Sheffield Hallam University
Faculty of Arts, Computing, Engineering and Sciences (ACES)
Department of Computing

MComp Individual Project Report

Depth-Based Checkerboarding

By *Eve Johnson Powell*

30/01/2024

Supervised by: *Dr Mark Featherstone*

This project report does NOT contain confidential material and thus can be made available to staff and students via the library.

A report submitted in partial fulfilment of the requirements of Sheffield Hallam University for the degree of Master of Computing in *MComp Computer Science for Games.*

Table of Contents

Table of Figures

1 Abstract

"Checkerboard" rendering techniques aim to accelerate 3D rendering by avoiding the shading of regularly spaced regions of the screen. Two main approaches exist; the first, interlaced rendering, uses a reduced size render target which is 'expanded' to expose sampling gaps that are then filled in, with respect to adjacent pixels and graphical information history. The stencil-based approach uses the depth-stencil buffer to reject pixel shading in regularly spaced 2x2 blocks (Wihlidal, 2020). Interlaced approaches have been used in successful commercial projects, but the stencil-based method has been overlooked, or even challenged such as by Frostbite Labs.

This paper tests the claim that the stencil-based approach produces inferior results visually, and seeks to provide numerical evidence of the computation speed advantages of using stencil-based checkerboarding versus a standard rendering approach.

We find evidence to indicate a performance benefit from using the stencil-based approach, but find the impact minimal in comparison to other techniques. Quality testing with human participants also indicates a substantial downgrade in quality compared to standard rendering techniques, contrary to the results teams have found with the interlaced approach.

Several improvements are identified for future work, both for the algorithms and for the testing environment.

2 Introduction

Checkerboard rendering techniques aim to reduce the cost of rendering scenes by logically reducing the size of the back buffer (Carpentier & Ishiyama, 2017). In this case of interlaced checkerboard rendering, this is a literal reduction in size, using a smaller render target that is later logically expanded to create gaps that are filled in. For stencil-based checkerboarding, this is done by blocking out areas with depths of zero, making half the pixels functionally non-existent with regards to pixel shading.

Interlaced checkerboard rendering has proved a commercial success; Ubisoft's Rainbow Six Siege engine runs on it (Mansouri, 2016), with results of high enough quality that there were no complaints at its implementation. Frostbite Labs also use it on Battlefield 1 (Wihlidal, 2020); Guerilla Games use it for Horizon: Zero Dawn. Each is slightly different to the other, with slightly different constant values and some varied techniques included, but with similar core methods. One of the key advantages of this approach is its densely-packed nature; for any given unshaded pixel, there is potentially useful information in the four adjacent pixels.

Stencil-based approaches are underrepresented in literature, and this paper aims to address this underrepresentation, by testing stencil-based checkerboarding in comparison to a standard rendered scene. Approaches will be compared based on execution time, memory consumption, and quality of the resultant video, as judged by human participants. This can then be compared to the interlaced techniques as presented by Guerilla Games, Ubisoft, and Frostbite Labs.

Section 3 (Literature Review) explores an early history of graphics hardware, before continuing to discuss modern advancements. The software approaches enabled by modern hardware are then explored, with a focus on techniques that aim to reduce frame time, rather than those that aim to increase the visual quality of a scene.

Section 4 (Method and Methodology) then outlines the approach that will be taken in terms of testing. It outlines some metrics that will be considered when testing the resultant applications, and justifies human testing for quality as opposed to image analysis algorithms.

Section 5 (Design and Development) narrativizes the development of the product, with reference to specific visual problems, and the algorithms used to address them. This chapter makes close references to the interlaced techniques found in literature, with modifications to apply their techniques bidirectionally.

Section 6 (Testing) then explores the results of human quality testing, and the metrics obtained from profiling both techniques.

We find that the stencil-based checkerboard approach has similar efficiency to interlaced techniques, but testing suggests that the images produced are of an inferior quality to those produced by interlaced techniques. Further work is suggested, including more complicated and refined test environments and some potential algorithmic advancements. Results are all

inconclusive, but suggest that Frostbite Labs assessment is correct, that sampling information is too sparse for a 2x2 reconstructive block (Wihlidal, 2020), and that single pixel reconstruction techniques as used in interlaced checkerboarding are preferable.

3 Literature Review

3.1 Abstract

Many applications require graphical output, from 3D modelling software to cutting-edge games, to visual effects software for use in film and television. Depending on the use, this comes with different challenges; for a film effect, a completely photorealistic final appearance may be desirable, even if it takes minutes or hours to process the resulting image (Rath, 2009). For 3D modelling software, the primary concern may be the efficient updating of vast meshes, while still maintaining a reasonable framerate (Lippens, Nagasamy, & Wolf, 1995). Games share a similar concern to 3D modelling software, which is maintaining the framerate, generally from 30 to 60 frames per second. This gives only 16.7ms to render a full image before players can report a noticeable difference, and in the context of huge scenes, this can be difficult to achieve (Claypool & Claypool, 2009). Games still need to look visually pleasing, as there is a commercial incentive to aim for graphical quality, as demonstrated in Figure 1, but with limited frame time, developers look for ways to accelerate the process of rendering. As such, techniques that reduce the rendering time of a scene, with no or minimal loss of quality are desirable.

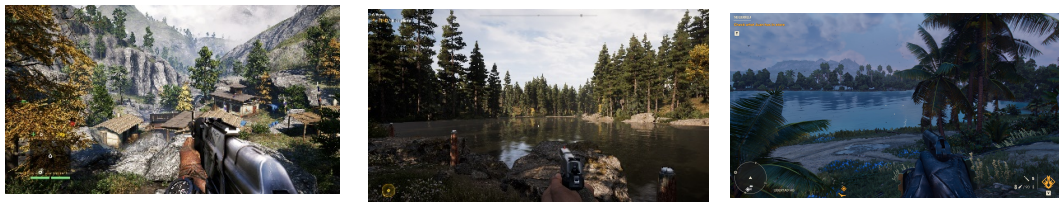

Figure 1 - The increase in photorealism in games as demonstrated through the Farcry series. From left to right, Farcry 4 (2014), Farcry 5 (2018), Farcry 6 (2021). All games rendered at highest available quality. (Farcry 6 missing the optional HD texture pack).

The commercial and academic importance of rendering acceleration can be gauged from the extensive research into it and is driven by demands for more computationally expensive effects, and for lower battery consumption on mobile platforms (Choi, Park, & Cha, 2019). AAA video games continue to push the limits of graphical fidelity, with many games aiming for photorealism, which can require expensive operations, such as path-traced lighting. To meet these ambitious standards, while maintaining frame time targets, software and hardware innovations are needed.

3.2 The History of Graphics Hardware

Early Developments

At its core, rendering is the process of converting a set of vertices, lights, and colour textures into a final image, representing either a photorealistic or stylised representation of the scene. In his history of the field, Machover describes the development of graphics capable machines as early as the 1950s, such as in the US SAGE air defence system; by the early sixties,

MIT had an ‘interactive graphics console’ for use outside of military situations. This was followed by MIT’s Sketchpad, providing the theoretical foundations for a lot of future work. However, at this time, graphics enabled machines were mostly used in commercial settings, with graphics labelled “a solution for no known disease” (Machover, 1978). From the late sixties, it appeared that that disease was a lack of fun; in 1968 MIT demonstrated “Spacewar!” (Figure 2, and by 1976, the first dedicated games chip had been created (Salomon, 2011).

Figure 2 – Joi Ito from Inbamura, Japan - Spacewar running on PDP-1, CC BY 2.0 *(Ito, 2007)*.

With the emergence of home gaming consoles, the rise in ownership of personal computers, and the discovery of foundational techniques such as rasterization and the concept of a pixel, games were responsible for a large share of the computing market by the late eighties. By the end of the decade, Atari’s “I, Robot” had pioneered 3D graphics in games, and Pixar had introduced the concept of a shader, a dedicated small CPU-bound function to calculate position, lighting, and shadowing in a rendered scene. Graphics calculations were still largely CPU bound (Peddie J. , 2023).

Discrete Processors and the Graphics Pipeline

The first dedicated graphics processor appeared in 1993 for arcade systems, although NVIDIA’s 1999 processors were the first to be commercially available for use in PCs. Where a CPU would execute a single instruction at once (although with pipelining, multiple instructions may be at distinct stages of execution) (Gronowski, Bowhill, Preston, Gowan, & Allmon, 1998), a GPU provides a set of smaller processing units, each capable of executing in lockstep with other units in its group. This is ideal for calculations that need to be executed for each member of a set, such as per-vertex or per-pixel calculations (Peddie J. , 2023). Where previously CPUs could be bottlenecked by running shader programs in sequence, there was now a method for running them on the full set in parallel, a clearly preferable option. However, with a discrete processor comes separated memory; the CPU still needs to upload data to the GPU for it to process, which takes time to manage and is limited by the bandwidth of the bus between the two processors (Wang, Huang, Chen, & Zhang, 2008).

Early graphics chips featured a set pipeline for rendering primitives, optimised for vertex transformations. These early chips featured limited programmability, acting primarily as extensions of existing transform-acceleration hardware, but with additional features such as

alpha blending and triangle culling that remain core features of GPUs today (Peddie D. J., 2021). With the next generation of chips came programmable discrete vertex shader and pixel shader stages, offering increasing control to developers (Han, 2011) (Microsoft, 2022). To paraphrase Han, GPU evolution is the story of fixed-function units being replaced with programmable stages. Later generations of chip introduced the discrete geometry shader stage, which allows the creation of new primitives dynamically, enabled by the emergence of a "unified architecture" for shaders, casting aside distinct execution units per-stage to allow all shader stages fast access to resources such as textures (Patidar, Bhattacharjee, Singh, & Narayanan, 2006).

The Compute Pipeline

At around the same time, the compute shader was introduced, for General Purpose GPU functionality, (Huang, Huang, Werstein, & Purvis, 2008). Where the existing rendering is rigid despite its programmability, affording useful features such as implicit rasterization and depth testing, GPGPU programming forgoes the pipeline for massively parallel arbitrary computation. This means compute operations miss the optimised hardware support for these implicit features (Laine & Karras, 2011), but similarly are not constrained by them, gaining flexibility at the expense of performance drawbacks in some operations. It is worth noting that in recent years some research has produced faster alternatives to the traditional rendering pipeline based around compute shaders, such as Unreal Engine 5s Nanite system, which uses hardware rasterization only when it is predicted to be faster than a software method (Karis, Stubbe, & Wihlidal, 2021). NVIDIA's compute implementation, CUDA, was designed to provide a C-style interface for GPGPU programming, becoming commercially popular in the late 2000s, and provided the first graphics processors capable of arbitrary computation, without having to follow the limitations of the graphics pipeline (Sanders & Kandrot, 2011).

Compute shaders have numerous uses across both graphics and non-graphics fields. For example, in graphics, compute shaders are used in the Forward+ pipeline, which shades a scene with a depth pre-pass, followed by a compute-based light culling step, after which final shading occurs. The light culling step divides the render target into "tile" regions, for which a list of the lights which reach the tile is calculated. This enables the pixel shader stage in the second pass to consider only the lights which are directly relevant (Harada, McKee, & Yang, 2012). Outside of graphics, compute shaders see use in physics, such as fluid simulation; each particle exhibits similar behaviour, making it an appropriate candidate for GPGPU, and keeping the particles in GPU memory reduces the streaming overhead of transferring data from the CPU to the GPU if simulation were done on the CPU (Gunadi & Yugopuspito, 2018). Research has also been conducted into using GPGPU functionality for AI in games; compute-accelerated SQL queries

could be used in place of finite-state-machine models of AI to offload often expensive CPU calculations to the GPU (Blewitt, Ushaw, & Morgan, 2013).

Hardware innovations continue to be introduced; in 2018 NVIDIA revealed their Turing architecture, which introduced hardware support for some techniques that had been created in software already. As well as advances to accelerate existing shaders, new cores for common deep learning operations (more proof that the GPU has moved beyond being a device solely for rendering), and ray tracing acceleration, the architecture introduced Variable Rate Shading (VRS). Taking cues from VR-specific techniques Multi-Resolution Shading and Lens-Matched Shading, VRS allows the program to alter the size of the region shaded by a single shader. Under normal circumstances, one pixel shader invocation corresponds to one pixel of the render target; with VRS this could be extended to as many as 16 pixels, in a 4x4 square. The screen is divided into 16x16 pixel tiles, and the shading rate set per tile at runtime, allowing the program to render some areas at lower detail based on real-time predictions (NVIDIA, 2018). Support for this has been implemented into DirectX 12, although not all features will be available on every device (Microsoft, 2023).

3.3 Software Techniques

Variable Rate Shading in Software

As alluded to, this was a hardware formalisation of techniques that had existed in software previously. In VRS terms, the group of pixels shaded by a single pixel shader invocation, be it a 1x2 region or a full 4x4 region, is referred to as a "coarse pixel" (NVIDIA, 2018). In its modern form, the concept of a "coarse pixel" was introduced by Vaidyanathan et al. in their Coarse Pixel Shading (CPS) technique. This was in response to the large power costs of pixel shaders in graphics computation (Pool, 2012), with the aim of reducing the power consumption without compromising image quality. CPS divides the screen into square tiles of size TxT and allow the coarse pixel size to be set per-tile, where the size is Nx x Ny; coarse pixels are put into 2x2 groups called coarse quads. When a primitive is drawn, it is rasterized, and its covered fragments stored per-tile. The tile is then divided into coarse quads, and the coarse quad shaded. Coarse pixel size can be set as an output of any shader before the pixel shader, allowing a huge amount of control over the desired effect. They note potential applications in a multitude of areas (Vaidyanathan, et al., 2014).

