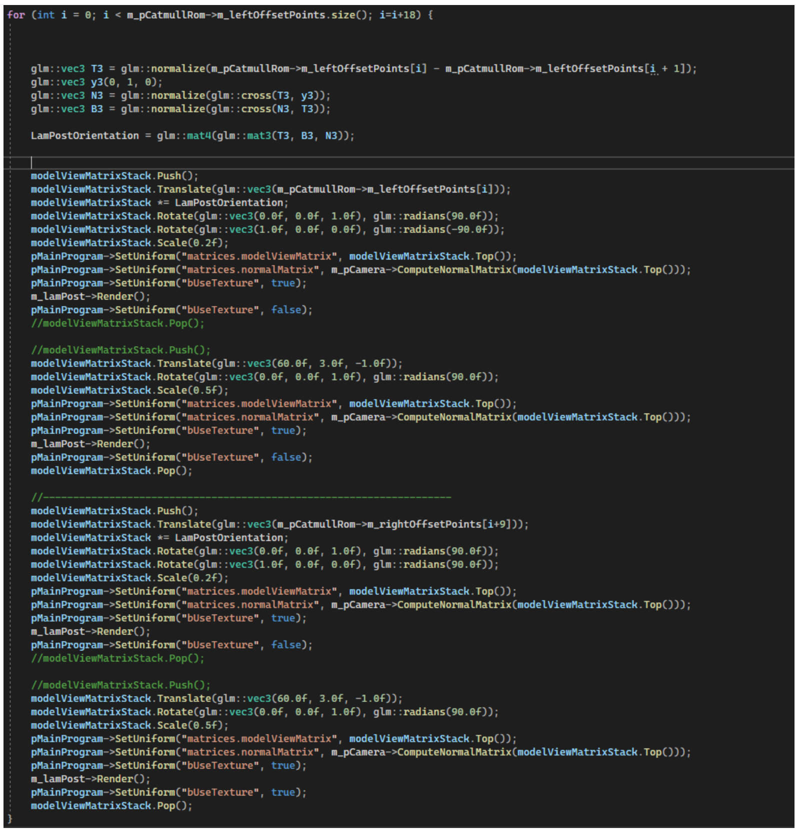
Basic objects, meshes, and lighting.

The two primitive objects created by the author are the diamond which is a pickup for speed boost. It has no texture or lighting and is rendered by its own shader program. It might have been better to give it texturing however the author liked the bright effect it gave off in its current form. See image below.



Secondly the author created the lampposts using primitive objects, it is created using a GL_TRIANGLE_STRIP which forms a rectangle, and then to create the lamppost this object is transforms and scaled down to be in the appropriate position to make the lamppost. Code for this is available in their respective classes lamppost.cpp and Diamond.cpp

