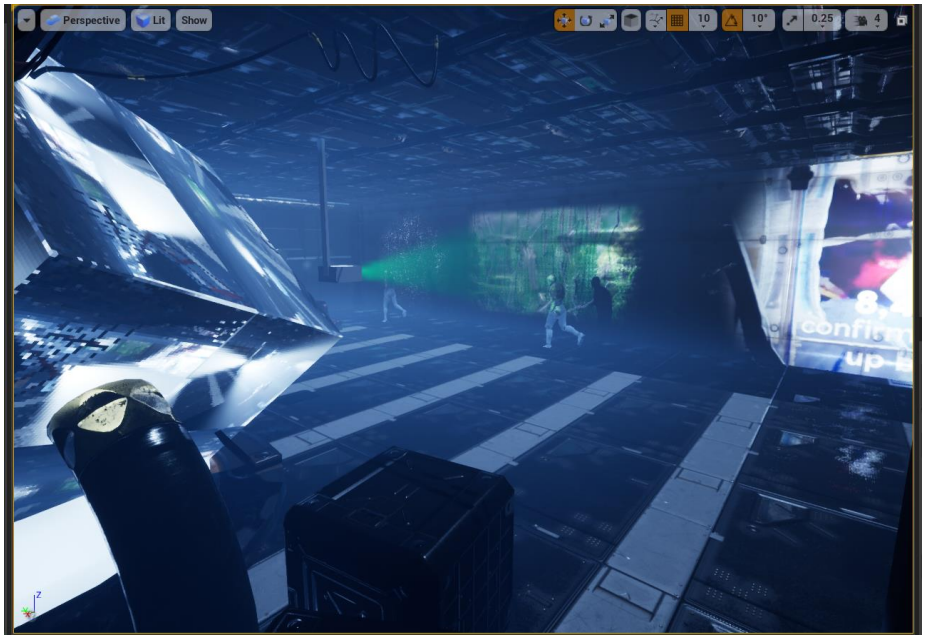
Game Engine Architecture: Projection Report

Ong Wei Yong
Department of Computer Science and Creative Technologies
University of the West of England
Coldharbour Lane
Bristol, UK
wei11.ong@live.uwe.ac.uk



Abstract
The concept of projection has been implemented in many different titles and genres in games. Projection can be used not only for aesthetics or storytelling, but it can be used for indication and gameplay purposes as well. This report will highlight and explore various use cases of the concept of projection in games, mainly through procedural toolsets, shaders, visual effects, and lighting. This report will also serve to evaluate the differences of implementing the concept of projection between game engines, namely Unreal Engine and Unity.

Author Keywords
Projection; Projector; Shader; Unreal Engine 4; Unity; Procedural; Spline; Light

Introduction
This report highlights and details several methods to implement the concept of projection in modern games. A couple of examples could include CCTVs, projectors, and shaders. A procedural tool for designers to use to easily add lights and lamps in a scene could also be implemented. Projection is not limited to a video or image on a flat surface, the concept can also be applied to curved surfaces and objects. Games such as 'Control' and 'Hitman' include projectors and CCTVs throughout their gameplay. They serve to improve storytelling and

to add an extra game mechanic to improve general gameplay.

Background & Research

The concept of projection was discussed through lectures. A projector from the game, 'Control' was presented, and the main task was to create a similar or related effect with several stretch goals, the addition of a shadow onto the projection as well as playing a 'YouTube' video via a link.

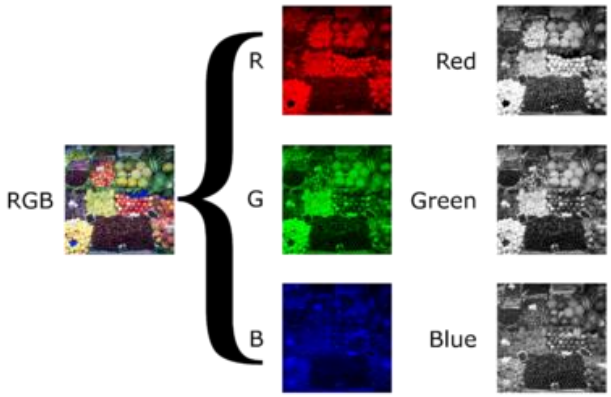


Figure 1. Colour channel mixing and separation.

Primary colours include red, green, and blue or in short, RGB.

The RGB colour channels such as that shown in Figure 1, can be combined, and separated within game engines. Similarly, this applies to lights as well.

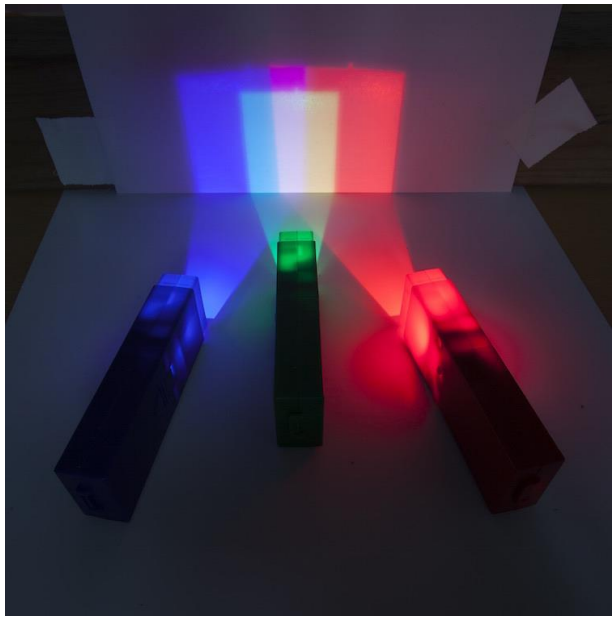


Figure 2. Combination of primary light colours.