Foveated rendering is noted as a technique that would be better served by a coarse pixel approach; foveated rendering is a technique designed to exploit the natural deficiencies in the human eye to reduce shading workload without the risk of losing visual fidelity. Commonly used in VR rendering, which requires offset stereo displays, one per eye, it aims to offset the

huge time cost of stereo rendering by rendering at a lower level of detail away from the point at which the eye is focussed. Previous research at Microsoft shows that there is a benefit to using an eye-tracking camera with desktop PCs; around the visual focal point, a rectangular region of the scene is drawn at high quality, and around its other rectangular regions are drawn at increasingly lower quality, creating lower quality images the further away from the visual focal point a pixel is. These 'eccentricity layers' are then blended to create the final image (Guenter, Finch, Drucker, Tan, & Snyder, 2012). As shown by Vaidyanathan et al., this can also be achieved with coarse pixel shading, using a radial function. Nearer the visual focal point, the image is rendered at a per-pixel rate, and further away at a 2x2 rate. They noted that sharp edges were preserved within the final image even at the 2x2 rate (Vaidyanathan, et al., 2014).

Motion-adaptive shading (MAS) (Yang, et al., 2019) is a technique that can be accelerated with VRS. Based on similar principles to foveated rendering, it exploits an existing loss in visual quality to save time; in this case the effects of engine-simulated motion blur, and the natural tearing caused by screens (NVIDIA, 2019). In areas that would be blurred already, the shading rate can be lowered, as it will be less noticeable to the end user; this is because motion naturally obscures details, usually the only downside of VRS. Many modern graphics applications use motion blur already, simulating the effect of a film camera, with the blur caused by the shutter speed of the camera, so this is a relatively non-invasive acceleration technique (Epic Games, 2023) (Unity Technologies, 2023). NVIDIA implement their MAS by taking motion into consideration when calculating an 'estimated error' value, scaling down the result based on the proposed shading rate. For example, a fragment that is moving at a rate of six pixels per frame will have its naturally estimated error halved, meaning it is more likely to be considered for a lower shading rate. A coarse shade only occurs if the estimated error falls below a dynamically calculated threshold, to avoid visually noticeable errors (NVIDIA, 2019).

Multi-rate shading (MRS) also uses the notion of a 'coarse shade' for acceleration of rendering; it splits rendering into two logical steps, the coarse phase and fine phase. Primitives are rasterized into 'coarse pixels' where every constituent is covered, and 'fine pixels,' pixels for which the coarse pixel to which they would belong is not fully covered. Coarse pixels go through a dedicated shader, which generates an output for every constituent pixel; it can also set flags, with the example of a 'specular refinement' flag. The constituent pixels then go into the dedicated fine pixel shader and undergo relevant adjustments; it is at this stage that the set flags are considered, to account for differences that cannot be easily found from slight adjustments. This allows the 'heavy' work to be done at low resolution (2x2 or 4x4) and the fine tuning done at larger resolution. This is an extension of multi-pass, mixed resolution rendering, which creates a ¼ sized render target (1/2 in both dimensions) for coarse data and then samples it while running the second pass. The advantage of MRS is requiring only one pass and allowing

adaptivity of the coarse pixel size; however, this required dedicated compiler support to implement (He, Gu, & Fatahalian, 2014). Vaidyanathan et al. also extended this technique to use VRS; using a 2x2 rate for computing ambient occlusion, but a per pixel rate otherwise (Vaidyanathan, et al., 2014).

Rendering Pipelines

Forward rendering pipelines are the most 'basic' form of lighting pipeline, in which geometry is lit as it is submitted. One object is submitted, its vertices are transformed and then all visible pixels are shaded by all lights, with the results written to a colour buffer and depth buffer (Mommersteeg, 2015). Subsequent objects are fully shaded if they pass the depth test, potentially obscuring pixels that had previously been fully lit. This is known as 'overdraw,' where time is wasted calculating shading information that is later deemed irrelevant. Forward rendering must also account for all lights in a scene, as there is no guarantee that obscuring objects would later render. Forward rendering results are generally considered less visually impressive, but the pipeline is still useful; until the early 2000s it was considered a state-of-the-art pipeline, and innovations such as the Forward+ method make it a more relevant technique.

Deferred rendering provides an alternative pipeline model; where forward rendering would calculate a colour and depth for its objects, deferred rendering adds a screen space normal buffer. Lighting is not calculated until after all geometry has been drawn, when it is done as a post-processing step (Klint, 2008). This cuts down the time wasted due to overdraw, as no complex lighting is calculated for geometry that is later obscured. This also cuts down on calculation complexity; a forward renderers complexity can be expressed as "the number of models * the number of lights," whereas in deferred lighting, it can be expressed as "the number of models + the number of lights." This can be extended to further cut down complexity, such as a scissor test around the area of influence of local lights, meaning they will act only on the locally relevant pixels (Thaler, 2011).

Checkerboarding

Several techniques aim to reduce the cost of shading by dividing the screen into regions, some fully shaded, and others reconstructed from various scene information.

Guerilla Games present a checkerboarding technique used in their Decima Engine and Horizon: Zero Dawn; a standard checkerboarding approach would render only half of all pixels every frame, but output some information at native resolution; for example, using a half resolution albedo render target, but a full resolution target containing hints, such as triangle index, that can aid in reconstruction. However, to reduce memory consumption, the Decima Engine avoids native resolution hint buffers entirely. Instead, the pixel corners are shaded each

frame, drastically increasing the amount of information available when shading each region; "2 samples for each pixel to work with, instead of only 1 sample per 2 pixels," as seen in Figure 3. After another frame has rendered, this can be increased even further to 4 samples per pixel. They demonstrate that even with only a single frame of information, their algorithm successfully smooths jagged lines vertically and horizontally, but can suffer on diagonals, which they note is the direction with the sparsest information. They solve this with a form of rotated anti-aliasing; the diagonal direction is reinterpreted as the new horizontal, achieved by storing the data in a packed square 'tangram' texture. This can then have anti-aliasing applied, smoothing out the original diagonals. To create the final output, the current and previous frames are blended where histories match; where histories don't match based on reprojections, only the newest frame is used (Carpentier & Ishiyama, 2017).

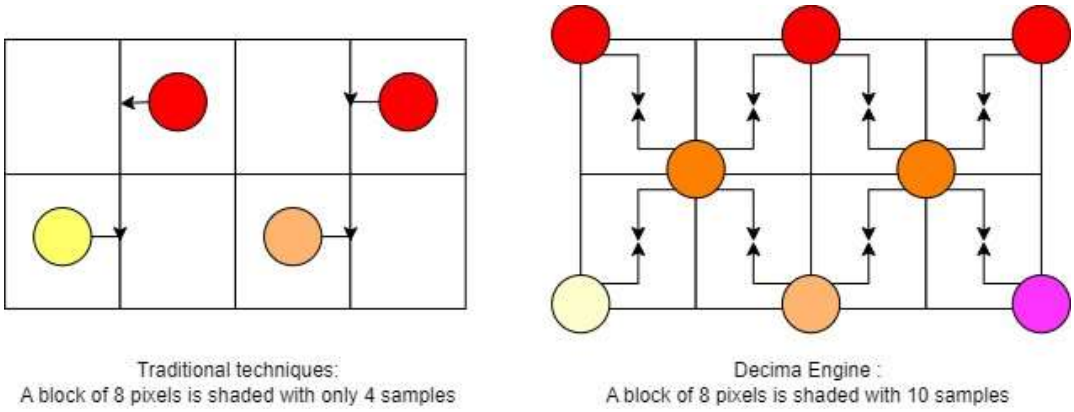

Figure 3 – A comparison of the per-pixel information available for use in reconstruction in a pixel-centre versus a pixel-corner checkerboard render.

A more traditional checkerboard system was used in Rainbow Six Siege; where Guerilla Games use checkerboarding as a means of increasing the screen resolution that can be rendered at 60 FPS, the Siege team aimed to increase performance from 30 FPS to 60 FPS at the same resolution. They use a system they call 'temporal interlaced rendering;' the geometry and lighting target is half the width of the final render target. Like the earlier mentioned motion buffer, a render target holds a 3D velocity vector, and a projection matrix offset. Rendering effects such as shadows that cannot be easily calculated from a velocity render follow a different path; similarly, effects that alternate every frame must remain for a minimum of 2 frames, so that the data is not entirely lost. The checkerboarding effect is achieved by offsetting the sampling of the true render, creating the effect of interleaved shaded and non-shaded pixels. This provides every non-shaded pixel with four fully shaded orthogonally adjacent neighbours. For each unshaded pixel, the depth and colour, both present and history, for each orthogonally adjacent pixel is sampled. The neighbour pixel closest to the camera is picked for motion resolution; following the vector in the colour history buffer gives a colour value to use. It is noted that this can introduce some accumulation errors. This re-projected colour is then clamped

with regards to the colour of all the neighbouring pixels; this ensures that the sampled geometry is not totally out of line with its neighbouring colours. The previous frame depth, following the motion vector, is sampled, to compute a confidence value; a large disparity in depth can indicate that reprojection has failed to sample from the same object. The confidence value is then used to blend back towards the original unclamped value, reducing the risk and effects of a 'bad clamp.' The final colour is then weighted twice more, by the colour coherency (the minimum difference in colour between each of the neighbouring pixels), and the magnitude of the velocity (Mansouri, 2016). The full reconstruction graph is shown in Figure 4.

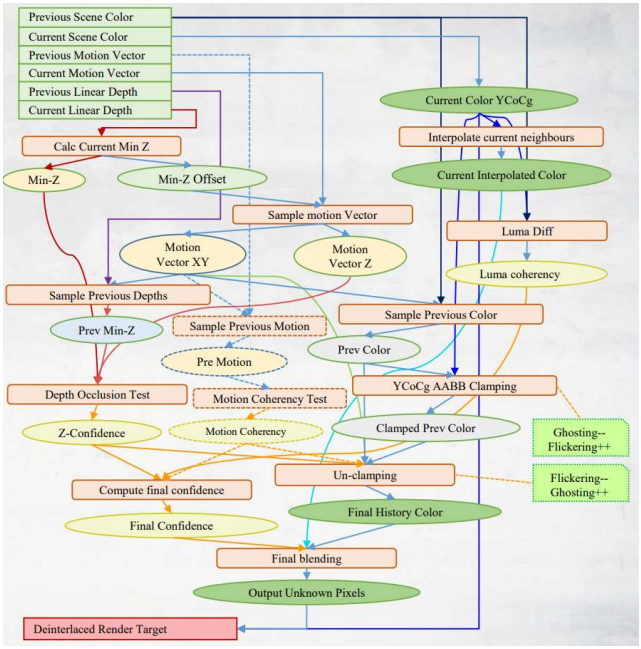

Figure 4 – The checkerboarding system used by Ubisoft Montreal for Rainbow Six | Siege Image Credit: *(Mansouri, 2016)*

Frostbite Labs use a similar solution but started from a stencil-based approach. First trying 1x1, then 2x2 blocks, they attempted to stencil out areas that should not be shaded. 1x1 blocks saw no increase in performance when attempting to render; GPU hardware shades in 2x2 pixel regions, so a 1x1 division means much of the hardware is inactive or wasted, a clear problem for an efficient solution. As such, they tried 2x2 stencil blocks; this was rejected not for efficiency concerns but because of poor sampling distribution, and 'dilated color [sic] bits every 2nd quad' (Wihlidal, 2020). Sparse availability of information is a reasonable concern, but little is explained with regards to the 'dilated color bits'; further research yields no relevant information on what this might mean (Raman & Wise, 2008).

Observing the array of techniques available for modern graphical application optimisation, there seems to be an important common point; doing less work, makes things quicker. Be that the bundling of work with variable rate shading, the reuse of unchanged polygons with temporally adaptive shading, or careful loss of detail in motion adaptive shading, executing fewer instructions, a direct result of fewer shader invocations, makes an application

run quicker.

This is not a trivial aim; customers are unlikely to buy games with mediocre performance (Sacranie, 2010), and graphical fidelity can be a strong selling point for games. Reduction of work while preserving quality allows both reliable performance and graphical fidelity; so far, interlaced checkerboarding has provided this. As a technique, it boasts lowered memory costs compared to standard rendering, and has proven its graphical worth in multiple published titles.

However, it does appear the stencil-based (or depth-based) approach has been somewhat unfairly maligned. Criticisms such as larger required render targets aren't big problems considering the available VRAM available in many modern systems (Steam, 2023) and comes with its own advantages. Where the interlaced method of checkerboard rendering relies on multi-sample anti-aliasing (MSAA), a depth-based approach could avoid this entirely, while still maintaining as much information about the scene. One fair criticism is the lack of information in the centre of a given 2x2 block that was not shaded; in the interlaced approach, the furthest away a pixel can be from true-rendered information is 0.5 pixels; in a stencil-based approach, it can be one pixel away.

4 Method and Methodology

4.1 Tools

With any software experiment, a key question is always what other software to use. Between commercial engines, third party libraries and low-level graphics APIs, finding the appropriate medium is of paramount importance.

Commercially popular game engines like Unreal Engine and Unity (Nardone, Muse, Abidi, Khomh, & Penta, 2023) already have well-featured renderers, supporting 2D and 3D scenes, different models of lighting and multiple platforms. This makes them attractive prospects for testing optimisation techniques. Before making any changes, these renderers have features such as dynamic and static shadowing, complex material systems, and ray traced reflections. Each of these techniques can be responsible for artifacts in accelerated algorithms, and having these readily available helps ensure that test data is as valid as possible.

