# Evaluating Rendering Techniques & Optimisations for Non-Euclidean Environments

**3rd Year Computer Science Project - Sample**
*Completed 15/05/2024 – Will Scully*

---


# Abstract

Originating from the game Portal, portals are pairs of blue & orange holes that can be used to create non-Euclidean environments by linking locations over a distance via an impossibly shallow doorway. This project tries to re-create their visual effect utilising Geometry Shaders, as opposed to using the Stencil Buffer or Render Textures as other approaches have used previously. In the end, three distinct renderer implementations are used – A single-pass forward renderer, a multi-pass forward renderer, and a deferred shading inspired renderer utilising sub-passes for each level of recursion.

Through testing of frame intervals, it is discovered that the accuracy of each renderer's portal effect improves through each iteration, however there is generally a performance penalty with each improvement. Breaking this trend however, the final renderer sees scene dependent improvements to its performance, alternating between the desktop and laptop platforms, creating an unusual profile.

# 1 - Introduction

Portals at their core concept connect two locations within an environment using an impossibly shallow doorway, thus allowing objects to be moved through each portal seamlessly, as if the two locations were immediately bordering to each other. Being quite an unusual effect, they create a technical challenge regarding the techniques required to render them, in addition to how they influence the environment both around and within them.

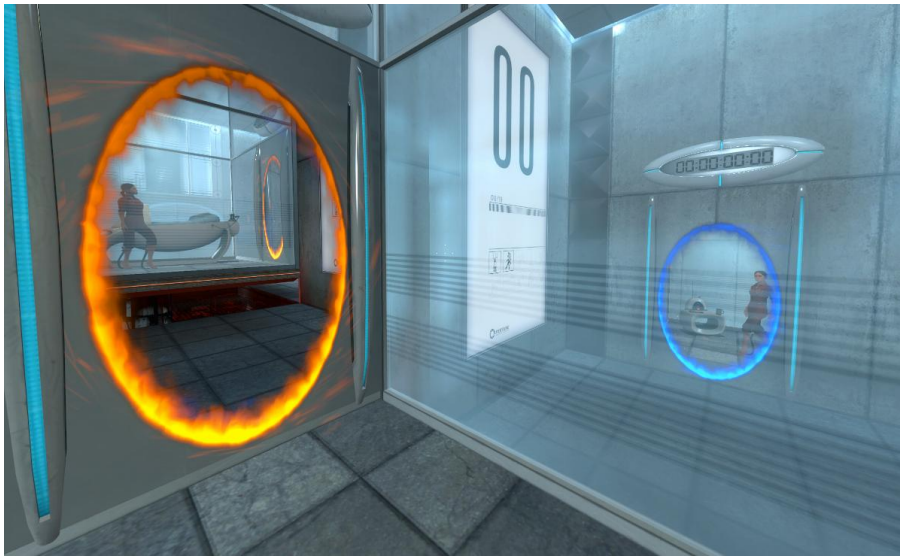

*Image 1: The opening of Portal*
*Portal 1 – In-Game Screenshot (Valve 2007)*

Being introduced as a self-titled game “Portal”, Valve spawned a classic puzzle game series with two entries being released in 2007 & 2011 respectively (Valve 2007). During the original entry’s development, the team iterated upon a few solutions to the rendering problem to create an efficient approach that worked well on the weaker hardware of the time. In-fact, the predecessor to the commercially released game was a university project named “Narbacular Drop”, created by the same team prior to being hired by Valve (Digipen Institute of Technology 2005; Kircher et al. 2007).

The rendering challenge resides in representing non-Euclidean space, being space on a flat plane where distances are no longer linear and our general intuition for how geometry should be represented across a flat plane becomes void (Ball 1960). How can we represent this within a graphics model designed over the years for Euclidean space? The graphics model is built around Euclidean principles, as such, most 3D applications are bound by its space including their optimisations. While the team at Valve had their own solutions and other developers over the years have created variations operating out of Stencil Buffer and Render Texture tricks, this project embarks on reimplementing portals using a different method – Geometry Shaders. Utilising geometry shaders raises several other challenges which are not encountered in the

other implementations due to its reliance on manipulating geometry, as opposed to manipulating the frame buffer directly.

Because of the challenges, this approach spawned 3 different renderer implementations, each improving the visual fidelity of the effect with impacts to the performance of the effect as each was built. Comparing these further revealed that overall, using geometry shaders could produce a convincing visual effect at framerates above 100 frames-per-second (FPS), even on a laptop using an integrated GPU. When pushed on the desktop, the most basic of implementations could reach upwards of 1400 FPS, at the sacrifice of visual quality.