By combining the primary light colours, we would achieve white. This is similar to combining the three main colour channels which produce the original colours of the chosen projector image.

With lights, come shadow. Pixels' visibility from a light source are measured by comparing the pixel to a Z-buffer of the light source's view and stored as a texture.

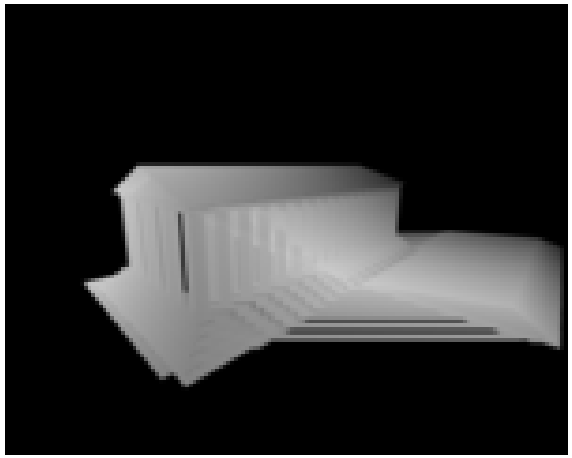


Figure 3. Depth and shadow map projection.

The combination of light and shadow, along with an image or video texture, makes an in-game projector function.

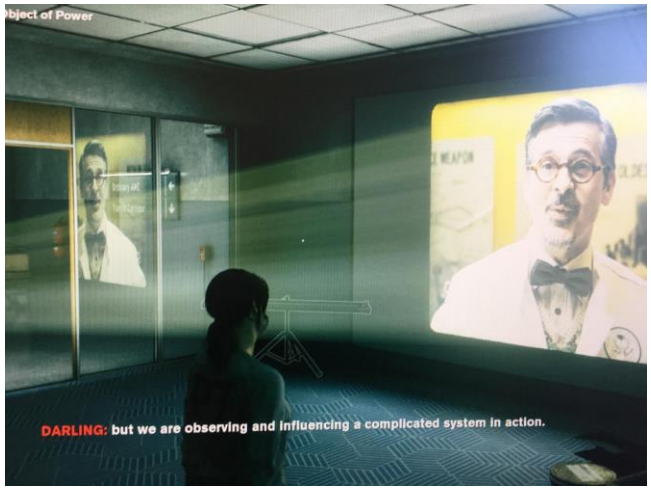


Figure 4. Projector in the game 'Control'.

Another example of using the concept is via CCTVs. CCTVs in games mainly make use of 'Render Targets'. Render Targets allow a 3D scene to be rendered onto a Render Target Texture. Additional effects can also be applied to the texture such as an old CRT/CCTV effect before displaying or applying the texture onto a mesh.

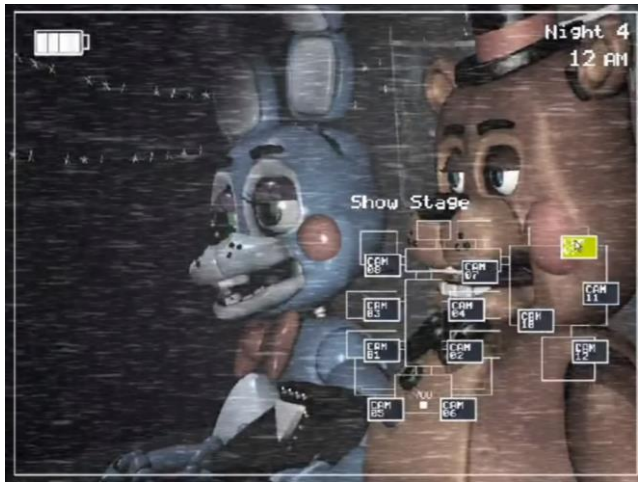


Figure 5. CCTV in the game 'Five Nights at Freddy's'.

Other than CCTVs, Render Targets can also be implemented in mirrors and mini maps.

There can also be a combination of projectors and Render Targets. For example, a render target could be set as a light texture where the image projected from the projector is a real-time scene captured from the CCTV.

Implementations

What shared aspects were there between each of your implementations. A section for each engine on how the selected effect / aspect was implemented and which aspect of each game engine were required.

Unreal Engine Implementation

The first implementation in Unreal Engine was a basic projector. A video file was first imported and made to play while the game was running via a Blueprint Function, a visual scripting feature implemented by Unreal Engine.