Alterations to the engine can be costly. Unity locks its engines source code behind a paywall, so changes to the renderer can be financially expensive. Epic Games provide the Unreal Engine source code for free, and this can be freely changed. As such, if an engine were to be used, it would be more suitable to use Unreal Engine 5 (UE5), the most recent in the series. UE5 is a very large codebase, containing not just the renderer, but physics, UI, input, AI, particles and so many other libraries. As such, changes to the source code can be very costly in terms of computation. Some test changes to the core renderer caused compile times of 40 minutes or more, for things as small as changing the value of a constant floating-point number. The large project files also cause long load times for development times. Between all of this, developing for UE5 with the equipment available is prohibitively long for experimental work, where a low iteration time is an extremely desirable feature.

Not using a commercial engine will have drawbacks, as the time cost of developing all the graphics features available in them is equally prohibitive; however, if this work is taken to be a prototype, valuable insight can still come from an implementation in a less developed renderer. Many companies have produced games using custom engines, though many of them are much more developed than any renderer produced for this experiment would be (Crytek, 2023) (Gregory, 2018).

Thus, a choice of rendering API must be made. While development for games consoles would be a worthwhile endeavour, the APIs they use are proprietary and subject to NDAs; this eliminates PS4, PS5, Xbox systems and the Nintendo Switch APIs from consideration. There are multiple available modern rendering APIs for PC platforms, specifically Windows development; DirectX 11, DirectX 12, Vulkan and OpenGL. OpenGL has been largely superseded by Vulkan, much as how DirectX 12 has superseded DirectX 11. Vulkan and DirectX12 are lower-level APIs, leaving work that would once have been left to the driver to the

application programmer; while this does not guarantee better performance, the lower-level APIs are capable of being faster than their predecessors. This makes them attractive choices for optimisation-based tests, as the programmer has the most possible control of the code. Newer APIs also contain features that are not present in older versions, such as Variable Rate Shading. This immediately eliminates OpenGL and DirectX 11 from consideration on that basis alone. Vulkan and DirectX12 are relatively evenly matched, however Microsoft's extensive documentation, the large suite of tools, and the author's existing experience with it make DirectX 12 the more sensible option in this case.

4.2 Measurements

There are two core aims of this work; to reduce the time taken to render a scene, and to preserve the visual quality of the scene. Both will require different means of testing.

4.2.1 Timings

From the perspective of the end user, frame time is the primary metric for efficiency. The critical success of games can depend on maintaining a good frame rate, which is inversely proportional to the frame time; the longer a frame takes to process and render, the fewer can be displayed every second. Frames per second (FPS) is a commonly used metric, with many games aiming for 30 FPS at a minimum, and many aiming for 60 FPS. Some systems cap their FPS, but this will not be considered here.

However, for a developer, there are other metrics that are worth considering. In multi-processor environments, in which work is allocated to different processors, workloads need to be balanced. For example, allocating the GPU only 1ms worth of work while allocating 30ms worth of work to the CPU every frame will cause a stall. As such, frames will take 30ms to execute, but this is not so relevant if only GPU optimisations are being evaluated. Had it been the case that the GPU had been doing 10ms of work but that had been optimised to take only 1ms with a new algorithm, which would still be a valid optimisation, even if the frame time does not reflect it. As such, processor-specific frame time is worth considering.

Within this, it may also be worth taking time measurements of only specific code sections. For example, perhaps the CPU is executing three algorithms, A, B and C, which together take 30ms. If B is optimised, this might take the average time taken down to 29ms. While we can identify the time saving in B to be 1ms in this example, it is possible that accuracy has been lost in measuring A and C at the same time. This can be demonstrated with the example of V-sync; if algorithm C were to present graphics to the screen, and V-sync were enabled, then the system would hang until 17ms had elapsed since the last time graphics had been presented. If B had been optimised, but the measurement still included C, the benefits of the optimisation would not show if the frame took less than 17ms.

V-sync can be an issue in the measurement of graphics optimisations via frame time; as it artificially extends the frame time, it can completely invalidate timing data if only the frame time is considered. This can be addressed in a few ways. The first is to force V-sync off at a driver level. For example, with NVIDIA Control Panel, the developer can disable all V-sync, which eliminates the problem with using frame time as a metric; however, it is generally not a recommended operation. Some graphics APIs allow control of V-sync, however some obstacles were encountered with this; while in theory V-sync is disabled, V-sync was still noticeably occurring, so while this would be the most appropriate solution, it is not feasible on the available systems, be that a hardware or driver issue. The third method is to time the frame excluding the graphics presentation, but this will require both GPU and CPU timings for the same frame; the frame time can be said to be the time of the processor which takes the longest. This also naturally allows the measurement of the processor load balancing.

4.2.2 Execution Metrics

Timings are valuable metrics; after all, for the desired effects an optimisation needs to make the program run quicker. But there are other things to consider; memory can also be an area of concern when optimising, and some metrics are available that correlate to frame time.

Readily available graphics debuggers such as PIX expose a wide range of performance counters to the developer; using these, otherwise hard-to-acquire metrics such as the number of pixel shader invocations across a frame are easily obtainable. The metrics available depends on the debugger and the GPU vendor, but this project will use the features of PIX with NVIDIA chips. Useful available metrics include:

- The number of pixel shaders invoked per draw call; this can be used to confirm that the number has reduced as intended.
- The time taken per draw call; this will be useful for identifying bottlenecks in the code, and good opportunities for optimising the pipeline.
- The pixel shader throughput percentage: if there is no noticeable difference in frame time, a pre-existing underusage of the pixel shader may be responsible.

A desirable metric, found in some graphics debuggers and used in Vaidyanathan et al. (Vaidyanathan, et al., 2014) is the number of shader instructions executed every frame, in millions. This would allow the identification of the number of executions saved with the proposed acceleration techniques, by taking the difference of two techniques; it may transpire that the instructions used in reconstructing the output outweigh the instructions saved. While this is desirable, it is not found in PIX; it can be done without, as instructions executed will theoretically correlate directly with frame time. PIX's features, especially its near-complete DirectX 12 tools, and great ease of use, put it above its competitors.

RenderDoc is another popular graphics debugger, but has underdeveloped DirectX 12 features, failing to include basic information such as pixel history. Intel's Graphics Frame Analyzer does support DirectX 12, including advanced features such as VRS; however, recent developer logs (Intel, 2023) imply that this implementation is not as strong as desired. NVIDIA's NSight debugger has a seemingly complete complement of DirectX 12 tools. However, the PIX debugger still stands out; it exists only for DirectX debugging, and is tailor made for it. It is the most feature-complete of the DirectX12 debuggers, being developed in tandem with the API itself (Microsoft, 2023). Besides that, it is the tool the author is most familiar with, and in a pool of equals would thus be the natural choice.

4.2.3 Memory Usage

Memory consumption should be assessed at the end of the project to ensure that the number of resources used is appropriate, and any excess size accounted for, with potential steps to address said excess size laid out. Memory is the metric of least concern until the technique has proven to be of good speed and quality.

4.3 Image Quality

There is little use in optimising the rendering process if the resulting image is of low quality. Thus, alongside execution time and other important metrics, the image quality must be accounted for.

Existing research provides a good guide of how to go about this. Many papers do not formalise the consideration of image quality, instead leaving it to the interpretation of the reader. This can be seen in papers such as (Vaidyanathan, et al., 2014), in which close-up screenshots of the image are provided, along with commentary as to anything interesting the researcher had noticed, such as the preservation of straight lines. This seems the weakest of the available methods; researchers may have misguided views of their own results, erroneously believing them to be of significantly better or worse fidelity than they really are. It also hinges on the opinion of one person, who may have a different tolerance for graphical error than others.

However, static image analysis tools may not be ideal for this either. Few papers in the field employ algorithms to analyse the quality of the generated image; this appears to be a more important feature for renderers that target film and television, which often demand greater fidelity and photorealism; in games, it seems that error is tolerated more, perhaps because of the strict frame time limits. Games also present potentially unique challenges, in that the predicted image result is not consistent between application instances; perhaps random dynamic waves make a pixel-pixel image quality assessment impractical (Amann, Weber, & Wuthrich, 2013) Some common image quality assessment methods include the Peak Signal-to-Noise Ratio (PSNR), Mean-Squared Error (MSE) and Structural Similarity Index Maps (SSIM).

MSE uses a reference image and a created image and creates an error value based on the per pixel differences. PSNR is derived from MSE. Both techniques fail to suitably reflect visual image quality, because they do not account for the human visual system (Korhonen & You, 2012). Most research makes it clear that good image quality metrics are difficult to find. As such, using image quality metrics seems unreliable in this case; if established techniques find technical errors but underrepresent visually relevant errors, that will misrepresent the final quality, and is likely to be irrelevant, or worse obstructive, to the true results.

Instead, using a survey of human participants is a more robust test. This has been noted in the literature as a useful process (Sheikh, Sabir, & Bovik, 2006). Using multiple human participants should mitigate the effects of individual tolerance of rendering inaccuracies and would prevent the predicted impact of the researcher's perception biasing results. However, this does not appear to be a popular approach in the existing research; perhaps this is because the authors believe their analysis of their own work to be accurate, which seems to be the case most of the time. However, unlike with using automated image quality analysis, there is little apparent risk of human surveying obscuring the results of the experiment.

The proposed experiment is thus to provide videos of a rendered scene to human participants, with them then rating the perceived quality on an integer scale, ranging from 1 (very bad quality), to 9 (very good quality). A clear problem is that there is no baseline for what 'good' is. Providing just one scene and asking participants to rate its visual quality is likely to lead to varying answers, as different users may assign the same level of quality different scores. As such, for each tested scene, users will be given three videos; one of them will use a default renderer, another will be rendered with one experimental technique, and the third with another experimental technique. We can thus use the score given to the default render as a baseline, on a per-participant basis. This both accounts for the individual tendencies of the participants and can also be used to measure the level of change in quality between techniques. There are concerns that providing videos in a pre-determined order may lead users to assume that the default scene is always presented first; this may lead to incorrect ratings, so the order in which the videos are presented will be randomised.

5 Design and Development

5.1 Technique Proposal

This paper will investigate the usefulness of a stencil-based checkerboarded rendering approach, in comparison with standard rendering approaches. In their GDC talk, Frostbite Labs discuss an attempted checkerboarding solution that would use 2x2 stencil blocks to create the checkerboarded effects. They reject the idea based on the sparsity of sampling data; this is not an unreasonable concern but can potentially be alleviated. They also mention an issue with dilated colour every second quad; from some basic tests this is not something that can be replicated; perhaps it was a side effect of the existing engine. Regardless, a depth- or stencil-based implementation theoretically has some merit and could be worth researching further.

The implementations discussed in the literature review generally use ¼ or ½ sized render targets; a depth-based implementation can render to a native resolution buffer with assurance that only ½ of the final pixels are shaded. This makes it of course more costly than a ¼ sized render target, but equally provides much more in terms of detail, a valuable resource for addressing the data sparsity concerns laid out by Frostbite Labs. Unlike a reduced size render target, this also does not condense data unduly in one axis; Guerilla Games compress their x axis by a factor of 2, while maintaining full detail in the y axis. A native resolution render target with stencilled areas ensures that any loss occurs across both axes, hopefully mitigating some of the visual effects of lost detail, if any. This will have the cost of some memory overhead, which is not negligible, but not so much as to be prohibitive; there is no more space expended on render targets than in traditional native resolution rendering, although some extra geometry may need to be stored for the pre-pass, to create the depth buffer.

Addressing the lack of data in the unshaded regions is a primary concern; even a single colour point at the centre of the region would be extremely useful in reconstruction, but prohibitively costly to achieve. Render APIs do not accept multiple render targets of varying sizes (Microsoft, 2021), and each must use the same depth buffer. An ideal solution would write G-buffer information in the shaded spaces and write a one-eighth resolution hint to a target with its own depth buffer. As it is, this would require an entirely separate rendering pass, and as such is almost certainly not worth pursuing.

Alternatively, this could be a use for variable rate shading. Uniform shading of 2x2 blocks is something it excels at, although there is potential friction with GPU hardware here; as Frostbite Labs point out, GPUs shade in 2x2 quads, and it is unclear if this applies to VRS shading also. This would still require a second pass, and with the entire screen being subdivided into 2x2 blocks, it is unlikely to be a better approach than a simpler reduced size render target, which takes up significantly less memory for the same effective resolution. VRS would be more

aptly suited to work for which the shading rate is uniform across a region, but in which regions require different shading rates; that is not the case here.

5.2 Implementation Plan

Stencil-based reconstruction techniques can take cues from the interlaced reconstruction techniques; the Ubisoft team have a graph, shown in Figure 4 that lays out their reconstruction process. While they don't go into great detail, this can be used as a template for this development.

The first steps will be to implement the stencilling by conditioning the depth buffer, and then a basic method of filling in the gaps, using neighbour colour averages and a simple motion transformation. From then on, techniques will be implemented to fix observed problems; for example, if ghosting is encountered, a fix will be found. These problems will be observed in a set environment; a Sponza scene has been set up, with a rotating object in the middle. The solution will be adjusted until this set scene looks 'good enough' at which point the algorithms used will be evaluated, and surveys sent to human participants. This agile approach ensures that algorithms do not become bloated with operations that serve no purpose, while meeting the requirement of producing a scene of acceptable quality.

