Rendering Portals in Virtual Reality

Milan van Zanten
University of Basel
milan.vanzanten@unibas.ch

November 1, 2022

1 Introduction

There are several uses for portals in computer graphics including determining the visibility of parts of a 3D scene[6], dividing a scene up into separate areas, rendering a mirror surface, or as traversable portals that can be seen and moved through. These uses can be broadly categorised as either an optimisation technique or a rendering trick. The former two of the mentioned applications are used to determine geometry that can be ignored when rendering a scene and speed up the process. Mirror surfaces and traversable portals though are effects purposefully implemented in an application to benefit the experience.

There are already many implementations of traversable portals in media like video games or architectural visualisations[2]. In this paper we will focus on an application of traversable portals concerning the space limitations in a virtual reality (VR) experience.

One of the main challenges when implementing VR applications is immersion, since errors in tracking and latency are noticed particularly strong[1]. In an effort to maximise immersion, most of VR has moved from seated experiences with movement limited to three degrees of freedom (just rotation) to "room-scale" tracking. Here, in addition to the rotation of the VR headset, the user's position is tracked either via external devices with fixed positions or by cameras that analyse the surroundings and use computer vision algorithms to determine the position. With the added positional tracking, the six degrees of freedom allow the user to move around the room freely. Thus, the only limitation now is the available space. To move around virtual worlds larger than the available space, several different methods have been developed[3]. Examples include head-directed locomotion, point & teleport and more. Indisputably though, the technique that preserves immersion the most is actual walking inside the real space.

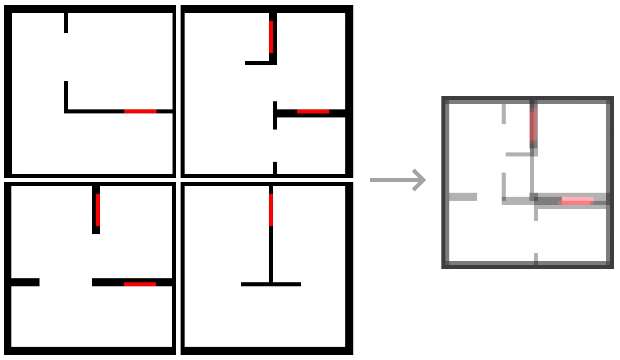
Figure 1: Four rooms with portals (in red) can all be accessed without leaving the smaller real space[5].

A recent method to circumvent the space limitations of walking inside a real space is the concept of impossible spaces. Overlapping rooms are connected through portals into a single space many times larger than the initial rooms themselves. If such an arrangement is made while factoring in the real available space, the whole virtual space can be accessed by passing through the portals. An example layout can be seen in Figure 1.

To allow for such impossible spaces to exist, the aforementioned portals are necessary. When viewed, they show what the user would be seeing through the portal in the other room, and if a user crosses the plane of a portal, they are transported to the connecting one. In virtual reality, implementing such a portal system poses some extra challenges.

The goal of this paper is to explore how classical portals from monoscopic applications can be implemented in stereoscopic VR. We will focus on two main objectives:

1. How can portals be implemented in a way that the user can look and pass through them without noticing?
2. What performance pitfalls should be considered when rendering portals in VR and what optimisations can be applied to alleviate them?

2 Unobtrusive Transitions

Portals in computer graphics are generally just flat quads onto which the view from within another portal is drawn. In monoscopic applications, a single camera moving from space A through a portal into space B will see a flat image of what is on the other side of it as long as it is still in space A. As soon as it crosses the portal plane, it is being transported to space B and will from that point on render that space directly.

The way a VR scene is rendered though raises the question of how to handle the teleportation of the two separate cameras rendering each eye. For example, consider the centre point between the eyes that could be used to decide when the portal plane has been crossed. If the user views the portal from an angle they could clip through the portal with one eye when they enter it before being transported, as shown in Figure 2. This results in a short flicker when the user passes through a portal at best and a completely wrong view on one eye if the user stops inside the portal.

One way to solve this problem could be transporting each eye separately whenever it passes through the portal, but that idea conflicts with one of the performance optimisations we will discuss in the next section.