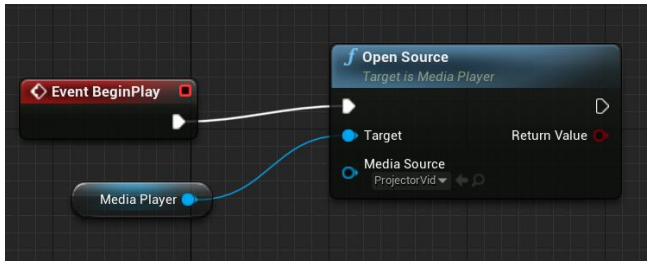


Figure 6. Blueprint Function to play a video.

The video was then added to a material to be edited in the material editor. In the material editor, a rectangular mask was created to achieve the rectangular shape of the projected image instead of the default round shape. The material was then set as a light function, a material to be used with lights. Parameters set in the master material include the blur amount, video height, video width, the ability to change the video as well as the channel colour.

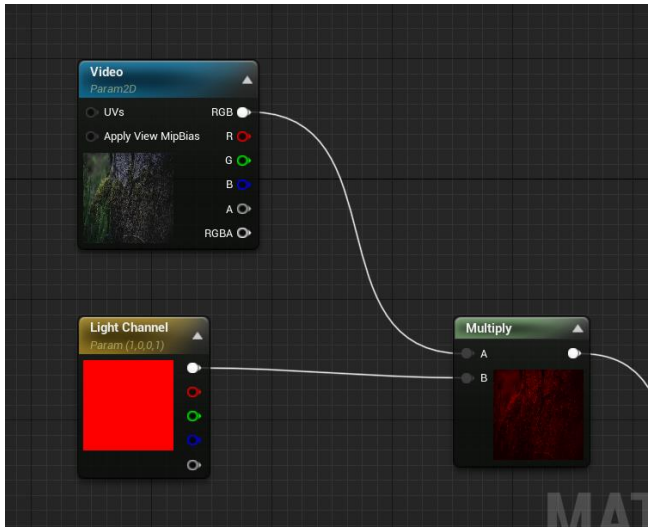


Figure 7. Manipulating Colour Channels in the Material Editor

A material instance was then created from the original material. The instance retains all the parameters of the original master material allowing the user to manipulate its parameters.

Figure 8. Material Instance Parameters

Three lights were then added to the scene. A red light, A green light and a blue light. Attached onto these lights are light functions with the ability to change its colour channel. From Figure 8, the light channel parameter was changed to its respective light colour. For example, a red light would have a light function with a red light channel and a blue light would have a light function with a blue light channel. The three lights would then combine by shining at the same surface and location to produce a complete coloured image with the three combined light and colour channels.

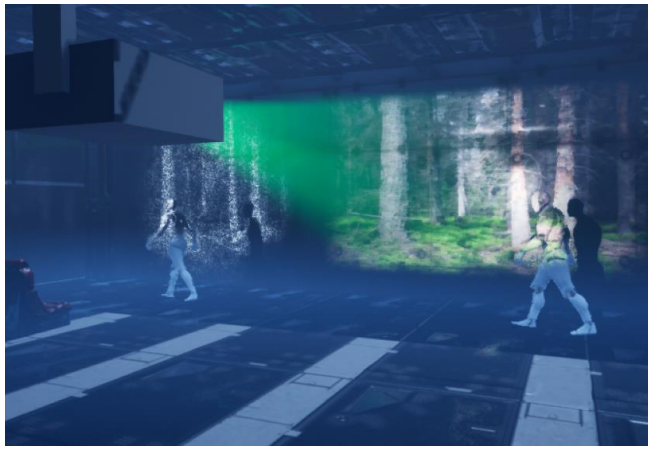


Figure 9. Projectors Implemented in Unreal Engine.

As seen from Figure 9, an extra feature was implemented to enhance the effect of having rays of light emit from the projector.

To achieve this effect, atmospheric fog was added to the scene. Volumetric lights were also implemented by enabling volumetric scattering of the projector lights.

Materials were then created to demonstrate different use cases and expansion of the original master light function material.

Figure 10. Old school Projector.

As seen in Figure 10, an old school effect was created to expand on the original projector material. Brightness, flicker speed, the video texture, line speed and noise speed can be manipulated via the material instance.

Following these effects, a CCTV shader was implemented.

Figure 11. CCTV shader implementation.

The CCTV shader was created by using Render Targets. By using the Scene 2D Capture Tool, a scene was captured and added onto a Render Target. The Render Target was then added to a material so additional effects can be applied. As seen in Figure 11, a CCTV effect material was created using multiple Render Targets. Parameters that were added in the material included brightness, number of active cameras, pixelation, camera activation and the three render target sources.

Materials can also be manipulated in blueprint.

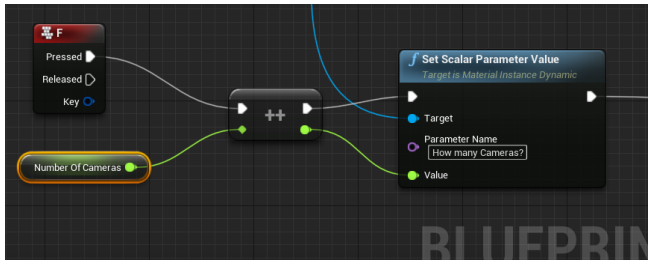


Figure 12. Controlling materials in blueprint.