Unfortunately, it's not textured because the author had issues with texturing it and couldn't find the cause of the issue, however in the code you can see there was an attempt. Sometimes the lamppost would take the texture of the last textured object. The lampposts are rendered along the spline and have a correct orientation facing into the road. They are placed along the track, appearing on the left side and then the right side. Image below shows how rendering is done on a spline.



```
for (int i = 0; i < m_pCatmullRom->m_leftOffsetPoints.size(); i=i+18) {



    glm::vec3 T3 = glm::normalize(m_pCatmullRom->m_leftOffsetPoints[i] - m_pCatmullRom->m_leftOffsetPoints[i + 1]);
    glm::vec3 y3(0, 1, 0);
    glm::vec3 N3 = glm::normalize(glm::cross(T3, y3));
    glm::vec3 B3 = glm::normalize(glm::cross(N3, T3));

    LamPostOrientation = glm::mat4(glm::mat3(T3, B3, N3));


    modelViewMatrixStack.Push();
    modelViewMatrixStack.Translate(glm::vec3(m_pCatmullRom->m_leftOffsetPoints[i]));
    modelViewMatrixStack *= LamPostOrientation;
    modelViewMatrixStack.Rotate(glm::vec3(0.0f, 0.0f, 1.0f), glm::radians(90.0f));
    modelViewMatrixStack.Rotate(glm::vec3(1.0f, 0.0f, 0.0f), glm::radians(-90.0f));
    modelViewMatrixStack.Scale(0.2f);
    pMainProgram->SetUniform("matrices.modelViewMatrix", modelViewMatrixStack.Top());
    pMainProgram->SetUniform("matrices.normalMatrix", m_pCamera->ComputeNormalMatrix(modelViewMatrixStack.Top()));
    pMainProgram->SetUniform("bUseTexture", true);
    m_lamPost->Render();
    pMainProgram->SetUniform("bUseTexture", false);
    //modelViewMatrixStack.Pop();

    //modelViewMatrixStack.Push();
    modelViewMatrixStack.Translate(glm::vec3(60.0f, 3.0f, -1.0f));
    modelViewMatrixStack.Rotate(glm::vec3(0.0f, 0.0f, 1.0f), glm::radians(90.0f));
    modelViewMatrixStack.Scale(0.5f);
    pMainProgram->SetUniform("matrices.modelViewMatrix", modelViewMatrixStack.Top());
    pMainProgram->SetUniform("matrices.normalMatrix", m_pCamera->ComputeNormalMatrix(modelViewMatrixStack.Top()));
    pMainProgram->SetUniform("bUseTexture", true);
    m_lamPost->Render();
    pMainProgram->SetUniform("bUseTexture", false);
    modelViewMatrixStack.Pop();

    //------------------------------------------------------------------
    modelViewMatrixStack.Push();
    modelViewMatrixStack.Translate(glm::vec3(m_pCatmullRom->m_rightOffsetPoints[i+9]));
    modelViewMatrixStack *= LamPostOrientation;
    modelViewMatrixStack.Rotate(glm::vec3(0.0f, 0.0f, 1.0f), glm::radians(90.0f));
    modelViewMatrixStack.Rotate(glm::vec3(1.0f, 0.0f, 0.0f), glm::radians(90.0f));
    modelViewMatrixStack.Scale(0.2f);
    pMainProgram->SetUniform("matrices.modelViewMatrix", modelViewMatrixStack.Top());
    pMainProgram->SetUniform("matrices.normalMatrix", m_pCamera->ComputeNormalMatrix(modelViewMatrixStack.Top()));
    pMainProgram->SetUniform("bUseTexture", true);
    m_lamPost->Render();
    pMainProgram->SetUniform("bUseTexture", false);
    //modelViewMatrixStack.Pop();

    //modelViewMatrixStack.Push();
    modelViewMatrixStack.Translate(glm::vec3(60.0f, 3.0f, -1.0f));
    modelViewMatrixStack.Rotate(glm::vec3(0.0f, 0.0f, 1.0f), glm::radians(90.0f));
    modelViewMatrixStack.Scale(0.5f);
    pMainProgram->SetUniform("matrices.modelViewMatrix", modelViewMatrixStack.Top());
    pMainProgram->SetUniform("matrices.normalMatrix", m_pCamera->ComputeNormalMatrix(modelViewMatrixStack.Top()));
    pMainProgram->SetUniform("bUseTexture", true);
    m_lamPost->Render();
    pMainProgram->SetUniform("bUseTexture", true);
    modelViewMatrixStack.Pop();
}
```

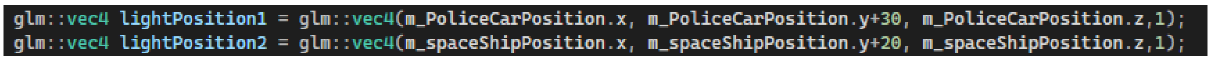
The author imported 3 meshes. The police car, the player car, and the rock. The cars have been transformed and correctly scaled to follow the track and always point in the direction of movement which is forward. The rock has been transformed and used multiple times as an obstacle on the track and there are many instances of it on the track.

Its important to note that the mesh for the rock game in a pack of 3 other rocks, so blender was used to remove the other rocks, leaving one rock that was scaled and transformed correctly within the code.