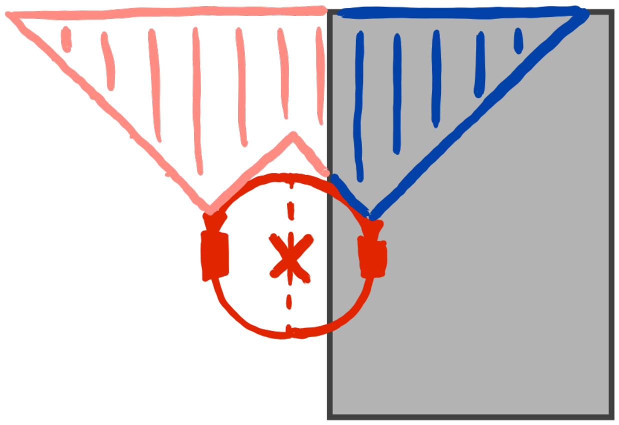
Figure 2: What should the right-eye camera render if it is inside the portal wall (in grey), but the centre of the head (in red) has not crossed the portal plane? If nothing is done, the blue part of the user's field of view would not render the next room, but whatever is inside or behind the portal wall.

If we want to keep the optimisations we introduce, a different method of making sure the transition through a portal is unnoticeable by the user, no matter how slowly they move or where they look. This method takes advantage of back-face culling[1].

Instead of the portal being a plane, we will have it be a box. Every surface of the box is shaded with the view from the other room. If one eye now clips inside the portal box, it will still see the other sides of the box that show the next room, instead of what was behind the portal originally. The last problem now is that if this eye now wants to look back out from the portal box into the old room, it should not see the portal plane. This is where we use back-face culling to make sure the front side of the portal box is only visible from the outside. A visualisation of what each eye now sees in this scenario can be found in Figure 3.

[1]A triangle in computer graphics is generally considered to have a front side and a backside, determined by whether the points of the triangle appear in clockwise or counter-clockwise order from where they are viewed. Back-face culling reduces the amount of drawn triangles by only drawing the triangles that are facing the camera.

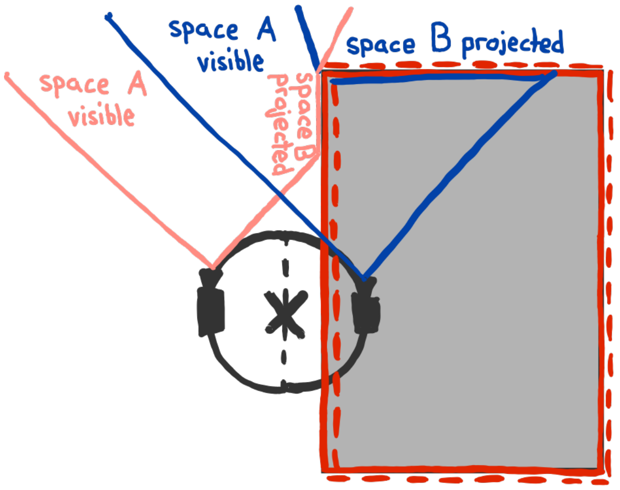

Figure 3: The sides of the portal box are only visible from the side with a solid red line. Therefore, the right eye can see space B on the inner wall and at the same look out from the portal box back into space A.

3 Optimisation Considerations

Each portal requires rendering an additional viewpoint. When rendering the portals in VR, where each eye is rendered by its own camera, there are now two additional viewpoints to render from. In the naive case where each viewpoint is rendered the same, we produce quadruple the amount of work compared to a basic non-VR scene without portals. We will present two optimisations that can reduce the rendering time and analyse their impact.

The first optimisation — using the stencil buffer to only render what is seen through the portal — is concerned with improving the rendering time of portals in general.

In contrast, the second optimisation improves the overall performance of rendering VR by not pushing the whole scene to the GPU twice and rendering a texture for each eye, but rather rendering both eyes onto a single texture while only keeping a single copy of each rendered object in memory.

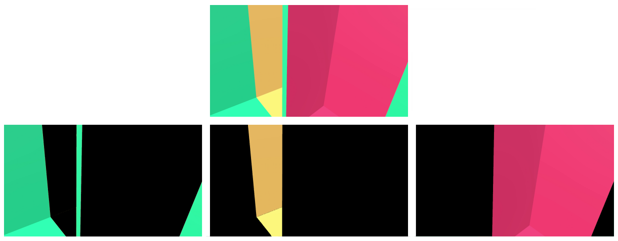

Figure 4: The user is standing in a green room looking at two portals to a yellow and a magenta room. The amount of rendering can in this case be divided by three if the pairs of cameras only render a masked out area of their room instead of the whole picture.