As seen in Figure 12, every time the 'F' key is pressed, the value attached to the number of cameras will be increased by 1 in the material instance.

The render target and projector implementations can also be combined. To demonstrate this, a projector playing a YouTube video via a link was created.

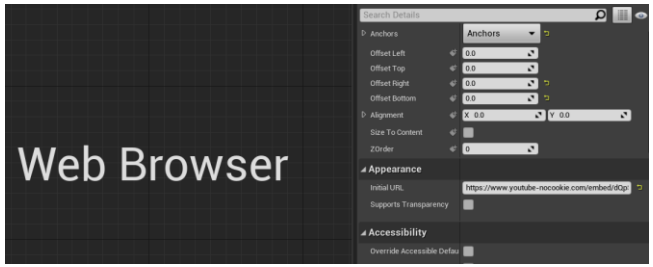


Figure 13. The web browser widget.

A widget blueprint was created containing a browser widget as seen in Figure 13. The YouTube video link was then applied to the 'Initial URL' parameter. The widget was then added as a component onto a blueprint class.

Figure 14. The web browser widget being captured onto a Render Target.

As seen in Figure 14, the web browser blueprint was then captured onto a Render Target and then implemented into a light function material to be displayed by the projector.

In order to ease the design and set dressing of environments, a procedural light projection tool was created.

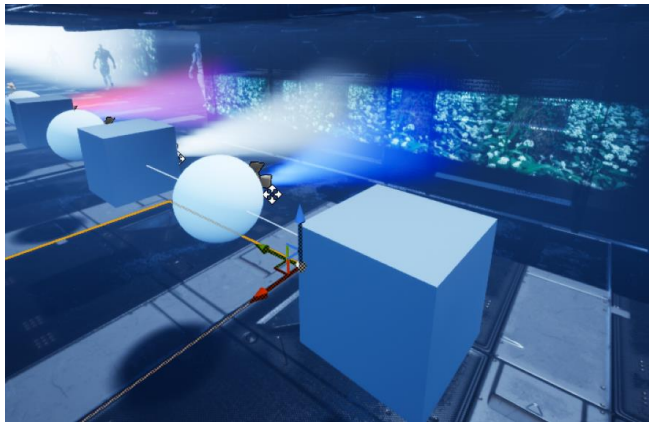


Figure 15. The procedural light projection tool.

The tool allows users to instantiate new light projectors by pulling on spline points. The tool was created with blueprints by using the spline tool. The spline tool includes features like the creation of a Bezier spline and spline points. The base offset was set as the bounding box of the chosen mesh. Added offset was also added and can be later manipulated by the user via the tool's parameters. After the spline point is pulled past a certain distance (length of chosen mesh's bounding box + added offset) a new light projector will spawn.

Parameters such as light mode, animation, rotation speed placed meshes, offset, material, light colour, video scale, projection width and height, blur amount, light scale and cone angle were implemented and are easily manipulated by the user.

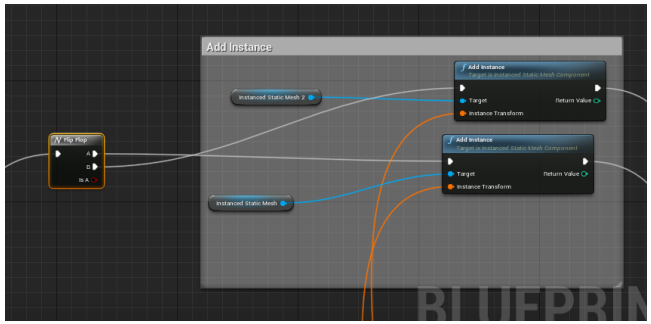


Figure 16. The procedural light projection tool.

Every time the tool instantiates a new light projector, a different mesh can be called by alternating between the two meshes by using the Flip Flop node as seen in Figure 16.

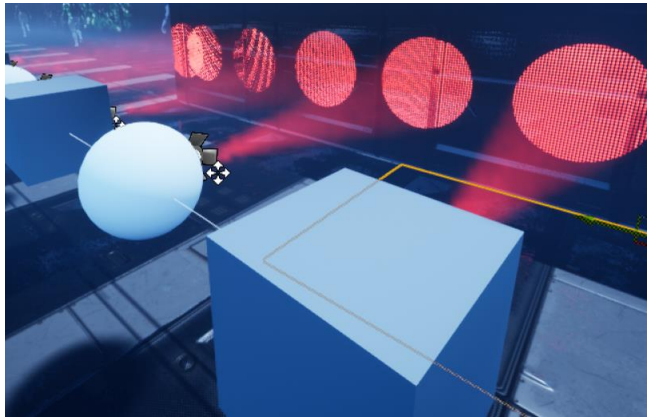


Figure 17. The procedural light projection tool's party venue mode.