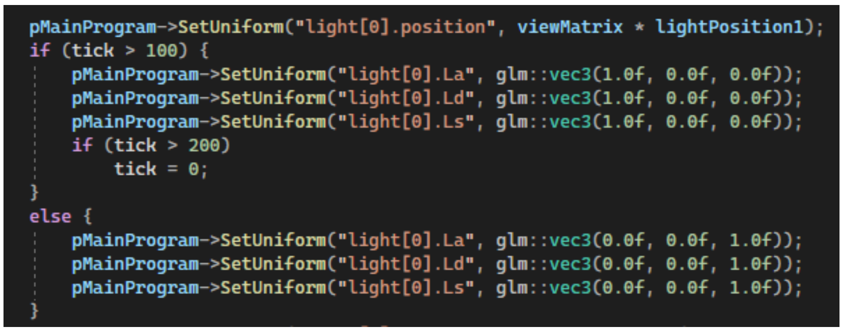
Lighting

There are 2 instances of dynamic lighting, first we have the spotlight for the players car, it points in front of the car even when the car changes direction, this is done using the cars orientation which is calculated using the TNB frame.

Next there is the police cars lights which flash blue and red. Once again, they effect the surroundings with their lighting, the lights are implemented by attaching them 30.0f units above the police car to give a nice effect. There is a tick counter that is responsible for measuring the ticks of the application, every 100 ticks the light of the police car changes from 100 to 200, then the tick counter is reset. See how this was done below. Note that lights were stored as an array in the shader.

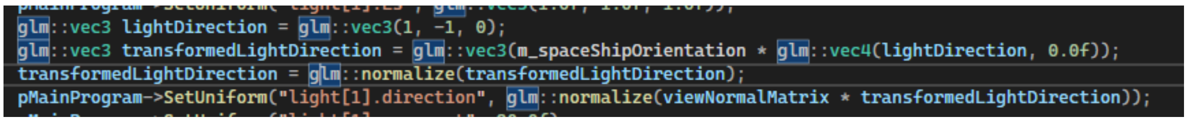


```
glm::vec4 lightPosition1 = glm::vec4(m_PoliceCarPosition.x, m_PoliceCarPosition.y+30, m_PoliceCarPosition.z,1);
glm::vec4 lightPosition2 = glm::vec4(m_spaceShipPosition.x, m_spaceShipPosition.y+20, m_spaceShipPosition.z,1);
```



```
pMainProgram->SetUniform("light[0].position", viewMatrix * lightPosition1);
if (tick > 100) {
    pMainProgram->SetUniform("light[0].La", glm::vec3(1.0f, 0.0f, 0.0f));
    pMainProgram->SetUniform("light[0].Ld", glm::vec3(1.0f, 0.0f, 0.0f));
    pMainProgram->SetUniform("light[0].Ls", glm::vec3(1.0f, 0.0f, 0.0f));
    if (tick > 200)
        tick = 0;
}
else {
    pMainProgram->SetUniform("light[0].La", glm::vec3(0.0f, 0.0f, 1.0f));
    pMainProgram->SetUniform("light[0].Ld", glm::vec3(0.0f, 0.0f, 1.0f));
    pMainProgram->SetUniform("light[0].Ls", glm::vec3(0.0f, 0.0f, 1.0f));
}
```

The image below shows how the author kept the light facing forward.

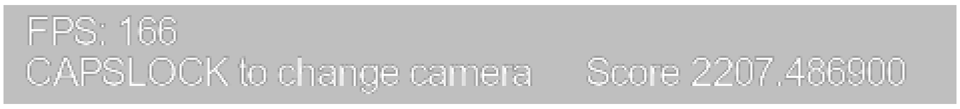


```
glm::vec3 lightDirection = glm::vec3(1, -1, 0);
glm::vec3 transformedLightDirection = glm::vec3(m_spaceShipOrientation * glm::vec4(lightDirection, 0.0f));
transformedLightDirection = glm::normalize(transformedLightDirection);
pMainProgram->SetUniform("light[1].direction", glm::normalize(viewNormalMatrix * transformedLightDirection));
```

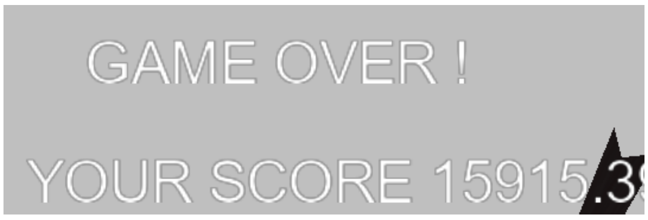
Head's up display (HUD), gameplay, and advanced rendering

The Hud displays the following information:

- Keybind to change camera view.
- Current Score.
- FPS
- Game Over Screen
- End game score.



FPS: 166
CAPSLOCK to change camera   Score 2207.486900



GAME OVER !