The Sponza scene was chosen both for its popularity and its variety of features. The multitude of research that uses Sponza (Vatjus-Anttila, Koskela, Lappalainen, & Hakkila, 2016) (Macedo & Rodrigues, 2018) as a test scene makes it an appealing benchmark, as results could be directly compared with other techniques. Part of this popularity is no doubt due to its features; a mixture of thick and thin geometry, some smooth and some rough, with transparent objects, finely detailed meshes, and large planar surfaces. Even the base scene provides plenty of opportunities for testing, so the optional Sponza features such as emissive candles and coloured fabric curtains are not used here (Intel, 2023). Future work could test the algorithm developed in this paper with these Sponza features.

During development, the researcher will be responsible for testing, consisting only of observing the visual quality of the technique. This ad hoc testing is discussed in detail in Section 5.3. Once development is complete, timings will be taken as discussed in Section 4.2, and a participant evaluation of the scenes carried out. This testing is discussed in Section 6.

5.3 Implementation

This section takes the form of a development diary, narrativizing the gradual development of the final software.

5.3.1 Compute Shader Depth Checkerboarding

The earliest development began with a test in Unreal Engine 5. The goal was to run a compute shader stage that would pre-condition the depth buffer to create the checkerboarded effect; two by two blocks set to a depth of either 0 or 1 in an alternating pattern. However, as

previously mentioned, this was hampered by the exceptionally high compile times of UE5 without distributed builds. The development process was too long to be worth continuing, so the move was made to DirectX 12. Upon the change, it transpired that DirectX 12 does not allow the binding of depth buffers to a compute stage. This is not made clear in any documentation but is verified by a Microsoft representative. (Natalie, 2022)

5.3.2 Vertex Shader Checkerboarding

With this restriction, the depth pre-condition will need to use the graphics pipeline. This requires geometry and indices, memory that would have been saved using the simpler compute pipeline. To avoid runtime generation of the geometry used for this, it is generated upfront, one set for the white squares of the board, and another set for the black squares. For this pass, no pixel shader is used, the geometry is simply transformed from pixel coordinates to screen space within the vertex shader, with a depth output of zero, and this is enough to trigger the rasterizer's depth write. It can be verified that this prevents rendering in the desired region

However, this comes with its own memory costs, as seen in Table 1; vertices and indices are stored in GPU memory to enable drawing without the slow process of generating and uploading geometry every frame. There is also a time cost to binding the geometry buffers and executing the draw at the full target resolution.

	Elements	Size per Element (Byes)	Total Size (Bytes)
White Vertices	115200	8	921600
Black Vertices	115200	8	921600
Indices	115200	6	691200
Total Bytes			2,534,400 (~2.4MB)

Table 1 – The memory costs of all the GPU resources required for depth buffer conditioning.

This makes the persistent memory cost just over 2MB; it's not negligible, but it is unlikely to be an issue in modern systems with multiple gigabytes of VRAM (Steam, 2023). The memory cost is directly proportional to the size of the render target; the metrics given are the costs for a 1280x720 pixel render target. Doubling this to a 2560x1440 pixel render target would make this a roughly 8MB memory cost, as the geometry needs to be created to cover the entire target.

5.3.3 Reconstitution

In accordance with techniques outlined in the literature review, the final image needs to be constructed from the half-detailed image created from object shading. This stage is very suitable for a compute shader; work needs to take place on 2x2 blocks, with no overlapping access, at an even distribution across the back buffer, very handily parallelisable. For a given

pixel, its final value is taken to be the value at the location that the same bit of geometry would have rendered in the last frame. For this, a motion buffer is required.

To create a motion buffer, a specialised pass is used; for each bit of geometry, we give its world matrix and camera matrix for both this frame and the last frame. The output location is, as with a regular shade, calculated with respect to this frame's world matrix and camera matrix. However, the stored value is the vector from this output location to the location that would have been used for output were the last frame's matrices used. This vector between pixels can be traversed in our case to find the shaded value of the same geometry in the previous frame. The proposed render graph is shown in Figure 5.

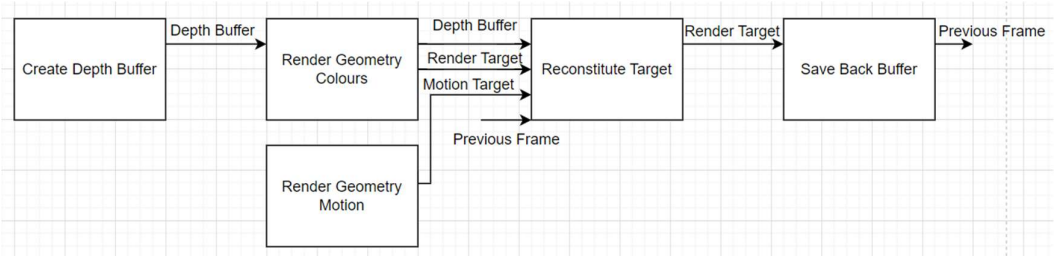

Figure 5 – Scene render graph for a checkerboard render with motion reprojection-based reconstitution.

From the reviewed literature, this seems logically sound. Assuming objects within the scene are moving slowly, the motion vector will be small, and thus when sampling the results of the previous frame, this is most likely to sample pixels that were fully shaded last frame. It can be reasoned that this method will favour geometry that is static (where one would expect an exact calculated value taken directly from the previous frame), or geometry that moves extremely fast, as any lost quality from sampling less-reliable pixels will be hidden by the speed of movement. Potential errors may be seen around geometry moving at a rate of around two pixels per frame; they will be sampling from 'interpreted' shading often.

Using a motion buffer introduces its own cost, both in memory and rendering time. In this basic implementation, it uses a full screen render, with no depth pre-conditioning, meaning it will be paying the full cost of rendering the scene geometry; where rendering colour information would skip 2x2 blocks, motion rendering has no such advantage. The memory cost of the motion target is dependent on the format used for the target; in this implementation, to minimize the risk of precision errors (such as float rounding errors) a large 32-bit per channel format is used, the equivalent of DXGI_FORMAT_R32G32B32A32_FLOAT. The simplest way to reduce the size would be to eliminate the alpha channel, but DirectX12 will not support a render target with the DXGI_FORMAT_R32G32B32_FLOAT (Microsoft, 2022). The Rainbow Six Siege implementation uses a 12-bit red, 12-bit green and 8-bit blue format for the target which is much more space-efficient; a future implementation could, with appropriate testing, switch format, effectively quartering the memory cost of the motion buffer.

The motion buffer has little use in this technique if there is nothing for it to sample; thus, at the end of this frame, the final back buffer needs to be saved. The output of the previous frame is bound as a texture input to the reconstitution shader, and at the end of the frame, the back buffer is copies to that same texture. The same memory is reused between frames.

The render target memory costs this version of the technique can be seen in Table 2.

	Format	**Dimensions (Pixels)**	**Total Size (Bytes)**
Geometry Depth Buffer	24-bit Depth 8-bit Stencil (unused)	1280x720	3,686,400
Render Target	8-bit RGBA	1280x720	3,686,400
Previous Frame Cache	8-bit RGBA	1280x720	3,686,400
Motion Buffer	32-bit RGBA	1280x720	14,745,600
Motion Depth Buffer	24-bit Depth 8-bit Stencil (unused	1280x720	3,686,400
Total Size			29,491,200 B (~28.1MB)

Table 2– Memory costs of all the output targets required for motion reprojection-based checkerboard reconstitution.

5.3.4 Colour Clamping

As discussed in the literature review, the Ubisoft implementation for Rainbow Six Siege uses colour clamping as part of the final computation. Motion vectors may sample unrelated geometry, which can cause a poor looking result.

Figure 6- Left: A sample from a scene with motion sampling, but no colour clamping. Right: A sample from the same scene.

Figure 6 shows the result of non-clamped colour. The left image shows a vending machine used only as a test object for development; it spins around the y-axis (camera-vertical), giving it a specialised motion profile. Especially with this object, relying on motion alone clearly creates a poor effect; the lines are blurred, with the checked pattern clearly visible. The effect is not so pronounced on static geometry as shown on the right; this wall is completely still, and so samples its previous frame value exactly.

An interlaced render as proposed by Guerilla Games and Ubisoft will provide 4 neighbour pixels for colour comparison; this isn't available here, with only 2 direct neighbours with full shading information.

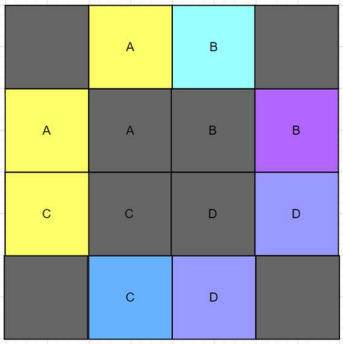

Figure 7 – The layout of a given reconstruction region; a grey pixel represents a pixel that has not been shaded this frame. Letters denote 'logical groups' of pixels.

Pixels within a 2x2 pixel reconstruction region are each grouped with their two directly adjacent pixels that have been fully shaded. This information is indeed sparser than the Guerilla Games implementation (de Carpentier & Ishiyama, 2017) but still provides a lot of information about the context of a given pixel. For example, in Figure 7, if the grey A pixel reprojects along the motion to vector to find green, it is reasonable to suspect that this is a misprojection, as the neighbouring pixels are all yellow. This is addressed by clamping the colour within the range of the neighbours. However, if the grey A pixel were to reproject and to find purple, this appears to be a misprojection, but with the context of the full square, it could be a very reasonable colour for that pixel. With the information available for this pixel, this potentially correct purple reprojection would be clamped into being yellow.

Figure 8 – Left: A wall lamp from a scene with motion sampling and colour clamping. Right: A sample from the same scene, of the same lamp.

An example of this can be seen in the lamp in Figure 8; this geometry is static (with neither it nor the camera moving) so should reproject perfectly. However, applying uniform colour clamping causes good reprojections to be destroyed, with alternating frames rendering completely different lamps. This creates a distracting flickering effect, especially around thin geometry like these lamps, and the thin edges of the curves above the Sponza archways. In some cases, such as with the dents in the brickwork, some geometric detail is destroyed in the flicker.

This presents a dilemma; with uniform colour clamping, dynamic faces, such as that seen in Figure 9, are vastly improved, but static edges are ruined. Without clamping, dynamic faces suffer, but static edges remain largely unaffected.

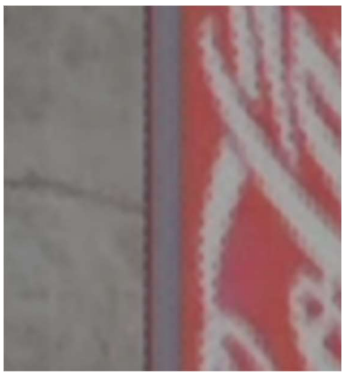

Figure 9- A face of a spinning object, reconstituted with motion reprojection and uniform colour clamping.

5.3.5 Conditional Clamping

So far, uniform colour clamping has been employed, meaning that for each pixel, it is clamped between its neighbour pixels. To alleviate the problems found around thin lines, or intricate details, conditional clamping can be employed, meaning that the colour is only clamped under certain conditions.

One of the simplest is a 'small motion clause.' This rules states that if a given pixel has very little motion, it is much more likely to be reprojecting to the correct value and can thus likely avoid a clamp. This can be used to cut down on the flickering of thin lines with little overhead; a shader branch will make the shader pay for the long path as long as even one thread takes that path (Rogers, O'Connor, & Aamodt, 2013), but this is acceptable. Paying the time for clamping is useful for dynamic surfaces, but this simply provides a way for very promising reprojections to be preserved. The Rainbow Six Siege implementation uses velocity magnitude as a blending weight (explored later), but here is used as a simple switch (Mansouri, 2016).

Figure 10 – All images from the same scene reconstituted with motion reprojection and motion-conditional colour clamping. Left: A lamp. Middle: The same lamp from a different frame. Right: Static geometry bordering a dynamic face.

Figure 10 demonstrates the use of this; in the rightmost picture, the dynamic faces maintain all the benefits of clamping, with appropriate colours giving a good enough impression of the desired image. The other images demonstrate the benefits when applied to static geometry; the rendering of this lamp is consistent, and much more pleasing than the blurry and flickering results previously. This is good both for a static image, and for a video, where the flickering no longer draws the eye.

When sampling the motion, its magnitude is compared against a constant value to determine if clamping takes place; this constant is a matter of fine tuning and is likely to be changed per application. The implementation shown in Figure 10 uses a value of 0.0000001, meaning that it will for the most part only prevent clamping where a non-zero motion is the product of a rounding error or filtering error (Even & Seidel, 2000) (Mitchell & Netravali, 1988). Applications with 'blockier' geometry could likely use a much higher value here.

5.3.6 Motion Optimisation

Based on observing the patterns of motion sampling in the reconstitution shader, there is currently no need to sample the motion for this frame outside of the reconstructed regions. Thus, an inverse checkerboard is suitable; motion and G-buffer information are rendered in separate passes, so can use a different depth target. As such, a second depth target can be created, with the opposite checks filled in. This combined approach does not sacrifice any necessary data, and helps cut down on shader invocations and instructions, one of the main aims of this technique. Overlaying the two, an interlaced structure can be created, providing all the necessary information for shading this scene with no wastage, demonstrated in Figure 11.