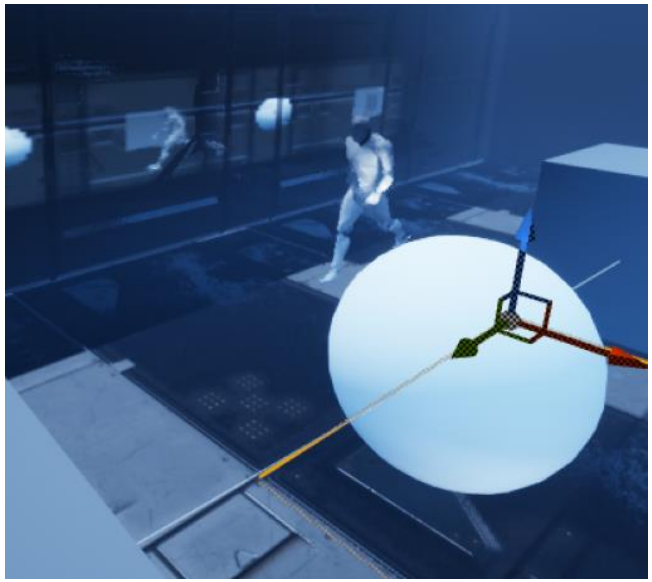


Figure 18. The procedural light projection tool's mirror mode.

There are three light modes. Projector, which instantiates projectors as seen in Figure 15. Party Venue, which spawns a single light and a separate light texture as seen in Figure 17. And Mirror, which spawns a projected mirrors as a light using Render Targets as seen in Figure 18. Animation is also set as an option to animate the light projectors, mainly to supplement the 'Party Venue' mode.

Finally, a post process shader was also implemented.

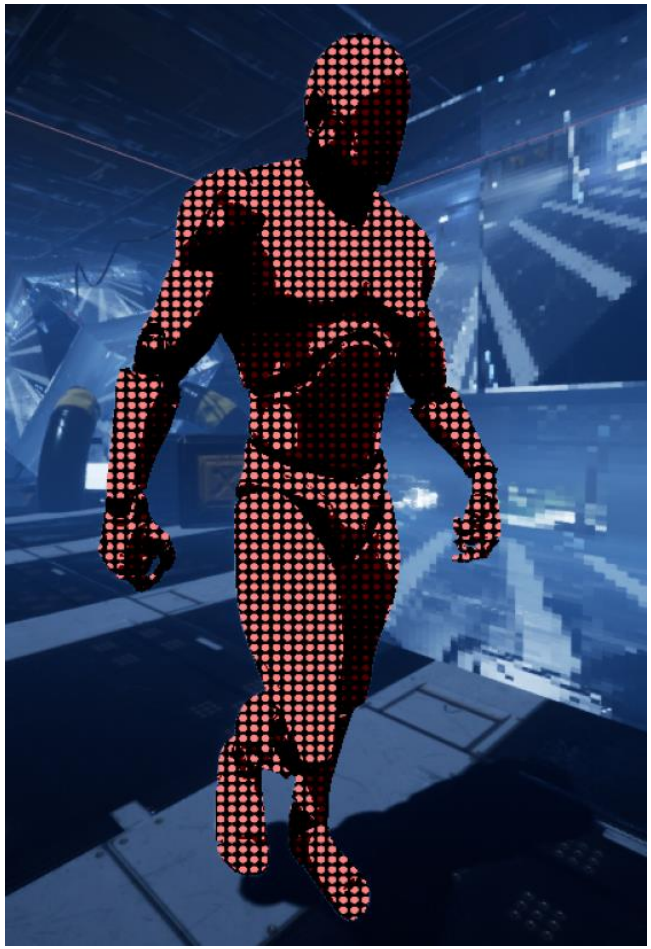


Figure 19. Circles post-process shader.

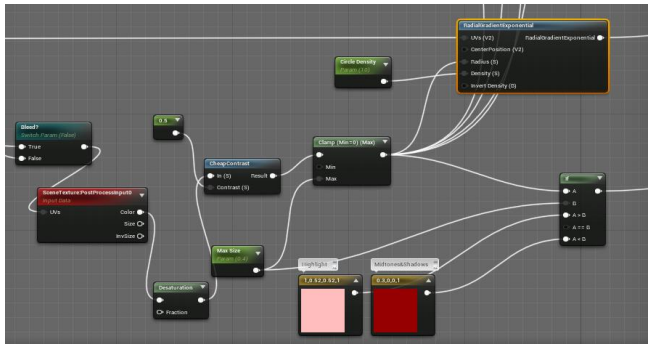


Figure 20. Manipulating circle size by measuring scene brightness.

The shader had circles that differed in sizes depending on its brightness. The brighter the scene section, the larger the circle. The larger circles were also set as a different colour to accentuate the highlights as seen in Figure 19 and 20.

Parameters such as Gradient Density, Max Size, Number of Circles, Make Square and whether the circles bleed were implemented.