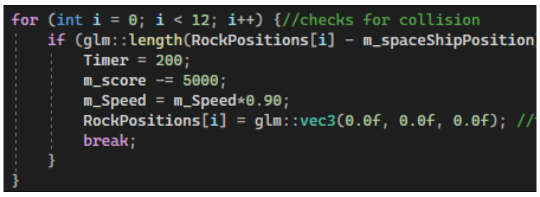
YOUR SCORE 15915.3

Gameplay

When it comes to gameplay, the objective is to avoid the rock obstacles (slow you down), while trying to collect the Diamonds (speed you up). The police car gets exponentially faster as time progresses, so the player needs to pick up diamonds to survive. The main objective is simply to survive as long as possible, and the players performance is measured with a score counter that increases with time and with pickups and is decreased by hitting rocks. The player can move left and right to avoid and pick up objects.

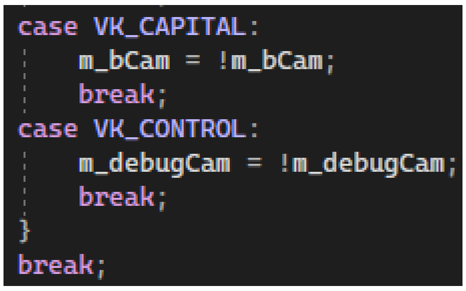
After a certain amount of time new objects will be placed on the track to give the player a chance to speed up further.

Basic collision detection using vectors that store the positions of objects, and a for loop that checks if the objects are close to the player. Scores are increased and decreased when collisions happen, one example below.



```
for (int i = 0; i < 12; i++) {//checks for collision
    if (glm::length(RockPositions[i] - m_spaceShipPosition
        Timer = 200;
        m_score -= 5000;
        m_Speed = m_Speed*0.90;
        RockPositions[i] = glm::vec3(0.0f, 0.0f, 0.0f); //
        break;
    }
}
```

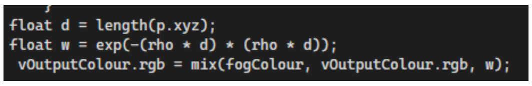
And key binds change camera views,



```
case VK_CAPITAL:
    m_bCam = !m_bCam;
    break;
case VK_CONTROL:
    m_debugCam = !m_debugCam;
    break;
}
break;
```

Advanced Rendering

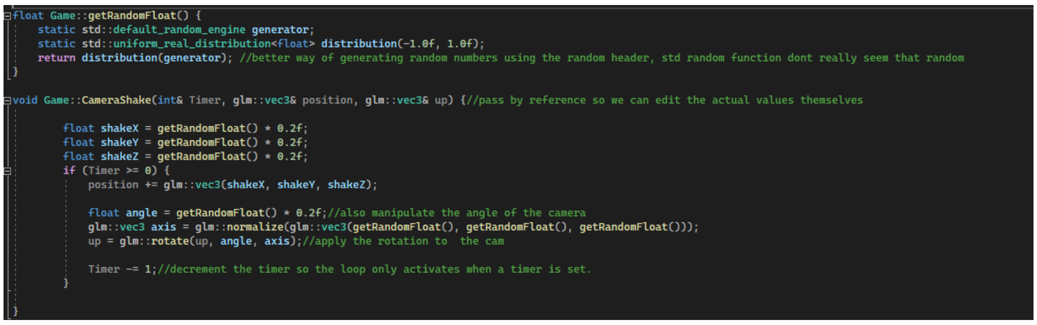
The first advanced rendering method used was **fog**. It was implemented inside the mainshader.frag and is calculated using square exponential. The fog makes it so that the player cannot see the upcoming track and cannot anticipate the turns of rocks that are ahead as easily. The code for fog is below.



```
float d = length(p.xyz);
float w = exp(-(rho * d) * (rho * d));
 vOutputColour.rgb = mix(fogColour, vOutputColour.rgb, w);
```

The second advanced rendering feature was **Screen shake.** The author did this by calling a function before the camera is set called CameraShake(int& Timer, glm::vec3& position, glm::vec3& viewPoint, glm::vec3& up).