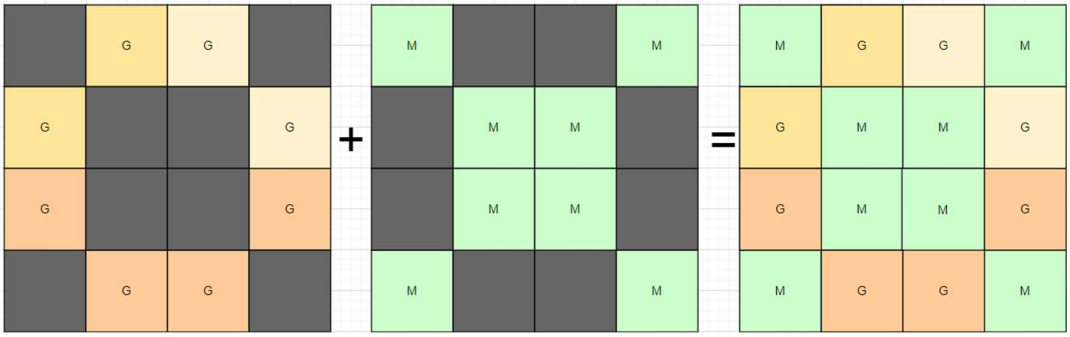

Figure 11- Interlacing the geometry buffer (left) with the motion buffer (middle) to create a fully informed reconstitution model (right).

A reorganisation of the depth creation process is thus in order; where before the appropriate geometry was used to generate one configuration of the depth buffer each frame, both configurations are now needed every frame. To run the vertex shader again, regenerating the same depth buffers every frame would be wasteful; as such, depth creation is moved to application startup. The same vertices and indices are used as before, but deleted after the depth buffers are first created. Instead, two persistent standby depth targets are maintained, used as a copy source; instead of running a shader to create the depth target, the premade one is simply copied in. The new render graph is shown in Figure 12.

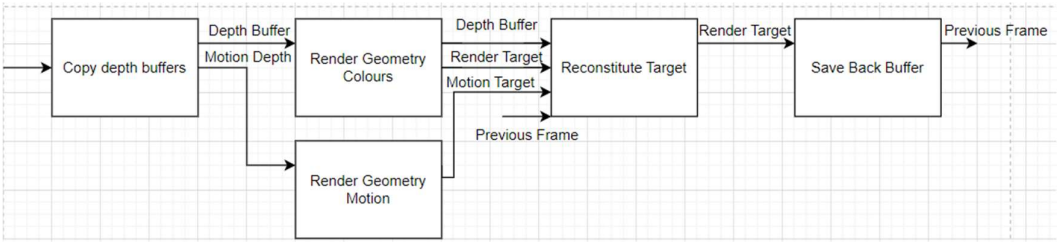

Figure 12 – Scene render graph with new motion depth pre-condition.

An unleveraged advantage of this is that overlaying the motion depth target and the G-buffer depth target can provide a complete depth buffer for the scene, every scene. This could be useful in future extensions of this work. However, it does come with the drawback of maintaining extra depth targets; this has a memory cost of 7MB, seen in Table 3, although should be considered alongside the savings on vertex memory, itself about 2MB, giving a final ~5MB increase in memory consumption.

	Format	**Dimensions (Pixels)**	**Total Size (Bytes)**
White Check Depth Buffer	24-bit Depth 8-bit Stencil (unused)	1280x720	3,686,400
Black Check Depth Buffer	24-bit Depth 8-bit Stencil (unused)	1280x720	3,686,400
Total Size			7,372,800 B

	(~7.0MB)

Table 3 – The memory cost of new depth buffers used as copy sources.

5.3.7 Blended Clamping

The first implementation of clamping was uniform colour clamping, in which every reprojected pixel colour was clamped between its neighbouring 'true render' pixel colour; then came conditional clamping, where the magnitude of sampled motion is used to verify that a reprojection is likely to be correct, with pixels with low velocity not clamping at all. Ubisoft's Rainbow Six Siege implementation uses weighted clamping, in which a point between the pure reprojection colour and the clamped colour is used, based on 'confidence values' calculated from other available information (Raskar & Low, 2002).

For this, depth history can be used. Previously, colour history has been used when reprojecting to find the expected colour of a pixel. Similarly, if a depth buffer is saved at the end of its frame, a given pixel can be motion-reprojected to find the depth value that same pixel is predicted to have resided at in the previous frame. In the Rainbow Six Siege implementation, which uses interlaced rendering (Mansouri, 2016), this could be compared to the natively rendered depth, but in this model, would need to be compared to the neighbouring two true-rendered pixel depths.

Before working with depth values, they need to be linearized; the standard projection matrix favours objects close to the camera, in that the majority of the [0,1] range is given to objects close to the camera (Depth Precision Visualized, 2023). Objects further away were found to largely occupy the [0.98,1] range, with some object depths differing by less than 0.01 despite having world Z differences of 5-10 units. Linearizing depth requires knowing the near- and far-Z values of the camera (the distances of the near and far planes), which are given as shader macros. Other implementations could easily move these to an upload value.

With the reprojected depth and the two neighbouring depths linearized, the differences between the reprojection and the two neighbours can be found. Two constants are introduced here; the 'minimum blend difference' is the smallest value of depth difference that will enable depth blending; any value lower than this will always use the reprojected colour. The 'maximum blend difference' is the smallest value of depth difference that will deem the depth 'mis-projected' and use the full clamped colour. The minimum blend difference serves to avoid blending based on rounding errors. The maximum blend difference serves to provide a reasonable range of depth disparity; huge differences in Z are likely to be better served by a full clamp than a blend.

Blending is applied bi-directionally, as the neighbouring pixels may be of different depths. It may be found that the vertical neighbour pixel has a linearized depth difference of near zero, while the horizontal neighbour has a linearized depth difference of over 0.1 (10% of the available depth space). This can be relatively common, especially where objects overlap.

The reprojected colour is interpolated twice against the clamped colour, using the directional confidence values (themselves calculated proportionally to the maximum depth difference).

The two blended colours are then themselves blended, with the mid-point of the colours taken as the final output colour. This has its own problems; in cases where the two neighbouring pixels are of different objects, the blending suits neither pixel. The result should tend towards a colour that looks suitable, even if it is much less likely to be perfect.

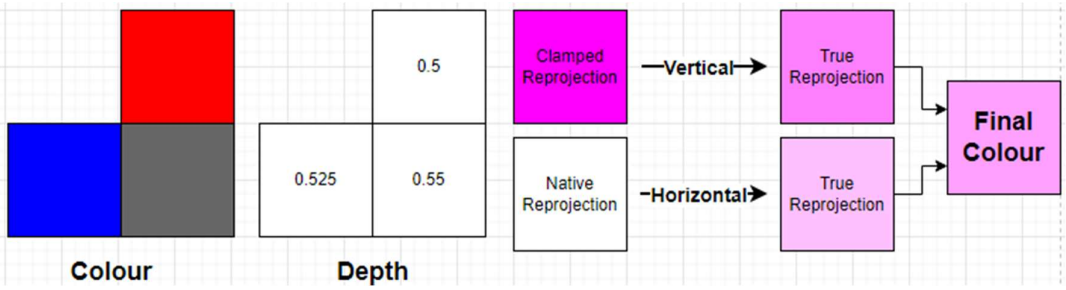

Figure 13 – A visualisation of logic related to the depth blend. Nearer depth values indicate that more of a weight should be given to the native reprojection.

As Figure 13 demonstrates, the colour is not a necessarily good match, but it should not look out of place next to its neighbour pixels. It is close enough to the native reprojection that, for a moving object, this should be acceptable, given the human visual propensity for natural motion blurring. The above example is assumed to be using a 'minimum blend difference' of 0.005, and a 'maximum blend difference' of 0.1. These are the values used in the final implementation; as with other constants, this can be tailored to each application, mostly as a process of gradual adjustment. A higher 'minimum blend difference' will favour scenes with accurate reprojections, thus few moving parts. A lower 'maximum blend difference' will reduce the influence of bad reprojections, but potentially overrule some good ones. These values may even be best varying by scene and can be tailored for future uses.

This technique will also require the depth buffer to be saved at the end of the frame and given as an input texture for the final reconstitution process. This will increase memory consumption (as depth history needs to be saved) and will require a small amount of time to copy the depth buffer at the end of the frame. The new render graph is demonstrated in Figure 14.

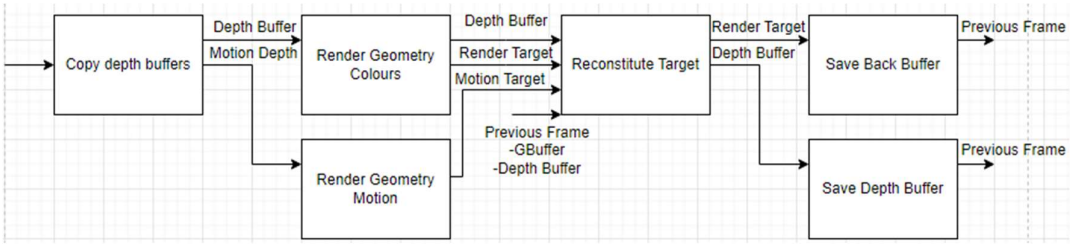

Figure 14 – The scene render graph with the depth buffer saving now included.

5.3.8 Motion Coherency

At this stage, static cameras look fine, but there is a noticeable error with moving objects, and some problems with moving the camera; ghosting can occur in the areas where an object has recently been but has since moved out of, such as in Figure 15.

Figure 15 – Left: The corner of a rotating image; the sawtooth pattern is caused by reconstruction pixels using a motion of zero (that of the object behind the box), which reprojects it onto a moved object. Right: The same object but with motion coherency enabled.

This is addressed with a 'motion coherency' test; it was discovered that the issue was occurring because motion rendering only applies to this frame; for example, rendering a cube's motion on the previous frame may have a wide profile, while on this frame it has a small profile. In Figure 16, the orange square is the only part of the final frame that will accurately render motion; for the remainder, it is likely to have motion of near zero. Thus, if reconstruction were to happen, those uncovered areas would reproject to the rotated cube, even though those areas should not be showing the cube.

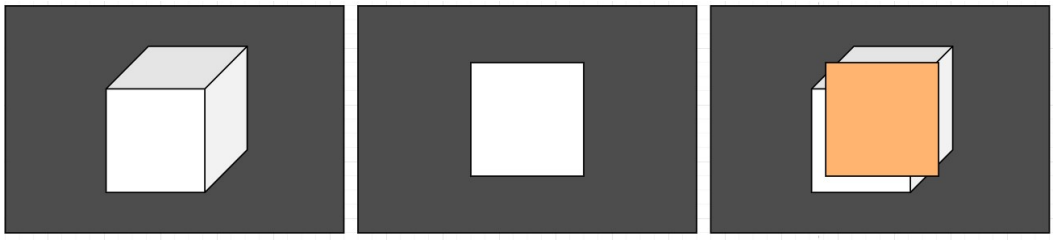

Figure 16 – Left: A cube rendered with a rotation applied. Middle: A cube rendered face on. Right: The face-on cube overlaid on the rotated cube. The cube outside of the orange would not be covered by motion rendering.

To solve this, the motion buffer is saved at the end of the frame, just as the final target is, and is given as an input to the shader. At this point, the shader uses colour history, depth history, and motion history targets. This frame's rendered motion is compared bidirectionally to the neighbouring history motion; if either neighbour's motion is large enough, colour clamping is enabled, if it would not otherwise. The reasoning is that an object motion should not change dramatically between frames; it may, and that is acceptable, but largely this assumes that objects will have somewhat continuous movement. If all the motions align, the true reprojection can be used, but otherwise, the clamping path is followed. No blending is done using the motion

differences as weights; this could be included in a future implementation, but the simple strategy of enabling clamping seemed to address the problem.

It is worth acknowledging that as features have been added to the reconstruction, true reprojections are used increasingly infrequently. This is not such a bad thing; of course, obtaining a true reprojection is the best-case scenario, as it will be the most correct representation at the lowest cost, but those accurate pixels cannot be at the cost of the quality of the rest of them. While many of these techniques risk discarding good pixels, that is preferable to keeping bad ones.

6 Testing

With the project implemented following the process laid out in the Design and Development section, the focus can now move to testing. As discussed previously, there are two key metrics for consideration; the visual quality of the scene, and the time it takes to render that scene. A standard rendering algorithm can be used as a baseline upon which to compare both metrics.

The Scene

Both techniques for testing (standard rendering and depth-based checkerboarding) use deferred rendering as the core rendering 'mechanism;' both have a motion pass, as this information can often be required for other techniques, such as motion blurring. Both techniques also end the frame with a gamma correction step, which shifts the colour space into SRGB (Anderson, Motta, Chandrasekar, & Stokes, 1996). The Sponza scene is loaded; this is the most basic version of the scene, with only the brickwork, metalwork, and wooden doors. Further work could run tests with a fuller version of the scene, including the optional curtains, emissive lights, and water effects (Intel, 2023).

The camera follows a set animation track, moving around the scene, and changing the view direction. Movement is varied to include slower and faster camera motion. The camera does two laps of the set track before coming to rest in the same position it began. This camera motion is recorded, for both versions of the scene. This video is recorded for two reasons; to avoid the interference of participant hardware in the final testing (poor or outdated hardware could be feature incomplete or cause distracting drops in frame rate), and to reduce impact on participants, sparing them from downloading and running whole applications. Each video is no longer than 45 seconds.

6.1 Visual Quality Test

6.1.1 Layout

As discussed in the literature review, human judgement can be an important part of analysing the visual quality of rendered scenes. Standard image analysis techniques are not as subjective as a human can be what may appear wrong to the algorithm could be rather unnoticeable to a human user (Amann, Weber, & Wuthrich, 2013). This subjectivity applies only to the visual quality of the product; while application frame rate (and by extension efficiency) can impact perceived quality, stronger objective measures exist for measuring the timings of rendering. As such, human participation can be limited only to their input on the visual quality of the scenes produced by a given technique.