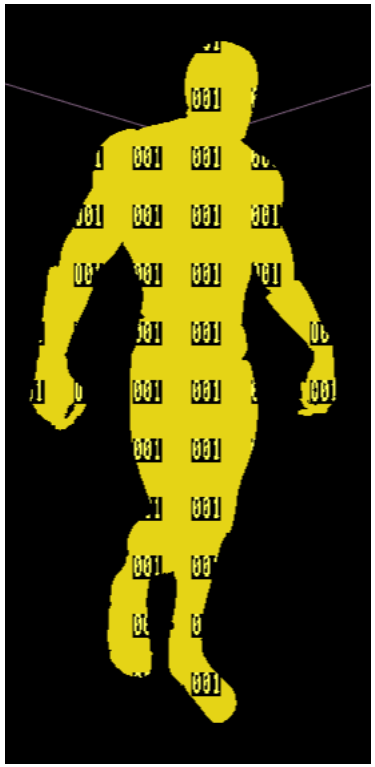


Figure 21. Custom Stencil.

A custom stencil as shown in Figure 21 was also added to the mesh to mask it from the other meshes in the scene. A custom scene depth was also added to occlude the mesh from meshes in front of it. If a custom depth was not implemented, the mesh would still show behind other objects, creating an X-Ray effect.

Unity Implementation
All the features were presented and implemented in the Unity HD Render Pipeline.

The first implementation in Unity is the basic projector.

Figure 22. A basic projector implementation in Unity's HD Render Pipeline.

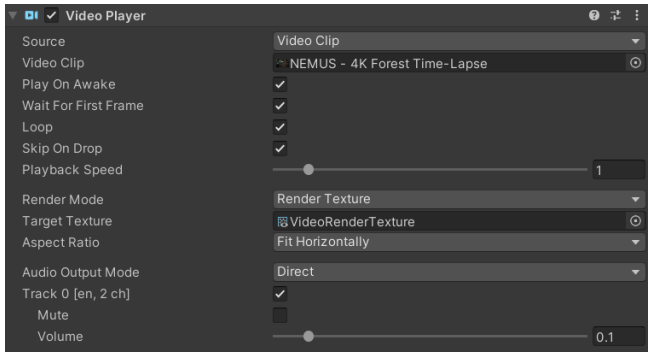


Figure 23. The attached Video Player component.

A video was first imported into Unity. The video was then added to a Video Player Component that is attached to a Game Object as shown in Figure 23. A Render Texture was then created and set as the target texture in the Target Texture Parameter as shown in Figure 23.

The Render Texture was then set under the 'Cookie' parameter under 'Emission' in the light's settings.

To achieve a rectangular shape, the light shape can be changed to pyramid from the default cone shape under the 'Shape' section.

In order to achieve the Light Ray Effects, volumetric lighting was implemented by enabling volumetrics under the 'Volumetrics' section.

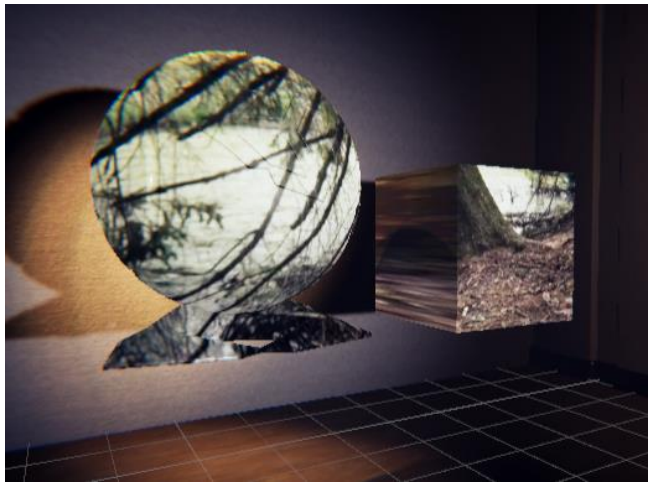


Figure 24. Decal Projections.

Other methods of projection were explored and implemented.

As shown in Figure 24, Decal Projections interact with the scene's lighting and wrap around meshes. To enable decal projections, a HDRP/Decal shader was created and placed into a light with a Decal Projector component.

Figure 25. Waterfall Projection.

The replication of Waterfall Projection was also explored and implemented.

A particle system was added to the scene. There are several notable additions and changes to the default particle system. Render mode was set to 'Stretched Billboard' to achieve a vertical rectangle shape. Emission rate over time and gravity was increased. Collision was also added as seen in Figure 25. To allow light to shine onto the particles, a separate material was created to allow light to shine clearly onto the particles. A light with a video render texture set as its Cookie was then shone onto the particle system to complete the Waterfall Projection effect.

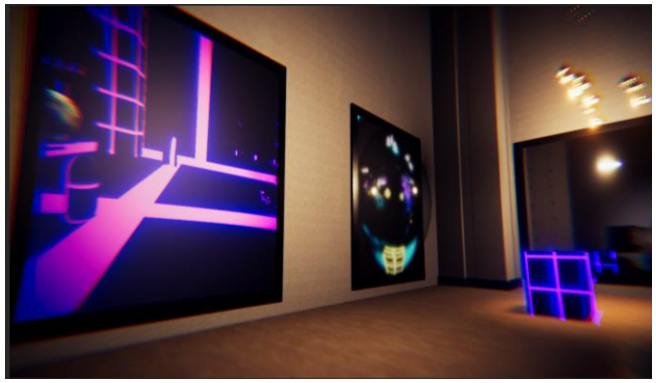


Figure 26. Mirrors with Post Processing Effects.