3.1 Stencil Buffer

The stencil buffer on graphics cards is used to mask out an area on the screen that will not be drawn to. If the portals are first rendered with a shader that simply marks the stencil buffer and nothing else, in a second pass, the virtual cameras behind those portals can then fill the remaining area with the view from the other side of the portals. This way, no redundant pixels are drawn and the amount of pixels that need to be drawn stays practically constant no matter how many portals are visible on the screen. An example demonstrating how a scene with two portals is split up by a stencil buffer can be found in Figure 4.

3.2 Single-Pass Instanced Rendering

The naïve way of rendering a stereoscopic image is to simply render it two times, once from the perspective of each eye. This is called multi-pass rendering and is illustrated in Figure 5. The advantage of this technique is its simplicity in implementing it. Shader code from monoscopic applications can be used in multi-pass rendering without any changes since we are essentially just rendering two images per frame.

This method comes with a large performance downside though. When rendering portals, the amount of work might in the worst case be multiplied by the number of portals. Thus, the rendering overhead caused by multi-pass rendering is in this case not acceptable.

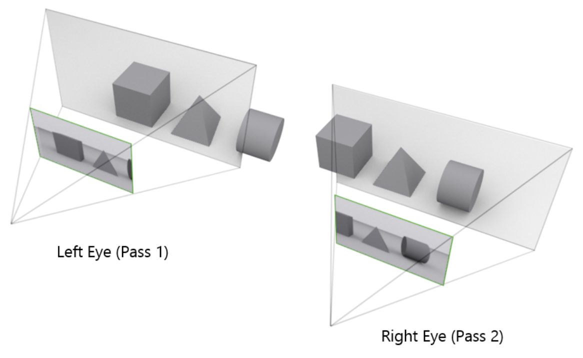

Figure 5: When rendering in multiple passes, all work is doubled. This illustration shows how basically the three objects are treated as though they were separate objects[4].

The second method described in Figure 6 — single-pass instanced rendering — improves upon this by rendering both the left and right eyes onto the same texture. Additionally, the geometry only exists on the GPU once, as well as any shadow maps and other view-independent data. The objects are rendered twice as different instances from the two perspectives of the eyes.

Single-pass instanced rendering should greatly improve the performance, but as mentioned in Section 2 rendering both eyes in one pass eliminates the possibility of transporting them through the portal separately from each other.

4 Performance Comparison

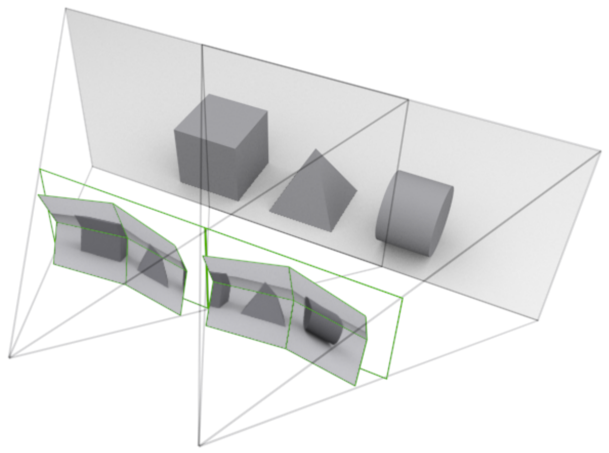

Figure 6: When rendering in an instanced single pass, the whole geometry of the scene is processed only once, which significantly reduces the amount of work[4].

References

[1] Michael Abrash. Why virtual reality is hard (and where it might be going). *Game Developers Conference 2013*, 2013.

[2] Daniel G Aliaga and Anselmo A Lastra. Architectural walkthroughs using portal textures. In *Proceedings. Visualization '97 (Cat. No. 97CB36155)*, pages 355–362. IEEE, 1997.

[3] Costas Boletsis. The new era of virtual reality locomotion: A systematic literature review of techniques and a proposed typology. *Multimodal Technologies and Interaction*, 1(4), 2017.

[4] NVIDIA Corporation. Vrworks – single pass stereo. `https://developer.nvidia.com/vrworks/graphics/singlepassstereo`. [accessed 1.11.2022].

[5] Daniel Lochner. Vr natural walking in impossible spaces. *Motion, Interaction and Games (MIG '21)*, 2021.

[6] Nick Lowe and Amitava Datta. A new technique for rendering complex portals. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):81–90, 2005.