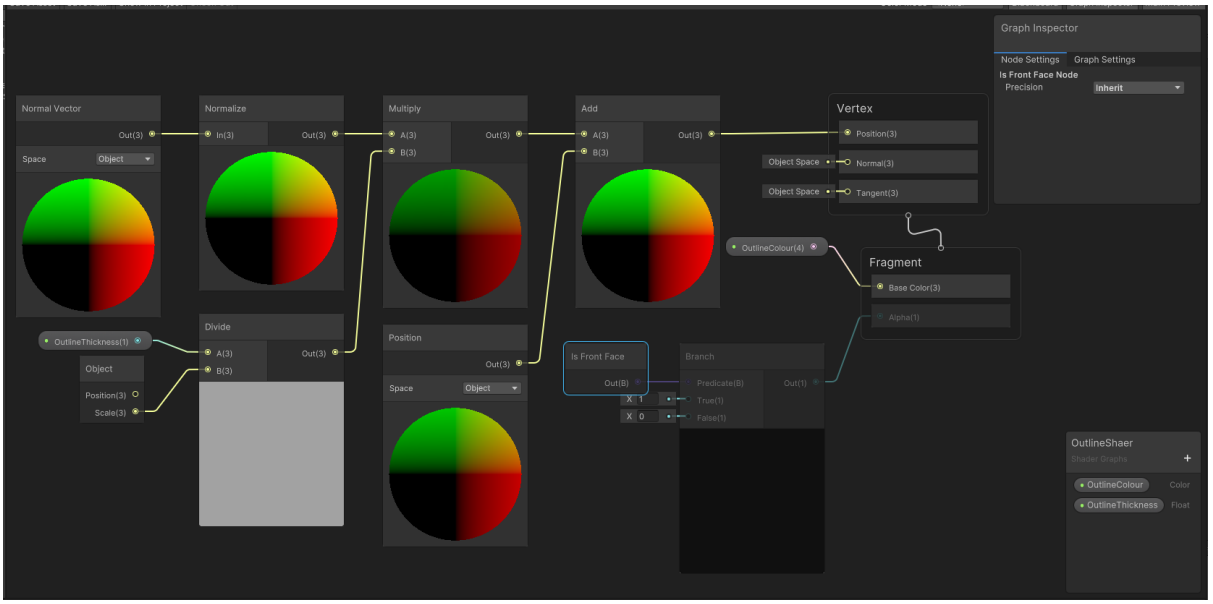
Edge Detector

In order to implement an image processing method that can detect edges in real-time, the Edge Detection Shader project in Unity was created as part of a university course. The project was challenging, but it was also thrilling and rewarding. Understanding the mathematical operations and image processing principles needed to implement the edge detection shader was one of the difficulties. Understanding the edge detection techniques and algorithms required intensive research. The concepts were then put into practice using the Unity visual shader language.

Since the edge detection shader required real-time processing of the input image, optimising the application's performance presented another challenge. However, in order to boost performance, the shader code was optimised and methods like texture filtering and mipmapping were applied. The project also required the discovery and correction of design errors and bugs. Feedback from the professor and other students was requested, which made it easier to find and fix bugs and flaws.

The Edge Detection Shader project provided opportunities for the growth and improvement of numerous skills. Real-time applications for image processing techniques and algorithms were learned. The ability to solve problems has improved, especially in terms of performance optimization. Incorporating suggestions from peers and the professor's feedback helped to improve communication and collaboration skills. When pursuing a career in game development, these abilities are crucial.



Using an unlit shader with the universal render pipeline, this was the graph that I ended up with. Firstly we get the object's normals which we normalise to make sure that the outline is the same size throughout the entire model then i divide the outline size custom variable by the size of the model it self, then to make sure the position stay relative to the model, add the outline to the position of the model. Another custom variable for the colour of the outline is passed straight into the fragment colour node.

Moving Target

The creation of the game mechanics and their integration into the Unity engine presented another difficulty. I had to create a physics-based game that precisely represented how the projectile and moving target moved. Additionally, I had to make sure that the game's user interface was simple enough for players to accurately aim and fire.

I ran into a number of bugs and design issues during the development phase that I had to fix. I asked my professor and peers for feedback, which assisted me in recognising and resolving these problems. Additionally, I was able to enhance the user interface and game mechanics thanks to the feedback.

Overall, the "Hit a Moving Target" project was a great opportunity for me to apply my knowledge of game development and learn new skills. The project helped me develop my problem-solving skills because I had to deal with a variety of bugs and design flaws. I was also able to improve my ability to cooperate and communicate as I solicited feedback from my professor and my peers.

```
Vector3 GetHitPoint(Vector3 targetPosition, Vector3 targetSpeed, Vector3 attackerPosition, float bulletSpeed, out float time)
{
    Vector3 q = targetPosition - attackerPosition;
    //Ignoring Y for now. Add gravity compensation later, for more simple formula and clean game design around it
    q.z = 0;
    targetSpeed.z = 0;

    //solving quadratic ecuation from t*t(Vx*Vx + Vy*Vy - S*S) + 2*t*(Vx*Qx)(Vy*Qy) + Qx*Qx + Qy*Qy = 0

    float a = Vector3.Dot(targetSpeed, targetSpeed) - (bulletSpeed * bulletSpeed); //Dot is basicly (targetSpeed.x * targetSpeed.x) + (targetSpeed.y * targetSpeed.y)
    float b = 2 * Vector3.Dot(targetSpeed, q); //Dot is basicly (targetSpeed.x * q.x) + (targetSpeed.y * q.y)
    float c = Vector3.Dot(q, q); //Dot is basicly (q.x * q.x) + (q.y * q.y)

    //Discriminant
    float D = Mathf.Sqrt((b * b) - 4 * a * c);

    float t1 = (-b + D) / (2 * a);
    float t2 = (-b - D) / (2 * a);

    Debug.Log("t1: " + t1 + " t2: " + t2);

    time = Mathf.Max(t1, t2);

    Vector3 ret = targetPosition + targetSpeed * time;
    return ret;
}
```

The hardest part of this task was figuring out how to locate the target in relation to the projectile's and the target's own speeds. This is done by first calculating the time difference between the target's current velocity and the time it will take it to travel from one position to a predetermined position in the future using a quadratic equation. It is crucial to remember that when attempting to estimate the amount of time it would take for two objects to intersect, both the projectile speed and the target velocity are crucial.