… Sections 2 – 4 – Background, Design, & Implementation …

# 5 – Evaluation

Several angles were taken when approaching the rendering problem, resulting in three distinct iterations of the renderer – the Naïve Renderer, the Multi-Pass Renderer, and the Layered-Compose renderer. With all these renderers placed into the latest version of the engine, comparisons were made to conclude which of the renderers is most usable in future projects. The two main factors lay in the accuracy of their effects, as well as the performance of each renderer in a variety of scenarios.

## 5.1 – Visual Accuracy

As previously mentioned during the implementation, each renderer was built to solve the issues of the previous iteration. The recurring issues boiled down to a sorting problem, where geometry which should only be visible from the inside of a specific portal, leaks out to recursion layers above.

### 5.1.1 – Naïve Renderer

Starting with the naïve renderer, there are glaring issues with the implementation pertaining to its handling of depth and colliding geometry. Specifically, the renderer does not make any attempt to discriminate between each level of recursion, resulting in an effect where extruded elements of an environment, such as corridors, pits, and recessions into the wall, cause visual disturbances.

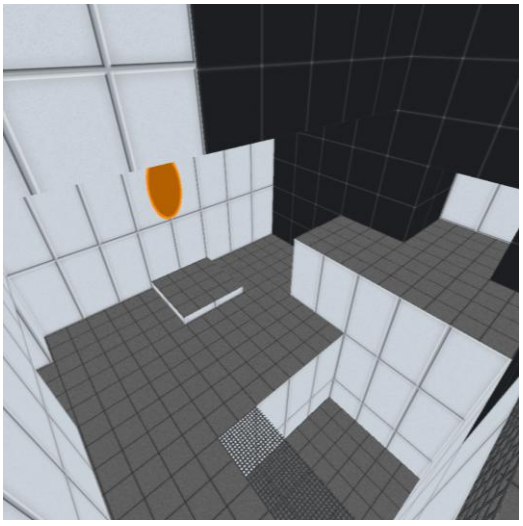

*Image 2: Naïve renderer prior to connecting portals.*

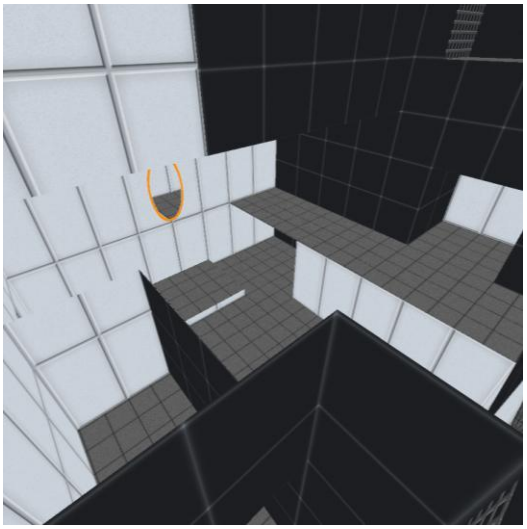

*Image 3: Naïve renderer with colliding geometry in the foreground & background.*

This is incredibly difficult to navigate as a user as this colliding geometry would not reflect the world's physics collisions, creating a very disorienting environment where the walls, floors, and ceilings cannot be trusted while a portal is connected. This issue only became worse as the portals were moved closer together in corridors, where several recursion iterations could occupy the same room, cluttering it further.

5.1.2 – Multi-Pass Renderer

The multi-pass renderer, on the other hand, effectively drew the geometry from a blank slate each time, from back to front, resulting in a significantly cleaner effect when observed from the room that the user is physically occupying.

