1.1.1 Real-time Content Capture Redesign

As originally outlined in Figure 6, generating 360º within a games engine is not natively supported for the style of rendering and playback the Soluis Portal requires. To overcome this initial content implementations within the Unreal and Unity games engines use specially constructed camera systems to capture the content. It is not within the scope of this research to fully explore the current methods for 360º rendering but an understanding is required. Using research from Paul Bourke (Bourke, 2009) the inherited Soluis system rendered six individual camera faces and passed them out of the games engine into Touchdesigner to deal with the pixel and vertex processing (Lindholm & Nickolls, 2009), creating either an equirectangular (360º image) or fulldome fisheye depending on requirement. Figure 15 shows an extract from the whole content pipeline detailing the process.

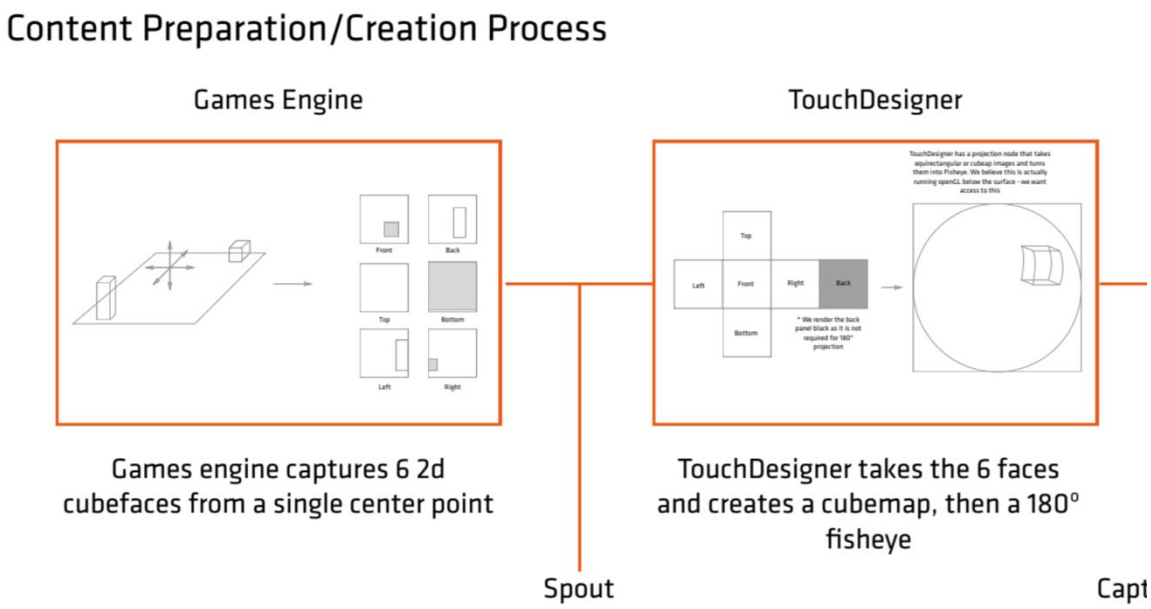

Figure 1: Detailed overview of real-time capture system

Understanding of this process is important for the impact of the changes made during this research, and the user experience improvements. The games engine outputs six individual 90º field of view 2d cameras (usually forward, back, left, right, up and down). These six images are then gathered into Touchdesigner and stitched together into the required output format.

Due to Touchdesigner and Unreal Engine not having a synced rendering system, and the manner in which spout texture sharing works (eg a constantly active link between an image) a situation would arise where a number of the camera faces had rendered the next frame, but the remainder had not. This left a ghosting or lagging effect on the Portal playback, an obvious technical error to users and a break in presence.

The obvious solution was to reduce the amount of information passed between each program to a single image, this way as long as Touchdesigner could process the data fast than 16ms per frame then each rendered frame on the Portal would be shown as delivered by Unreal Engine.
This would also allow for the leveraging of the more powerful GPU utilisation within the games engines. The development would later become known as the 'Portal Plugin' internally, over the time with this research it has taken two forms; The first a stepping stone using the methods outlined above but internally compressed into the fisheye format. The second using direct pixel and vertex shader implementation for the most efficient direct method of processing the data.