To enable the test to reach as wide an audience as possible, Google Forms was chosen. Google Forms is a free service, both as a creator and participant, and comes with many useful features; email verification allows the researcher to ensure that no participant has filled in the

form twice; export to Google Sheets allows easy analysis of the collected data (Google, 2023). Free access makes Forms the most appealing of the available services; for example, Survey Monkey charges a fee to create forms.

The first section of the form, seen in 'Appendix E – Human Participant Form' acts as the participant information sheet and consent form; if any participant does not consent to the statements laid out, they are not asked any further questions, or shown the test videos. Once the participant has completed the consent form, they are shown the introductory message.

> *On the following page will be a video of a scene rendered with one of two algorithms; you should watch this video once, and then answer the following questions on your impressions of that video. The following page will then show another video of the same scene but rendered with a different algorithm. You will be asked the same questions about this scene. You will then be asked a final question on the next page.*
>
> The order of the videos is randomised, but the same for each participant.
>
> *Please read all the questions on this page and then view "**Test_Effect_1**" from the link below. For best results, please download the video before viewing [Link Removed]*

The page with the video is headed with this text. Google Forms embedded video was not used for a few reasons; foremost, it uses YouTube as a host, which entails quite destructive video compression. This would be detrimental to achieving fair and accurate results. There were other issues too, such as the potential of YouTube automatically showing a video at low resolution, and that most browsers will not allow a video to go full screen when embedded in a Form. As such, it was decided that users should download the video, to ensure minimal influence of the testing medium on the result.

A 1-9 rating system is used for the videos; results are taken to be comparative ratings, with the difference in answers taken to be the information of most importance. This is a slight variation on the Absolute Category Rating (ACR) test, standardized by the International Telecommunication Union for video quality assessment. An alternative is to make the first video the unaltered version, and ask the user to rate the degradation in quality. Both are considered optimal choices of test; here the ACR test was chosen to allow order randomization, and to simplify the questions (Nehme, Farrugia, Dupont, & Lavoue, 2019).

6.1.2 Results

Overall, participants seemed to prefer the standard rendering of the scene. The standard scene was given an average score of 6.2, while the checkerboarded scene was given an average score of 4.3. The standard deviations of scores given were similar for both scenes, with the score given to the checkerboarded scene being slightly more varied at a SD of 1.7, versus 1.4 for the standard scene. This could be taken as an indicator of preference for the standard scene.

No participants rated the checkerboarded scene better than the standard scene.

Eight participants left comments; of these seven participants identified either jitter, blur, or graininess in the scene. For example, one respondent notes "visual jitteriness around the edges of some objects while the camera is moving." This makes sense; a still camera in a still scene would mean good motion coherency, and such perfect reprojection, meaning no jitter. The introduction of camera movement invalidates some reprojections, and it is clear this has not been sufficiently addressed, leading to this jitter. Edges are a clear weakness of the implementation, with another respondent mentioning that "[A]rtifacts were particularly visible around the edge of the single door." This could be addressed in a number of ways; Ubisoft propose a 'detoothing algorithm', designed for removing vertical pixel sawtooth patterns, (a line of 5+ pixels of alternating colours), which could apply here (Mansouri, 2016). This may also be the result of over-aggressive colour clamping, and could be alleviated by adjusting the constants related to this.

Edges are not the sole source of this; one respondent comments that "the textures … have a blur over them", and another that there was "a persistent grainy effect visible on most surfaces". This blur and graininess are more likely to be due to incorrect colour clamping or blending than failed reprojections; this could perhaps be remedied with some alterations to the constants used in the reconstitution shader but may need more to resolve. Surfaces are the "easiest pickings" for a technique like this; reprojections should be accurate, and neighbour pixels are highly likely to be of the same object. However, persistent, and noticeable blur lends some credence to Frostbite Labs' statement that there is simply insufficient information to sample to accurately reconstruct a 2x2 region.

Others discussed the other features of the scene, such as shadowing (shown in Figure 17), colours, and lighting. These were not the focus of the experiment so will not be fully explored here. For example, shadowing is not implemented, beyond the natural simple shadowing from considering normal vectors; considering feedback on shadowing is not worthwhile.

Figure 17 – The underside of an arch; the natural shadowing has a clear 'spine' of light down the middle. Taken from the checkerboarded scene, and not present in the standard scene. This only occurs every so often.

6.2 Performance Metrics

6.2.1 Timings

As discussed in the Methodology section, performance can fall into two broad categories; time-based metrics, encompassing frame time and associated values, and memory-based metrics, accounting for the spatial costs of buffers, textures and other objects that are required for a given technique. All metrics are taken for the Release build, using DirectX12; many are measure using PIX. All raw metrics can be found in Table 5 and Table 6 in Appendix D – Performance Metrics Tables.

Average frame times show a slight improvement for the checked version, with an average frame time of 49.6ms for the checked version, and 50.4ms for the standard version; this is a 1.6% reduction in frame time. Using the in-built profiler of Visual Studio 2022, it can be observed that most of the frame time is spent in the swap-chain presentation stage. In this implementation, this indicates a GPU-bound application (Thinakaran, Raj, Sharma, Kandemir, & Das, 2018) with the CPU having completed its work before the GPU. In the checkerboarded version, an average of 76.4% of the frame was taken by the swap-chain presentation stage; in the standard version it was a significantly larger 91%. The clear implication here is that the GPU work completes quicker in the checkerboarded version than in the standard version.

To get more information on both techniques, some metrics were gathered for the Deferred Mesh Draw functionality of the pipeline; in both versions, this stage takes mesh and texture data, performs the necessary camera, view, and world transformations, and writes geometry information for a G-buffer, ready for lighting. As this is a stage that encompasses a lot of pixel shader work, it is an important stage to examine; the core premise of checkerboarded rendering is saving rendering time by cutting down on pixel shader work.

Taking timings for 60 seconds, and averaging over five separate application instances, the checkerboarded version of the deferred mesh rendering took 296.3us, a 3.3% reduction of the 306.6us of the standard version. This is significantly lower than expected; by checkerboarding, the aim was to cut pixel shader work by around 50%, but this clearly is not translating into noteworthy results.

Pixel shader ‘work’ can be directly tied to both invocations, and instructions executed. In the standard version of the scene, 959,773 pixel-shaders were invoked; in the checkerboarded scene, it was only 479,923 (Figure 18). This is a just under 50% reduction, a strong improvement. This can also be seen with an 88.5% increase in depth-test rejections between the checkerboarded scene and the standard scene. The checkerboarded scene causes 42.2% fewer pixel shader warps to be launched than the standard scene (Figure 18). This all combines to prove an effective reduction of pixel shader work.

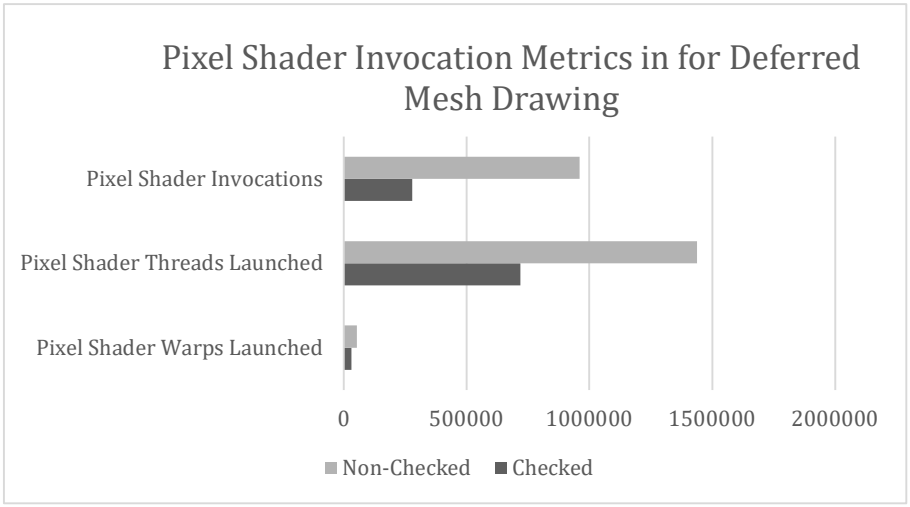

Figure 18 – Comparative metrics of non-checked and checked techniques, related to the pixel shader.

The question remains as to why so little benefit is seen in the final frame time, even with the clear reductions in work. It appears that instead of being pixel-bound, the GPU is vertex-bound. This is best summed up by the throughput of the Inter Stage Buffer Entry (ISBE) allocation; an ISBE holds vertex and primitive attributes (US Patent No. 8947444B1, 2008), and in both versions of the scene, the ISBE allocator was at an average 94% throughput, with many individual draws reaching 100% throughput. If the ISBE cannot allocate for vertex (or tessellation or geometry) information, it will stall warp launches until it can (NVIDIA NSight, 2023). It thus becomes clear that the primary restricting factor here is vertex processing. This is to be expected; Sponza is a detailed scene, with some meshes comprising hundreds of thousands of primitives, and this implementation clearly does not adequately address this. This bottleneck can be put down to a lack of a LOD system (Levenberg, 2002), a lack of a camera frustrum test (Lengyel, 2005), and other standard optimisations that will reduce the amount of vertex processing.

Even without being bound by an ISBE bottleneck, vertex operations would still be obscuring the results. The checkerboarded scene executed 42.1% fewer pixel shader instructions than the standard scene, but only 16.9% fewer instructions in total (Figure 19). The earlier mentioned 42.2% reduction in pixel shader warps launched matches with only a 12.1% reduction in overall warps launched. Simply, regardless of whether the optimisations to pixel shading are useful, which they seem to, that is not the part of this pipeline which most needs the optimisation.

For the reconstitution stage, there is no direct comparison available; the standard version of the scene does not have one. However, there are good indicators as to its effectiveness; compute throughput for the reconstitution phase reached 97%, a good indicator that it could be bottlenecked. An ideal situation would be all compute stages executing in tandem, but this is constrained by register allocation; 15% of attempted warp launches are

stalled due to failed register allocation. This can be remedied either by optimisation of the shader itself, such as cutting down on data sizes (such as using half instead of float) or could be an ideal use for asynchronous compute (Bondarev & Niverty, 2021). Asynchronous compute does not make sense in this implementation, but in a more fully-fledged renderer, it could make sense to run on the dedicated compute pipeline, saving register space for 3D work.

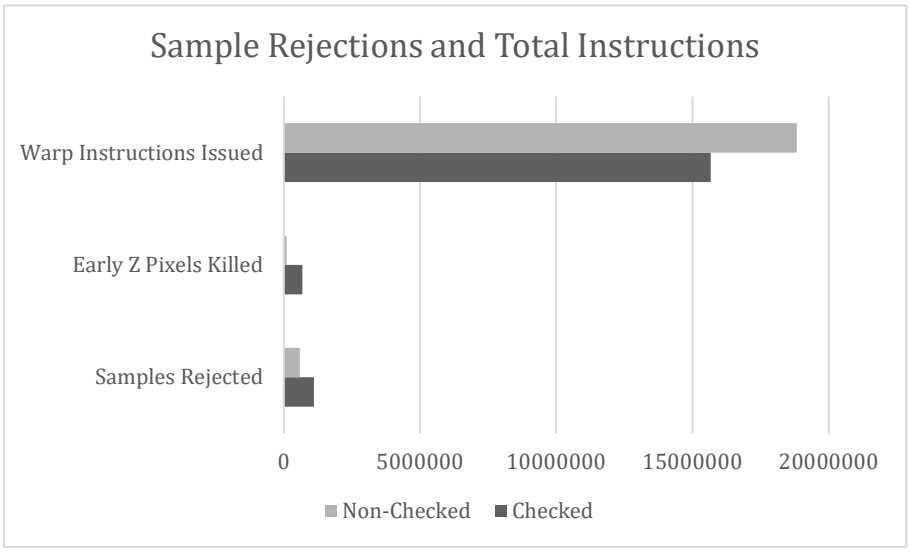

Figure 19 – Metrics related to sample rejections and Z kill, versus instructions issued. There is an inversely proportional relationship.

The stage comprises 59,520 warp launches, covering 61,440 compute shader executions. Considering 479,850 pixel-shader invocations were saved in the deferred mesh drawing alone, this seems a good improvement. The entire reconstitution stage was comprised of 2,976,000 instructions; combining this with the deferred mesh drawing, the creation of the final render target took 18,588,252 instructions, a reduction of 1.1%. However, this does not account for the instructions saved in the motion rendering, and does not account for the constant instruction count dedicated to vertex processing.

Considering only the pixel shaders and the compute stage can give a more accurate impression of the savings, as it considers only the saving of what was 'available to save.' The standard version of the scene executes 7,540,260 pixel-shader instructions; the checkerboarded scene executes a combined 7,338,540 instructions, giving the checkerboarded scene a 2.7% reduction. This saving is not particularly significant; however, this can be expected to go up as more complex shading techniques are implemented.

The number of instructions taken for a given reconstitution phase is constant, with respect to the size of the back buffer. This is not true of shading, where the shading cost is dependent entirely on the number of objects visible, the complexity of the lighting in the scene, and many other factors. It is expected that a high-end commercial game will implement techniques beyond those demonstrated in this experiment; for example, volumetric fog would be expected to yield a better per-pixel instruction saving than a simple pixel shader that outputs a

constant colour (Wronski, 2016). Pixel shader cost will scale up; the compute cost will remain the same.