To demonstrate Render Textures in Unity, several mirrors with Post Processing Effects were implemented.

A Render Texture, Material and Camera to capture the desired scene was created. The Target Texture in the Camera's settings was set as the desired Render Texture to capture the scene onto. The Render Texture was then placed onto the material. The material was then placed onto a rectangular mesh to replicate a mirror effect.

A Post Processing Volume was then added onto each base mirror effect by enveloping the camera capturing the mirrored scene. As shown in Figure 26, two different styles of post processing effects were implemented. One of which had all objects reflected in the mirror have its material replaced by a grid material.

To implement this feature, a material and shader was created.

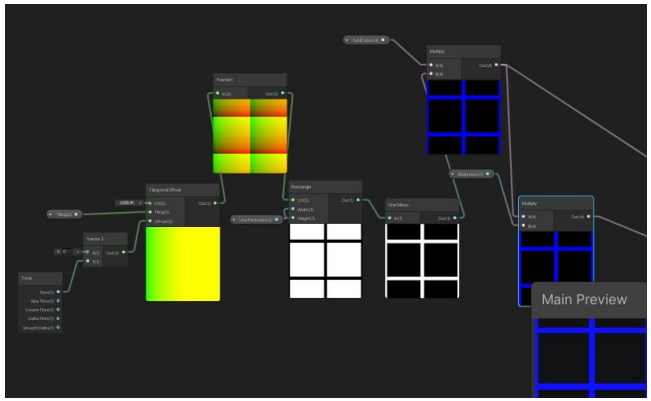


Figure 27. Grid Shader implemented using Unity's Shader Graph.

Using Unity's Shader Graph system, a Grid Shader was created as shown in Figure 27. The material then had its shader set as the Grid Shader that was created earlier.

In order to replace every material reflected in the scene with a Grid Material, a Custom Pass Volume was implemented. Custom Pass Volumes allow users to inject shaders at certain points inside the render loop. The Grid Material was added to the Custom Pass Volume's Override, completing the effect.

The final implemented tool is the Projector Spline Tool.

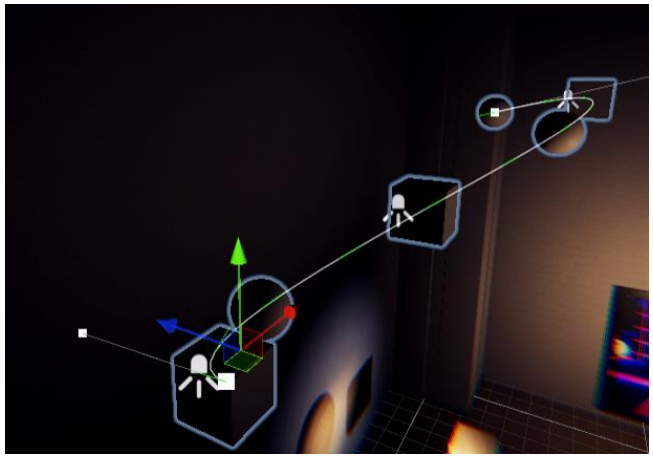


Figure 28. The Projector Spline Tool

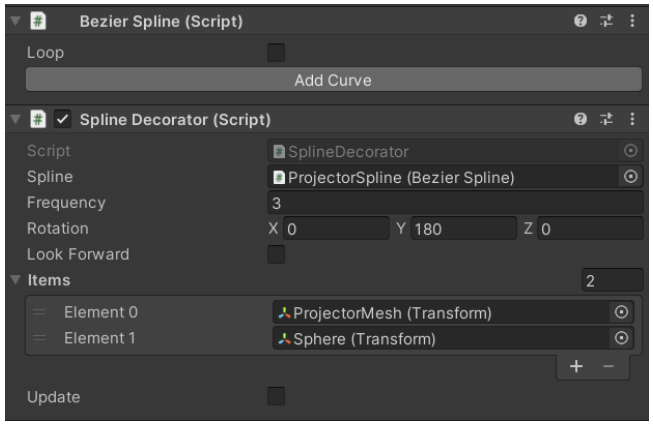


Figure 29. The Projector Spline Tool Parameters

The tool allows users to instantiate new light projectors by pulling on spline points.

```
private void OnSceneGUI () {
    spline = target as BezierSpline;
    handleTransform = spline.transform;
    handleRotation = Tools.pivotRotation == PivotRotation.Local ?
        handleTransform.rotation : Quaternion.identity;

    Vector3 p0 = ShowPoint(0);
    for (int i = 1; i < spline.ControlPointCount; i += 3) {
        Vector3 p1 = ShowPoint(i);
        Vector3 p2 = ShowPoint(i + 1);
        Vector3 p3 = ShowPoint(i + 2);

        Handles.color = Color.gray;
        Handles.DrawLine(p0, p1);
        Handles.DrawLine(p2, p3);

        Handles.DrawBezier(p0, p3, p1, p2, Color.white, null, 2f);
        p0 = p3;
    }
    ShowDirections();
}
```