The first iteration took the six internally captured cameras and warped them onto a special sphere mesh hidden inside the Unreal Game Scene, a single orthographic camera then captured the result from within the origin of the sphere pointing outwards, towards the desired direction.
Figure 16 gives a profile shot of the inside of the sphere in concept and Unreal. This method of capture was neither an improvement nor a sustainable rendering method due to a number of failings in the execution but served as a proof of concept required to create the complete shader implementation in iteration two. The system suffered as it was now having to internally capture six direction cameras, map them into a cubemap and onto a sphere, then recapture that sphere at incredibly high resolution – all whilst rendering the scene. With some optimising it may have been a feasible approach, however its overall improvement to system performance where never fully measured.
The second issue was the method of capture, using an orthographic camera from the internal sphere rendered something known as a hemispherical (or linear) fisheye image. For the Portal and its image to be correctly orientated and represented we needed to utilise something called an angular fisheye, a method for equality spreading the resolution of an image across the fisheyes width

(Bourke, 2001). This did not happen in the internal camera build as it compressed the angle of view as it approached the edge of the half-sphere.

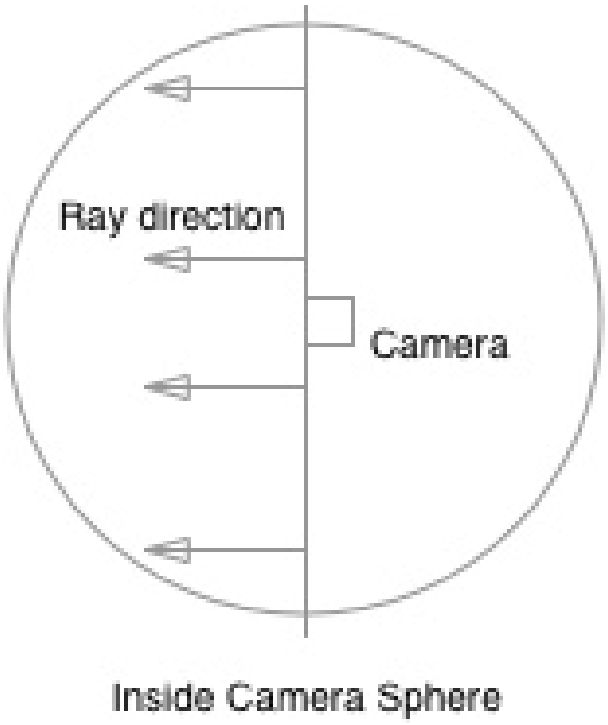

Figure 2: Inside initial plugin capture sphere

The second iteration of the Portal Plugin set to put the theory of the first into a more sustainable, efficient method. Using the same research from Paul Bourke (2001) we were able to implement a completely shader-based version of the calculations. Still using the same six camera operators within the play environment (accurately capturing beyond 90º FOV is an issue of current generation games engines) a cubemap was rendered, equirectangular made and then warped using code rather than physical representation within the game scene. Appendix 11.2 has an early example of the complete pixel shader code as created for the Portal Plugin.

With the implementation complete, this research was able to move to a single point of connection between the games engine and rending pipeline in Touchdesigner. Meaning that no matter how variable the framerate between the two, the playback would not show tears, stutters or lag based on the cubemaps not rendering in sync. Measuring the technical impact of the change is more complex than listing the numbers as above in the technical streamlining section due to the

complexities of multiple rendering system working in tandem. However, with the new camera system both Unreal Engine and Touchdesigner were able to fully maintain a lower than 17ms rendering time across all actions per frame. The means that both Touchdesigner and Unreal could generate content at up to 60fps.

The real power of the Portal Plugin and its new rendering method came in the form of being able to push the boundaries of playback possibilities and the ability to evaluate the effect on quality on a user's experience.

2 Bibliography

Bourke, P. (2001). Computer Generated Angular Fisheye Projection. Retrieved May 29, 2018, from http://paulbourke.net/dome/fisheye/

Bourke, P. (2009). iDome: Immerisve Gaming with the Unity game engine.

Lindholm, E., & Nickolls, J. (2009). NVIDIA TESLA: A UNIFIED GRAPHICS AND COMPUTING ARCHITECTURE TO ENABLE FLEXIBLE, PROGRAMMABLE GRAPHICS AND HIGH-PERFORMANCE COMPUTING. Retrieved from https://fenix.tecnico.ulisboa.pt/downloadFile/3779576765088/IEEEMicro_TESLA.pdf