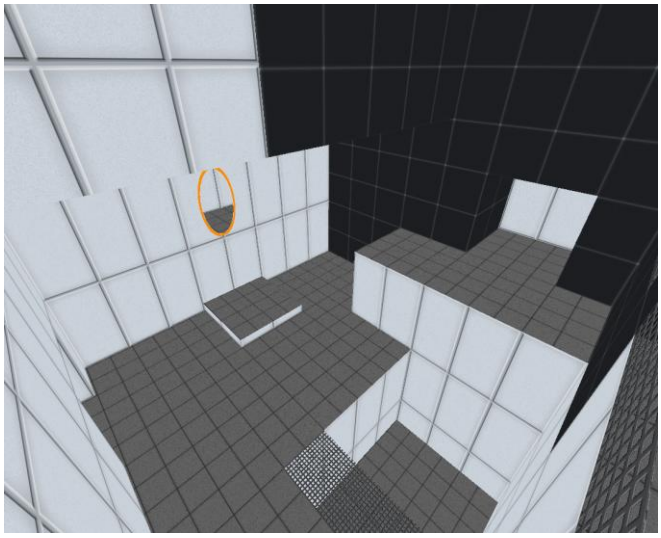

*Image 4: Connected Portals, using the Multi-Pass Renderer*

The approach isn't perfect however, and this becomes particularly apparent when looking through portals. When placing portals on topology similar to pillars, or in this case, corridors extruded from the main geometry, overdraw occurs. The layers are drawn furthest to nearest without any discrimination for if they lay within the bounds of a portal. This results in the geometry that's occupied by the user drawing over the inside of the portal, where the user would be able to see through the back of a wall as back sides are culled. Similarly to the naïve renderer, this issue will be present in any scene with a topology more complex than a simple box, making the issue easy to encounter in most environments.

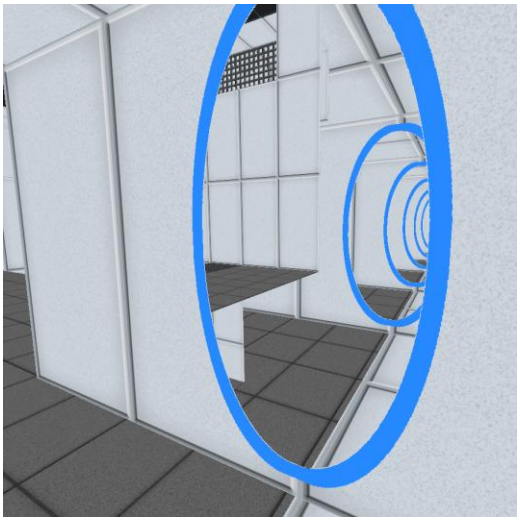

*Image 5: Multi-Pass Renderer with nearest geometry overdrawing issue.*

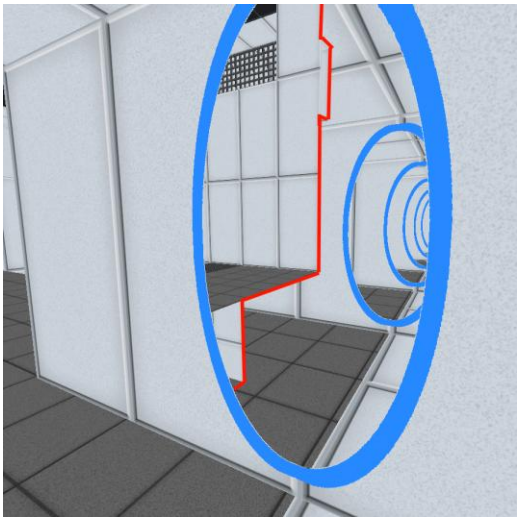

*Image 6: Multi-Pass Renderer with issue highlighted in red.*

In contrast to the naïve renderer, this effect is navigable despite the portals breaking immersion. There's potential for it to be used in environments designed to make this effect harder to reach, such as by preventing portal placements near corners, however, this would be a strict constraint to reason with. Ultimately, this could be improved.

5.1.3 – Layered-Compose Renderer

With the most believable effect, comes the Layered-Compose renderer, ensuring that the portal only ever shows the layer that is meant to be associated with it. Rendering the layers separately and sorting them afterwards allows the backsides of geometry to be properly overwritten, resulting in a working effect, both through portals and in the primary environment.