Scoreboard

The scoreboard project was a university coursework in which an application that displayed a list of players in order of their high scores was designed and implemented. The project required selecting the appropriate programming language and tools for the application, implementing a data structure to store and sort the players' scores, and addressing design flaws and bugs during the development and testing phases.

Choosing the appropriate programming language and tools, implementing the data structure to store and sort scores, and addressing design issues and bugs were some of the challenges the project presented. However, these difficulties offered a chance to develop programming and problem-solving abilities while creating a scalable and effective application.

During the development and testing phases, working with the professor and fellow students improved communication and teamwork abilities, allowing for insightful feedback and direction to raise the project's quality.

Using the C# programming language and Unity, the scoreboard project taught participants how to design and implement applications, implement data structures and algorithms to sort and store data effectively, and improve communication and collaboration skills by soliciting and incorporating feedback.

```
public IEnumerable<Score> GetHighScores()
{
    return sd.scores.OrderByDescending(x=>x.score);
}
```

The main section of code that enables me to order the array of float numbers representing the scores of the various players is this brief snippet. Doing so makes it much simpler to order the ui element. The OrderByDescending is a prebuilt method only available when system.linq is imported to the script.

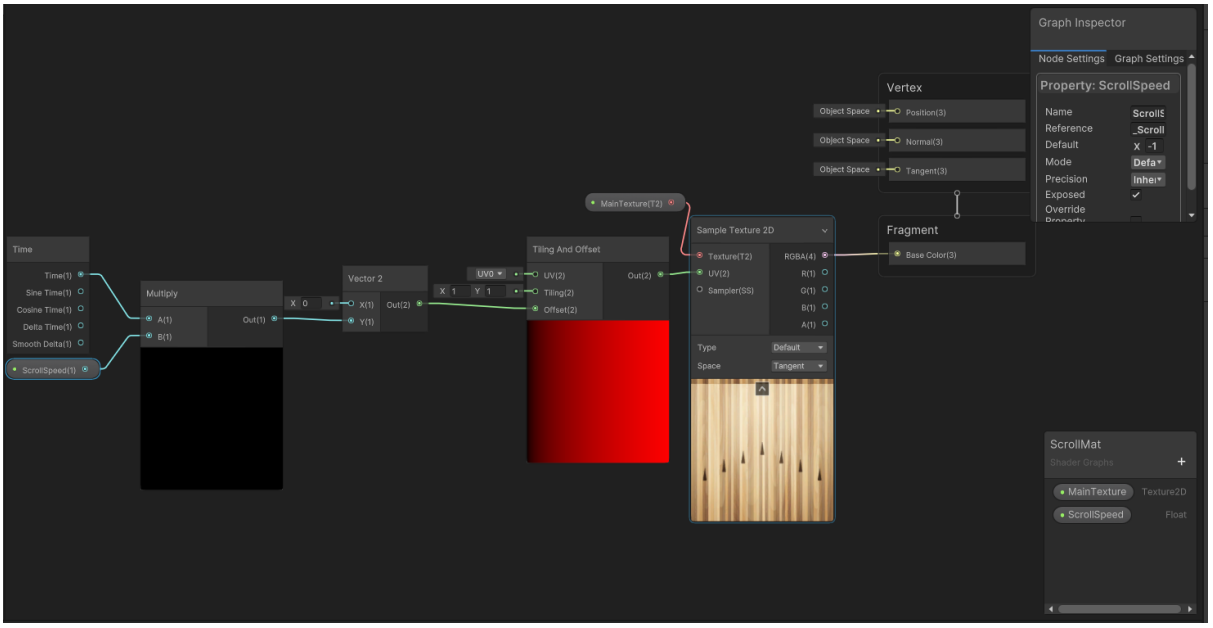
Instanced Scrolling material

For the instanced scrolling material project, it was necessary to develop an application that used instanced rendering to give a material a scrolling effect. The project was a stand-alone assignment for academic credit. In order to customise the scrolling effect, the application needed to be scalable, simple to maintain, and have an intuitive user interface.

Although challenging, the project was exciting. The main difficulty was figuring out how to use instanced rendering to produce a scrolling effect on a material. Efficiency in implementation was achieved using the Unity game engine. The use of a low-poly mesh for the material and modifying the instancing buffer size allowed for the performance optimization of the application to be accomplished. Bugs and design flaws were fixed by asking the professor and the other students for their input.

Despite being a solo project, the quality of the project was improved through collaboration with the professor and classmates during development and testing. While the classmates found and assisted in fixing bugs and design flaws, the professor provided insightful criticism and direction.

Instanced rendering, problem-solving, communication, and collaboration skills were all developed and enhanced as a result of the instanced scrolling material project. The developer's future career as a game developer will benefit from these abilities.



Simply put, the instanced scrolling content uses a time.A texture 2D is offset on either the x or y axis of the image using a delta multiplied by a scrolling speed variable, and this is then applied to the fragment input of the shader to simulate an instance scrolling material. The texture can also be changed in the unity editor with the custom variable.