Figure 30. The Custom Editor Spline Script Snippet

The custom editor script was created to draw the spline onto the screen. The spline involves the drawing of a Bezier Curve, a curve created with four points. Thus, a Bezier Curve Script was created to handle the Bezier Curve calculations.

```
for (int p = 0, f = 0; f < frequency; f++)
    {
        for (int i = 0; i < items.Length; i++, p++)
        {
            item = Instantiate(items[i]) as Transform;
            Vector3 position = spline.GetPoint(p * stepSize);
            item.transform.localPosition = position;
            item.transform.localRotation = Quaternion.Euler(rotation.x, rotation.y, rotation.z);
            if (lookForward)
            {
                item.transform.LookAt(position + spline.GetDirection(p * stepSize));
            }
            item.transform.parent = transform;
        }
    }
```

Figure 31. The Spline Decorator Script Snippet

A Spline Decorator script was also created to instantiate prefabs onto the spline as shown in Figure 31. By enabling ‘Execute In Edit Mode’ via the Spline Decorator Script, the user would be able to make and see changes while in edit mode rather than on game start after clicking the Update Button in the Inspector as shown in Figure 29.

The tool instantiates prefabs rather than static meshes. Animation and other features are dependant on the prefabs themselves. Any number of prefabs can be added to the spline as the user chooses. In this case, a rotation script and video projection were added to one of the prefabs to demonstrate animation and light projection.

Evaluation

Unreal Version: 4.27.1

Unity Version: HD Render Pipeline (2020.3.22f1)

For both engines, both engines performed well in terms of visuals and aesthetics. Both engines made it simple to implement shaders and post processing effects with matching high visual fidelity with Unreal Engine's Material Editor and Unity's Shader Graph. Both engines also made it simple to implement Render Targets/Textures.

In terms of replicating the base projector effect, Unity had a much simpler method of implementation in comparison to Unreal Engine. Unity did not require the use of three separate lights and material instances compared to Unreal Engine's implementation. Unity also allowed for different light shapes compared to Unreal Engine which only had the cone shape.

For the YouTube Implementation, neither engine was able to place YouTube video onto a material. In Unity Engine, there are no features that allow playing of YouTube videos. However, there are third party plug-ins available to download to enable and implement the feature. For Unreal Engine, the Web Browser Widget enables playing of YouTube videos. However, Unreal widgets and blueprints are not allowed to be placed onto a material. The workaround is by using a render target to capture the widget in real time so the render target can be applied to a light function material. Via the YouTube API, the embed links also had to be altered to allow video looping, and to bypass the 'Allow Cookies' window.

The projector spline tool had the most differences. A spline tool was already available in Unreal Engine, making it a lot easier to implement robust splines compared to Unity where a spline tool had to be created and scripted separately. In terms of functionality, Unity was the better engine. Tools could be scripted in Unity to allow prefabs. In Unreal Engine, the built-in Blueprint system did not allow for instantiation of other prefabs/blueprint components. Extra features had to be implemented on the spline tool itself in Unreal Engine compared to Unity where extra features could be implemented on the prefab itself. This made Unity's implementation much more versatile.

In terms of performance, Unity had a better performance compared to Unreal Engine when it came to the main projector effect. This is more noticeable on the spline tool where multiple instances of a projector were spawned in the scene.

In Unreal Engine, all three lights and material instances had to be dynamically instantiated per projector instance compared to a single material and light in a prefab in the Unity implementation.

Shaders in both implementations can be implemented in games which require shaders that can be easily or dynamically altered during gameplay.

The spline tools allow artists and designer to easily set up scenes with light projectors. This is more noticeable in large party venues where many animated lights are required to be setup evenly along a curve or line.

Effects like the implemented ‘Projected Mirror’ effect can also be easily setup using the tool.

Extra features that could be implemented in the projector spline tool could include:

- Adding a feature so the light points at the player for stealth games.
- Adding additional animation styles.
- Adding VFX or Particle Emitters to each projector instance to emit particles or projectiles rather than just light.

Acknowledgements
Thanks to all the lecturers and staff of the University of the West of England involved in supporting the creation of this report.

References

Catlike Coding (no date) *Curves and Splines, making your own path*. Available from: https://catlikecoding.com/unity/tutorials/curves-and-splines/ [Accessed 22 January 2022].

Colour theory (2022) *Wikipedia* [online]. 05 January. Available from: https://en.wikipedia.org/wiki/Color_theory [Accessed 22 January 2022].

Epic Games (2021) *Unreal Engine 4* (4.27.1) [computer program]. Available from: https://www.unrealengine.com/en-US/ [Accessed 22 January 2022].

IO Interactive (2000) *Hitman.* [Video game]. IO Interactive.

Remedy Entertainment (2019) *Control.* [Video game]. 505 Games.

Shadow mapping (2021) *Wikipedia* [online]. 01 December. Available from: https://en.wikipedia.org/wiki/Shadow_mapping [Accessed 22 January 2022].

Unity Technologies (2020) *Unity* (2020.3.22.f1) [computer program]. Available from: https://unity.com/ [Accessed 22 January 2022].