From the available evidence, it is inconclusive as to whether this technique has the intended performance effects; improvements have been less than expected, due in part to other obfuscating factors. The huge negative impact of vertex processing has made taking useful timings difficult, and the low complexity of lighting and shading in the scene has made differences seem very subtle. It seems appropriate that another test, using an identical reconstitution algorithm, but in a scene with more complex shading would be appropriate. Many games use ray-traced reflections, fog, water effects, and shadows, all features missing from this renderer (Christensen, Fong, Laur, & Batali, 2006) (Liu & Pang, 2009). This is what really makes the results inconclusive; the pixel shaders are not being given enough work to accurately measure the benefits of replacing them with checkerboarding.

6.2.2 Comparison with Existing Implementations

The Rainbow Six Siege implementation presented by Ubisoft has been a key resource in developing the techniques used in this paper, and as such is the best place to start for comparisons. Their resolve shader costs 1.4ms of frame time, a significantly longer time than the tested reconstitution shader, which takes on average only 120.2us. This is not a direct comparison, as the Rainbow Six Siege render target is slightly larger; they target 1920x1080 (2,073,600 pixels), whereas the test application targets 1080x720 (777,600 pixels), making the Rainbow Six solution have to handle roughly 2.6x the number of pixels. Weighting by number of pixels, our solution can be assumed to take roughly 320.73us. (This may be an oversimplification; there are other limiting factors than number of pixels, but is appropriate enough as a demonstrative approximation).

Taking that to be true, this is a 77% reduction in time in the reconstruction step. There are plenty of obfuscating factors; Rainbow Six Siege is a AAA game with a complex graphics pipeline, and its occupancy patterns are likely to be very different to this demonstration. Reconstruction shaders may be contending for warp space with UI rendering pipelines, for example. Ubisoft also include a temporal anti-aliasing (TAA) (Labrador, 2018) step in their reconstruction shader, which could account for a significant portion of this time, though they don't break down how much of the time is spent on this. However, this dramatic reduction could suggest a performance benefit to stencil-based techniques over interlaces techniques.

The reconstitution shader developed in this paper operates on a 2x2 block in every invocation, whereas the Rainbow Six Siege implementation appears to operate on a single pixel in every invocation. This should theoretically favour the Rainbow Six Siege implementation, as the work per invocation is minimised. Larger reconstruction blocks should mean fewer dispatch calls are required, which could account for some of this speedup, but that should be minimal. The performance improvement could be somewhat due to the dispatch, invocation, and

occupancy patterns, but is likely to be due to operations that Ubisoft's implementation carries out, that this implementation does not.

Despite the lower execution time of the reconstruction phase, Ubisoft note a far greater saving per-frame, of 8-10ms. This is significantly higher than the savings found in this paper, but this is likely due to the previously discussed obfuscating factors.

Luma coherency tests (Malvar & Sullivan, 2003) and bounding-box based clamping are a channel of Ubisoft's reconstruction graph that was not covered in this implementation; this is due to a lack of information on the techniques they used, and because it did not meet the criteria set out in Section 5.2. Missing these could account for both the performance gains, and the difference in visual quality; where human participants indicated that the checkerboarded scene was of inferior quality in the test, Ubisoft customers did not report any difference in quality.

Overall, it appears that the created checkerboard technique is inferior in terms of visual quality to Ubisoft's implementation, but this could be addressed by refining the reconstruction shader, whether by implementing further techniques they lay out, or others. These changes could impact the apparent performance improvement over the Ubisoft implementation, though that would be acceptable to improve visual quality.

Guerilla Games' Decima Engine checkerboarding implementation executes the resolve step in <2ms. This renders in 4K (de Carpentier & Ishiyama, 2017), so is of a much higher resolution than the solution described in this paper. We avoid the tangram stage used in Decima, which was intended to help with 'zigzaggy' lines; exclusion of techniques such as this may be responsible for the graininess described by some respondents. Weighting with respect to pixel count, our implementation would take 1.2ms, a saving on the upper bound of 2ms given by Guerilla Games. This is less of an improvement that compared with Ubisoft's implementation, time wise, but there is still a distinct difference in quality graphically.

The Decima implementation is much less complex than the Ubisoft implementation, an advantage afforded to it by 4K rendering; one off pixel does not look as bad when its 1 in 8 million (Wihlidal, 2020). This explains the cheaper per-pixel reconstruction, and the lack of tests such as motion coherency and depth confidence. The implication here is that checkerboarding is an effect better suited to larger resolutions; two interlaced techniques require very different complexities of reconstruction, and this could lie in the target size. A 360p render target would surely be impacted hugely by some checkerboarding effects; this may be a limitation of the testing environment, and increasing the size of the render target may improve the visual quality of the checkerboarding technique, with no other alterations.

Finally, Frostbite Labs also saw a greater performance benefit, with a 24% reduction in frame time in checkerboarded scenes at 1800p. The techniques applied in reconstruction are more complicated than in the Decima engine, using hints such as primitive index rendered at a

native resolution. They explain that some of their shading cost is due to the packed target, with 10% extra time spent adjusting gradients to enable the interlacing of pixels, a cost a stencil-based implementation will have avoided. They also explain the advantages of using 16-bit floating point numbers in shader calculations, at a 30% performance improvement, a missed saving in the stencil-based implementation. (Wihlidal, 2020)

Compared against all interlaced techniques in literature, the stencil-based approach fails to deliver improved performance without loss in visual quality. This could be due to the sparsity of sampleable data, as posited in (Wihlidal, 2020), but could also be the result of the targeted resolution, with the test program targeting 720p, much smaller than the standard 1080p, 1800p or 4K that other checkerboard implementations target. There appears to be a correlation between better performance gains and larger resolutions, perhaps enabling the simplification of techniques required for accurate reconstruction. Cost per pixel is observed to be lower than in some existing implementations, but this is somewhat overshadowed by the inadequate quality of the pictures produced.

6.2.3 Memory

Memory can be a metric of concern, especially when targets are large. In this implementation, higher precision targets have been preferred, such that effects can be judged as freely of precision errors as possible.

	Format	**Dimensions (Pixels)**	**Total Size (Bytes)**
Geometry Depth Buffer	24-bit Depth 8-bit Stencil (unused)	1280x720	3,686,400
Render Target	8-bit RGBA	1280x720	3,686,400
Previous Frame Cache	8-bit RGBA	1280x720	3,686,400
Motion Buffer	32-bit RGBA	1280x720	14,745,600
Motion Depth Buffer	24-bit Depth 8-bit Stencil (unused	1280x720	3,686,400
White Check Depth Buffer	24-bit Depth 8-bit Stencil (unused)	1280x720	3,686,400
Black Check Depth Buffer	24-bit Depth 8-bit Stencil (unused)	1280x720	3,686,400
Previous Frame Depth Cache	24-bit Depth 8-bit Stencil (unused)	1280x720	3,686,400
Previous Frame	32-bit RGBA	1280x720	14,745,600

Motion Cache			
Total Size			55,296,000 B (~52.7MB)

Table 4 – Final memory costs of the resources required for our depth-based checkerboarded version.

As seen in Table 4 the final memory cost is around 52.7MB. This is small due to the small render target; these costs can be expected to grow with respect to the size of the render target. There is room for improvement, reducing the precision per component of the motion vector; Ubisoft suggest a 32-bit packed scheme (Mansouri, 2016). While memory reduction can be an incidental advantage in checkerboarded techniques, the primary concern is the availability of important data. Our implementation does not dramatically exceed the memory costs of other implementations, except in the motion buffer, which can be easily and simply remedied, and uses a comparatively reasonable number of resources (Mansouri, 2016) (Wihlidal, 2020).

7 Critical Review

The project has been inhibited by a number of factors.

Perhaps foremost is the number of independent variables. When measuring any form of rendering acceleration, there are many things to consider, including the number of scene lights, the detail of the geometry, the targeted resolution, the targeted platform, and existing acceleration techniques in the engine. Failing to account for any of these hampers the ability to fairly compare with existing implementations; at best a weighted estimate of metrics can be calculated (Section 6.2.2), but for others, no true metrics can be calculated. A truer test would be to implement an interlaced checkerboard technique within the same engine that tests the stencil-based technique, as this will account for all of these independent variables.

At the inception of the project, the intention was to develop the technique inside of an existing game engine, namely Unreal Engine 5 (UE5); one of the key draws was the complete rendering feature set. While development time in UE5 proved to be prohibitive, the lack of many of these rendering features has been detrimental. Many timing results have been obfuscated by ISBE memory contention, an issue that would be solved with frustrum culling and mesh LOD considerations. As it was, these did not fit within development time; implementing a more feature-complete renderer would be very beneficial for clarifying the timings of the techniques developed in this paper.

A key metric that should be changed in future work is the targeted resolution; the literature examined for comparisons to existing implementations shows a pattern of improving checkerboarding results as the targeted resolution increases, with the ‘best’ interlaced technique being used in 4K rendering. Targeting 720p, a lower resolution than any existing checkerboarding literature, is an insufficient test, both for timings (a larger target can be expected to take longer to reconstruct) and the visual quality ratings.

The final goal of the project was to create a stencil-based checkerboard implementation of similar or better quality and speed than an interlaced technique. This mostly failed; the visual quality of the scene appears to be poor, based on participant feedback. The timings collected imply that the techniques implemented are of a good speed, but inferior to the gains of some interlaced techniques. Regardless, timing comparisons are largely invalidated by the poor quality of the output.

The project objectives somewhat changed throughout; at the beginning, the idea was to make an adaptive form of checkerboarding, but this quickly proved to be a dead end, as research continued. The time to calculate adaptivity information would almost certainly have been prohibitive. From this, it transpired that stencil-based checkerboarding was relatively under-documented; there was a good body of research on interlaced techniques, but a comparatively small body on stencil-based techniques. Although the only existing research on the stencil-based approach was critical, those criticisms appeared quite weak, or unjustified.

Perhaps this was a mistake; the research in question comes from Frostbite Labs, from their experience in developing Battlefield 1, so they definitely aren't novices in the field. While no evidence of the 'dilated color bits' (Wihlidal, 2020) was found, it appears that other criticisms, such as insufficient sampleable data in a stencil-based target, were correct. There may be a use to verifying and quantifying this, but the project could have been better spent investigating newer, cutting-edge techniques, rather than an alternate method of a near decade-old approach. (Mansouri, 2016)

The design approach was potentially problematic; observing graphical problems and addressing them until the picture appeared clear did help to avoid feature (and thus time) bloat. However, this clearly did not catch all imperfections; many participants noted graininess and flickeriness, something not observed during the design. While image analysis techniques as discussed in Section 4.3 are not ideal for judging a final product, they could have perhaps been good metrics to use during design. Integrating a PSNR test into the development process may have caught the visual noise causing the graininess effect.

The design suffered from the lack of a good set endpoint to the process, but this was somewhat unavoidable. The nature of graphics is often that 'good enough is good enough,' especially where optimisation is concerned. Laying out a minimum feature set, such as a checkerboarded depth target and motion reprojection did help to provide some structure, but finding an ideal point to stop development is difficult if not laid out explicitly in the plan. An attempt was made, with development stopping when the picture looked okay, but as mentioned above, this was insufficient. Anything different in future work is still not recommended; laying out a full pipeline will still suffer from the same problem of having no good framework for when to stop development, but potentially lead to work bloat, including operations with no good use. In this implementation, it appears that development was stopped too early, but its equally problematic if a solution is overworked, as it will hamper the efficiency gains.

To conclude, the project has suffered from some key oversights that have hampered the final test, from missing features, to an insufficiently structured design. Plenty of potential future work has been hinted at throughout the paper, and some is further explored in Section 8. To ensure fair testing, future work should target a higher resolution, with appropriate optimisation of vertex pipelines (to avoid ISBE bottlenecks), and with a reworked design process, defining a more suitable clear endpoint for development. Memory disparities can be simply addressed, and in future work, time spent pursuing implementations in commercial engines can be avoided.

8 Conclusions and Future Work

While results are inconclusive, there are indications that stencil-based checkerboarded rendering is inferior to interlaced approaches. The necessary 2x2 block layout creates large regions without sampleable data, making inferences about pixel data difficult. Human participants indicate that stencil-based checkerboard effects create videos of inferior quality than a standard rendering, in comparison all commercial implementations in literature, that report no complaints from customers (Mansouri, 2016) (de Carpentier & Ishiyama, 2017).

Our timing comparisons may indicate a more efficient checkerboard reconstruction than some commercial interlaced implementations, but without the same visual quality preservation, this holds little weight. We observe that there may be a correlation between targeted resolution and the visual quality of the reconstructed scene, a correlation followed by our implementation.

Future work could address the correlation between targeted resolution and visual quality, either within interlaced or stencil-based techniques. Future work may also look into a direct comparison between interlaced and stencil-based techniques (rather than the indirect comparison via standard rendering in this paper), and may explore new techniques involving the motion buffer depth target, a unique advantage afforded by stencil-based techniques, that can be used to create a native resolution full-screen depth target, replacing or working in tandem with depth history buffers.

Bibliography

Amann, J., Weber, B., & Wuthrich, C. A. (2013). Using Image Quality Assessment to Test Rendering Algorithms. *WSCG 2013: Full Papers Proceedings: 21st International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision in cooperation with EUROGRAPHICS Association* (pp. 205-214). Vaclav Skala - UNION Agency.

Anderson, M., Motta, R., Chandrasekar, S., & Stokes, M. (1996). *Proposal for a Standard Default Color Space for the Internet - sRGB.* Microsoft and HP.