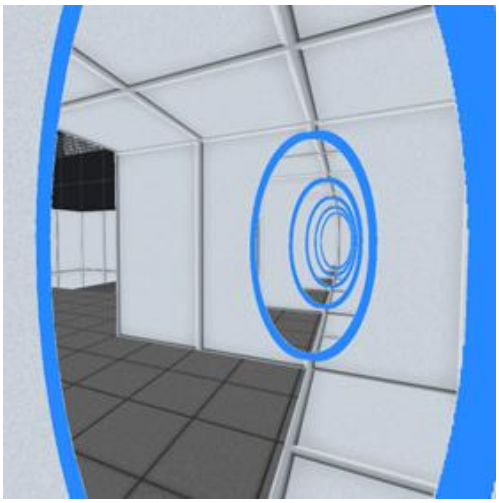

*Image 7: Layered-Compose Renderer with overdraw fixed.*

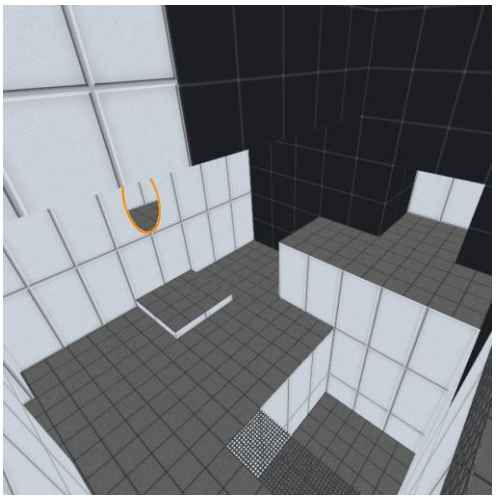

*Image 8: Layered-Compose Renderer continuing to work in original test*

.

This implementation isn't without issue though. It introduces shimmering in translucent textures due to poor management of transparency. This, in comparison to the previous issues, is minor and not immersion breaking, but is noticeable in areas with a lot of partially translucent textures and may raise issues in scenes with fully translucent textures.

## 5.2 – Performance

As with any fair experiment, the conditions for profiling each portal must be the same, within reasonable control. For example, the camera and portal positions for each of the tests across the renderers should be the same, because of which, a set of pre-made "Test Cases" were made to be tested across each hardware & renderer configuration. These can be found in Appendix C, with descriptions of their locations described within Appendix B

All the tests are grouped by device to keep comparisons clear, however, performance impacts from the operating system's behaviour, will be unmitigated. For the hardware, two different devices were selected to benchmark the project. The first of which is a PC Tower ("desktop") with a dedicated GPU, and the second of which is a laptop with an iGPU, being a CPU with an embedded GPU. Detailed specifications for each device can be found within the Appendix A. It should be noted that the graphics chips within both, while being from the same manufacturer, AMD, use different architectures, as such, may not be directly comparable.

Conducting the data collection – for each test case configuration of camera and portal positions, all frame intervals should be sampled & stored over a period of 5 seconds. From these recorded frame intervals, the minimum, maximum, and average frame intervals should be extracted and recorded. This should be done 3 times, each at least 10 seconds apart for each hardware & renderer configuration to reduce the risk of erroneous readings caused by dips in performance outside of the program's control. The average framerate can then be calculated by taking the reciprocal of the average frame time. Working with averages over a fixed period should mitigate outliers caused by framerate instability. Furthermore, logging the minimum & maximum frame intervals can provide an insight into how consistent the rendering performance of the engine is. This statistic is desirable as a program with consistent frame intervals, is usually perceived as smoother than one with unstable frame intervals. (Pastrana-Vidal et al. 2004)

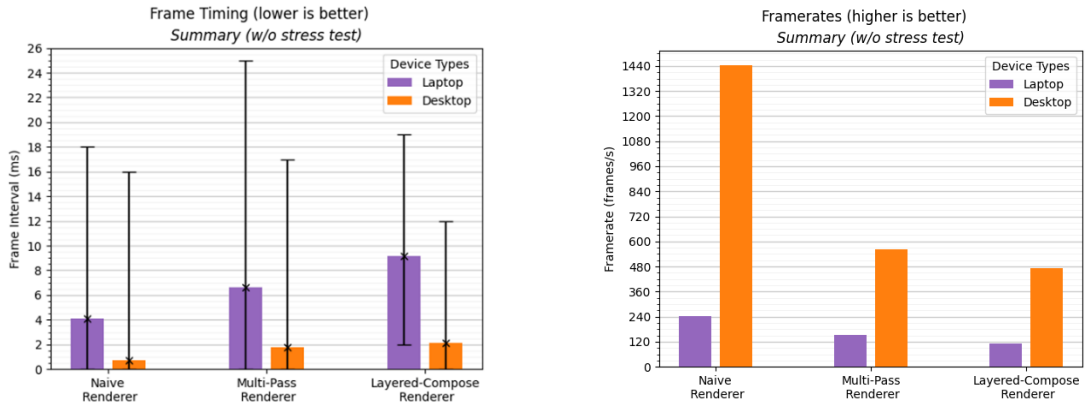


*Image 9: Chart plotting average frame timing, next to chart plotting average frame-rate.*
*Error bars represent minimum & maximum recorded frame intervals.*

Taking a summary of performance aggregated across every test case (minus Test Case #6 – the stress test), each iteration of the renderer got gradually less performant, as shown by the increase in the average time interval it took to generate each frame. The difference in average frame timing between each iteration of the renderer on the laptop, was significantly larger than the difference observed on the desktop. Regarding frame consistency, it's difficult to tell from the aggregate data, but it does appear that the multi-pass renderer fared worse than the other two, based on its significantly larger range on both devices.