It takes the camera position and up vector as well as a timer, then generates a semi-random float for x,y,z screen shake amounts. Random values are generated for the angle of the camera so the position of the camera and angles can be affected by the screen shake. The function is triggered when a rock is hit and then a timer is set to 300, the timer trigger a for loop in the function that does the logic. See below the main part of the screen shake.



```
float Game::getRandomFloat() {
    static std::default_random_engine generator;
    static std::uniform_real_distribution<float> distribution(-1.0f, 1.0f);
    return distribution(generator); //better way of generating random numbers using the random header, std random function dont really seem that random
}

void Game::CameraShake(int& Timer, glm::vec3& position, glm::vec3& up) {//pass by reference so we can edit the actual values themselves

        float shakeX = getRandomFloat() * 0.2f;
        float shakeY = getRandomFloat() * 0.2f;
        float shakeZ = getRandomFloat() * 0.2f;
        if (Timer >= 0) {
            position += glm::vec3(shakeX, shakeY, shakeZ);

            float angle = getRandomFloat() * 0.2f;//also manipulate the angle of the camera
            glm::vec3 axis = glm::normalize(glm::vec3(getRandomFloat(), getRandomFloat(), getRandomFloat()));
            up = glm::rotate(up, angle, axis);//apply the rotation to  the cam

            Timer -= 1;//decrement the timer so the loop only activates when a timer is set.
        }

}
```

Instance Rendering

Unfortunately, the author didn’t manage to implement this. Sending an array to the shader would break everything in such a way that no other textures would be rendered, and transformations send to the shader would affect all the objects in the game, the author wasn’t able to figure out the cause of this issue in the end so omitted it from the code due to the issues it caused.

Used Assets

Name/Info	URL	Date Of Download	Licence
Road Texture	https://www.freepik.com/free-photo/black-gypsum-wall_1037501.htm#query=asphalt%20texture%20seamless&position=1&from_view=keyword&track=ais&uuid=dad16982-4819-4efd-b576-3032b7b4c1f1#position=1&query=asphalt%20texture%20seamless	28/03/2024	Free Licence (freepik)
Police car mesh	https://www.cgtrader.com/free-3d-models/car/car/police-car-ecf76ed6-fed8-4203-96a7-7cddacad1484	04/04/2024	Editorial No Ai License
Player car mesh	https://casual-effects.com/data/	04/04/2024	CC0/Public Domain
Lamppost Texture	https://www.freepik.com/free-photo/abstract-metallic-surface-close-up_12336139.htm#fromView=search&page=1&position=19&uuid=a20617f6-8f6a-42ab-a6e7-9c14d01abc41	13/05/2024	Free Licence (freepik)
Rock Mesh	https://opengameart.org/content/low-poly-stone-pack	30/04/2024	Public Domain
C++ randomization header	https://cplusplus.com/reference/random/	n/a	n/a

This header was much better at doing randomisation than what is in the standard library, standard header seemed to have poor randomization compared to this.

Conclusion:

The author believes that the concept of the game was solid and has lots of potential for interesting gameplay features. The current project does demonstrate many OpenGL and graphics fundamentals however there is room for improvement. Firstly, the use of particle effects would have been a great edition to the game. Exhaust flames or tire smoke is one idea. The addition of a better, more interesting track would have also greatly improved the players experience, the current track does not move across the Z axis (verticality). Quality of life features that could improve performance and usability such as instance rendering, and a restart button are also valuable. However, the author had issues with instance rendering, namely sending orientation of an object to the shader so this could not be done. The environment was also bland, some sort of dynamic moving objects in the background or different meshes would have improved the game.

There are some positive take aways from the project such as the lighting which the author thought has a positive effect on the environment. When driving you can see the reflection of the light on the road and when looking back using the rear camera the red and blue flashing lights really adds some realism to the game. Additionally, the concept of getting chased by a police car and having to dodge obstacles that when hit have a rigorous screen shaking effect do a great deal at giving the game a next level of excitement when playing. Lastly the author's implementation of the three meshes where good because not only where the meshes a good fit for the game setting, but they followed the path reasonably well especially with the forward movement, left and right movement had room for improvement. The rock mesh was hard to see unless the headlight was shining on it, which is another immersive feature, making collisions more interesting.

Overall, the game was a success, the concept was solid and there is a good base to build from, the author really enjoyed manipulating the camera with screen shaking and adding lighting effects, so this paves the way for future work in these areas, hopefully with blur effects, shadows and particle animations.

See full code more details and features.