The jump between the Naïve Renderer and the Multi-Pass Renderer was largely expected, as the Multi-Pass Renderer only added steps on top of the Naïve Renderer's logic. The performance of the Layered-Compose Renderer was less predictable prior to testing, as it relied on sub-passes which can behave differently to standard render passes. These were in theory better optimised on mobile hardware and tile renderers (Youssefi 2024), but with the hardware provided which is not classed as mobile, they showed limited benefits.

… Section 5, continued – Performance & Weighing Up …

# 6 – Conclusions & Future Work

In conclusion, the project circled back to its roots. The limitations of Narbacular Drop, Portal 1, and their other implementations were observed, several approaches were tried in an attempt to recreate the portal effect in a more efficient manner, and the final approach ended up using a similar technique to Narbacular Drop. Each of the techniques, however, all relied on the uncommonly used concept of geometry shaders, which proves that they have use-cases outside the usual use of adding levels of detail to geometry. The unusual result in the Layered-Compose renderer raises future questions yet provides some evidence that taking existing rendering concepts such as deferred rendering, and applying its principles to unconventional implementations such as portals, can result in an efficient, and visually complete solution, which otherwise would have been left unexplored. Echoing this, the project could still be taken further in both its evaluation and its implementation, as time limits were encountered that prevented both from being explored in more depth.

For the evaluation, extending profiling to understand exactly where the performance penalties came from would be ideal. During the evaluation process, tools such as Intel's Graphics Monitor & GPA Suite (Intel 2025) as well as AMD's GPUOpen Profiler (AMD 2025) were tried. Both tools failed to work with the Java implementation of Vulkan, so more time spent here would be ideal. Furthermore, the evaluation could be amended with a comparison between geometry shaders and the previously tried stencil-based approach. As that approach was a major component of the Portal series, comparing it to the impact of geometry shaders may place the results into context.

Iterating on the implementation, a few points were presented in the evaluation for extending both the Multi-Pass Renderer, and the Layered-Compose Renderer. The limits of each have not been fully explored, particularly in the Multi-Pass Renderer's case. During early development, the idea of using k-D trees, a spatial representation structure, to cull geometry through the portal was considered, however, the approach of using the Layered-Compose renderer took priority as it was more feasible to complete in the time-frame. Investigating this path could yield a consistently quicker renderer with a usable visual effect. Taking the Layered-Compose Renderer further would be a little more difficult as

Finally, extending the engine with physics is necessary for a game – portal physics have proven tricky and has spawned a few pieces of research such as Hossenfelder's video essay on explaining reality-accurate portal physics (*The Physics of Portals (Made With Love)* 2024), and Valve's approaches to creating physics which are fun to play with, as described in their Developer Commentary (Kircher et al. 2007). No matter the model, figuring out how objects interact with the environment while passing through the portal is a large problem, with many different edge cases, thus it would be a good direction to investigate.

# References

AMD. 2025. *Radeon™ GPU Profiler*. Available at: https://gpuopen.com/rgp/ [Accessed: 15 May 2025].

Intel. 2025. *Intel® Graphics Performance Analyzers*. Available at: https://www.intel.com/content/www/us/en/developer/tools/graphics-performance-analyzers/overview.html [Accessed: 15 May 2025].

Kircher, D. et al. 2007. *Portal developer commentary (Transcript)*. Available at: https://theportalwiki.com/wiki/Portal_developer_commentary [Accessed: 1 February 2025].

Pastrana-Vidal, R.R., Gicquel, J.C., Colomes, C. and Cherifi, H. 2004. Sporadic frame dropping impact on quality perception. In: *Human Vision and Electronic Imaging IX*. SPIE, pp. 182–193. Available at: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/5292/0000/Sporadic-frame-dropping-impact-on-quality-perception/10.1117/12.525746.full [Accessed: 7 May 2025].

*The Physics of Portals (Made With Love)*. 2024. Available at: https://www.youtube.com/watch?v=cox7481IE6o [Accessed: 22 March 2025].

Youssefi, S. 2024. *Efficient Render Passes — On Tile-Based Rendering Hardware*. Available at: https://medium.com/androiddevelopers/efficient-render-passes-on-tile-based-rendering-hardware-621070158e40 [Accessed: 2 May 2025].