Blewitt, W., Ushaw, G., & Morgan, G. (2013). Applicability of GPGPU Computing to Real-Time AI Solutions in Games. *IEEE Transactions on Computational Intelligence and AI in Games*, 265-275.

Bondarev, V., & Niverty, S. (2021, October 22). *Advanced API Performance: Async Compute and Overlap.* Retrieved from NVIDIA: https://developer.nvidia.com/blog/advanced-api-performance-async-compute-and-overlap/

Carpentier, G. d., & Ishiyama, K. (2017, July 31). *Decima Engine: Advances in Lighting and AA.* Retrieved from Guerilla Games: https://www.guerrilla-games.com/read/decima-engine-advances-in-lighting-and-aa

Choi, Y., Park, S., & Cha, H. (2019). Graphics-aware Power Governing for Mobile Devices. *MobiSys '19: Proceedings of the 17th Annual International Conference on Mobile Systems, Applications and Services* (pp. 496-481). MOBISYS.

Christensen, P. H., Fong, J., Laur, D. M., & Batali, D. (2006). Ray Tracing for the Movie 'Cars'. *2006 IEEE Symposium on Interactive Ray Tracing.* Salt Lake City, UT: IEEE.

Claypool, M., & Claypool, K. (2009). Perspectives, frame rates and resolutions: it's all in the game. *FDG '09: Proceedings of the 4th International Conference on Foundations of Digital Games* (pp. 42-49). ACM.

Crytek. (2023). *CryEngine Documentation*. Retrieved from CryEngine: https://docs.cryengine.com/

de Carpentier, G., & Ishiyama, K. (2017). Decima: Adavances in Lighting and AA. *SIGGGRAPH 2017.*

Depth Precision Visualized. (2023). Retrieved from NVIDIA Developer.

Epic Games. (2023). *Motion Blur.* Retrieved from Unreal Engine Docs: https://docs.unrealengine.com/4.27/en-US/Resources/ContentExamples/PostProcessing/1_12/

Even, G., & Seidel, P.-M. (2000). A comparison of three rounding algorithms for IEEE floating-point multiplication. *IEEE Transactions on Computers*, 638-650.

Google. (2023). *Google Forms*. Retrieved from Google Workspaces: https://www.google.co.uk/forms/about/

Gregory, J. (2018). *Game Engine Architecture.* CRC Press.

Gronowski, P. E., Bowhill, W. J., Preston, R. P., Gowan, M. K., & Allmon, R. L. (1998). High-Performance Microprocessor Design. *IEEE Journal of Solid-State Circuits*, 676-686.

Guenter, B., Finch, M., Drucker, S., Tan, D., & Snyder, J. (2012). *Foveated 3D Graphics.* Microsoft Graphics.

Gunadi, S. I., & Yugopuspito, P. (2018). Real-Time GPU-based SPH Fluid Simulation Using Vulkan and OpenGL Compute Shaders. *2018 4th International Conference on Science and Technology.* Yogyakarta: IEEE.

Hakura, Z. S., Kilgariff, E. M., Shebanow, M. C., Bowman, J. C., Johnson, P. B., Rhoades, J. S., . . . Corp, N. (2008). *US Patent No. 8947444B1.*

Han, J. (2011). *3D Graphics for Game Programming.* Chapman and Hall.

Harada, T., McKee, J., & Yang, J. C. (2012). Forward+: Bringing Deferred Lighting to the Next Level. *EUROGRAPHICS 2012.* Eurographics Digital Library.

He, Y., Gu, Y., & Fatahalian, K. (2014). Extending the graphics pipeline withadaptive, multi-rate shading. *ACM Transactions on Graphics, Volume 33, Issue 4*, 1-12.

Huang, Q., Huang, Z., Werstein, P., & Purvis, M. (2008). GPU As a General Purpouse Computing Resource. *2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*.

Intel. (2023). *GPU Samples Library*. Retrieved from Intel: https://www.intel.com/content/www/us/en/developer/topic-technology/graphics-research/samples.html

Intel. (2023). *Intel Graphics Performance Analyzers (Intel GPA) Release Notes*. Retrieved from Intel: https://www.intel.com/content/www/us/en/developer/articles/release-notes/intel-gpa-current-release-notes.html

Ito, J. (2007, May 12). *Spacewar! running on the Computer History Museum's PDP-1.* Retrieved from Wikipedia: https://commons.wikimedia.org/w/index.php?curid=2099696

Karis, B., Stubbe, R., & Wihlidal, G. (2021, October 30). A Deep Dive into Nanite Virtualized Geometry. *SIGGRAPH Advances in Real-Time Rendering*. YouTube.

Klint, J. (2008). *Deferred Rendering in Ledweks Engine.*

Korhonen, J., & You, J. (2012). Peak Signal-to-Noise Ratio Revisited: Is Simple Beautiful? *2012 Fourth International Workshop on Quality of Multimedia Experience* (pp. 37-38). Melbourne: IEEE.

Labrador, C. A. (2018). *Improved Sampling for Temporal Anti-Aliasing.* Lund: Lund University.

Laine, S., & Karras, T. (2011). High-performance software rasterization on GPUs. *HPG ;11: Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics* (pp. 79-88). ACM SIGGRAPH.

Lengyel, E. (2005). *Oblique View Frustum Depth Projection and Clipping* . Retrieved from Terathon: https://www.terathon.com/lengyel/Lengyel-Oblique.pdf

Levenberg, J. (2002). Fast view-dependent level-of-detail rendering using cached geometry. *IEEE Visualization.* Boston, MA: IEEE.

Lippens, P., Nagasamy, V., & Wolf, W. (1995). CAD challenges in multimedia computing. *Proceeding of IEEE International Conference on Computer Aided Design (ICCAD)* (pp. 502-508). San Jose : IEEE.

Liu, N., & Pang, M.-Y. (2009). A Survey of Shadow Rendering Algorithms: Projection Shadows and Shadow Volumes. *2009 Second International Workshop on Computer Science and Engineering.* Qingdao: IEEE.

Macedo, D. V., & Rodrigues, M. A. (2018). Real-time dynamic reflections for realistic rendering of 3D scenes. *The Visual Computer*.

Machover, C. (1978, November). *A Brief, Personal History of Computer Graphics.* Retrieved from UIC: https://www.evl.uic.edu/datsoupi/502/14_mach.pdf

Malvar, H., & Sullivan, G. (2003). *YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range.* Trondheim: Microsoft.

Mansouri, J. E. (2016). *Rendering 'Rainbow Six | Siege'.* Retrieved from GDC Vault: https://www.gdcvault.com/play/1022990/Rendering-Rainbow-Six-Siege

Microsoft. (2021, 11 23). *ID3D11DeviceContext::OMSetRenderTargets method (d3d11.h)*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/windows/win32/api/d3d11/nf-d3d11-id3d11devicecontext-omsetrendertargets

Microsoft. (2022, 12 08). *Hardware Support for Direct3D 12.1 Formats*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/windows/win32/direct3ddxgi/hardware-support-for-direct3d-12-1-formats

Microsoft. (2022, March 16). *Shader Model 1*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl-sm1

Microsoft. (2023). *PIX on Windows*. Retrieved from Microsoft Devblogs: https://devblogs.microsoft.com/pix/download/

Microsoft. (2023, 03 02). *Variable-rate Shading (VRS)*. Retrieved from Learn Microsoft: https://learn.microsoft.com/en-us/windows/win32/direct3d12/vrs

Mitchell, D. P., & Netravali, A. N. (1988). Reconstruction filters in computer-graphics. *ACM SIGGRAPH Computer Graphics, Volume 22, Issue 4*, 221-228.

Mommersteeg, V. (2015). *The differences between Deferred and Forward Rendering.* NHTV University of Applied Sciences.

Nardone, V., Muse, B., Abidi, M., Khomh, F., & Penta, M. D. (2023). Video Game Bad Smells: What They Are and How Developers Percieve Them. *ACM Transactions on Software Engineering and Methodology, Volume 32, Issue 4*, 1-35.

Natalie, @. J. (2022, 05 30). *DirectX Discord Server.* Retrieved from Discord: https://discord.com/channels/590611987420020747/590965902564917258/980867372

Nehme, Y., Farrugia, J.-P., Dupont, F., & Lavoue, P. L. (2019). Comparison of subjective methods, with and without explicit reference, for quality assessment of 3D graphics. *SAP '19: ACM Symposium on Applied Perception* (pp. 1-9). ACM.

NVIDIA. (2018). *NVIDIA Turing Architecture Whitepaper*. Retrieved from images.nvidia.com: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

NVIDIA. (2019). *NVIDIA Adaptive Shading Overview.* Retrieved from http://www.leiy.cc/publications/nas/nas-gdc19.pdf

NVIDIA. (2023, November 5). *VRWorks - Lens Matched Shading*. Retrieved from NVIDIA Developer: https://developer.nvidia.com/vrworks/graphics/lensmatchedshading

NVIDIA. (2023, November 5). *VRWorks - Multi-Res Shading*. Retrieved from NVIDIA Developer: https://developer.nvidia.com/vrworks/graphics/multiresshading

NVIDIA NSight. (2023). *Advanced Learning*. Retrieved from NVIDIA NSight Graphics: https://docs.nvidia.com/nsight-graphics/AdvancedLearning/index.html

Patidar, S., Bhattacharjee, S., Singh, J. M., & Narayanan, P. J. (2006). *Exploiting the Shader Model 4.0 Architecture.* Hyperabad: Center for Visual Information Techology, IIIT.

Peddie, D. J. (2021, February 25). *Famous Graphics Chips: Nvidia's GeForce 256*. Retrieved from IEEE Computer Society: https://www.computer.org/publications/tech-news/chasing-pixels/nvidias-geforce-256

Peddie, J. (2023). *The History of the GPU - Steps to Invention.* Springer Cham.

Pool, J. (2012). *Energy-Precision Tradeoffs in the Graphics Pipeline.* Chapel Hill: University of North Carolina at Chapel Hill.

Raman, R., & Wise, D. S. (2008). Converting to and from Dilated Integers. *IEEE Transactions on Computers*, 567-573.

Raskar, R., & Low, K.-L. (2002). Blending multiple views. *10th Pacific Conference on Computer Graphics and Applications.* Beijing: IEEE.

Rath, J. (2009, 12 22). *The Data-Crunching Powerhouse Behind Avatar*. Retrieved from DataCenter Knowledge: https://www.datacenterknowledge.com/archives/2009/12/22/the-data-crunching-powerhouse-behind-avatar

Rogers, T. G., O'Connor, M., & Aamodt, T. M. (2013). Divergence-aware warp scheduling. *Proccedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture* (pp. 99-110). ACM.

Sacranie, J. (2010). *Consumer Perceptions & Video Game Sales: A Meeting of the Minds.* Illinois Wesleyan University.

Salomon, D. (2011). *The Computer Graphics Manual.* London: Springer-Verlag.

Sanders, J., & Kandrot, E. (2011). *CUDA by Example: An Introduction to General-Purpose GPU Programming.* NJ: Addison-Wesley.

Sheikh, H. R., Sabir, M. F., & Bovik, A. C. (2006). A Statistical Evaluation of Recent Full Reference Image Quality Assessment Algorithms. *IEEE Transactions on Image Quality*, 3440-3451.

Steam. (2023, May 1). *Steam Hardware Survey*. Retrieved from Steam: https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam

Thaler, J. (2011). *Deferred Rendering.* TU Wien.

Thinakaran, P., Raj, J., Sharma, B., Kandemir, M. T., & Das, C. R. (2018). The Curious Case of Container Orchestration and Scheduling in GPU-based Datacenters. *SoCC '18 Proceedings of the ACM Symposium on Cloud Computing* (p. 524). ACM.

Unity Technologies. (2023). *Post Processing Effects: Motion Blur.* Retrieved from Unity Learn: https://learn.unity.com/tutorial/post-processing-effects-motion-blur-2019-3

Vaidyanathan, K., Salvi, M., Toth , R., Foley, T., Akenine-Moller, T., Nilsson, J., . . . Lefohn, A. (2014). *Coarse Pixel Shading.* The Eurographics Association.

Vatjus-Anttila, J., Koskela, T., Lappalainen, T., & Hakkila, J. (2016). Simplification of 3D Graphics for Mobile Devices: Exploring the Trade-off Between Energy Savings and User Perceptions of Visual Quality. *3D Res*.

Wang, L., Huang, Y.-z., Chen, X., & Zhang, C.-y. (2008). Task Scheduling of Parallel Processing in CPU-GPU Collaborative Environment. *International Conference on Computer Science and Information Technology* (pp. 228-232). IEEE.

Wihlidal, G. (2020, March 31). *4K Checkerboard in Battlefield 1 and Mass Effect.* Retrieved from EA: https://www.ea.com/frostbite/news/4k-checkerboard-in-battlefield-1-and-mass-effect-andromeda

Wronski, B. (2016). Volumetric Fog and Lighting. In W. Engel, *GPU Pro 6: Advanced Rendering Techniques.* CRC Press.

Yang, L., Zhdan, D., Kilgariff, E., Lum, E. B., Zhang, Y., Johnson , M., & Rydgard, H. (2019). Visually Lossless Content and Motion Adaptive Shading in Games. *ACM Computer Graphics Interactive Technology, Vol. 2, No. 1*, 6:1-19.

Appendix A – Project Specification

MComp Individual Project: Project Specification

Student name	Eve Johnson Powell	
Student contact details	**SHU Email**	b9009554@my.shu.ac.uk
	Telephone	07544589729
Course *(Delete as appropriate)*	MComp Computer Science for Games	
Supervisor name *(if known)*	Mark Featherstone	
Title of project *(provisional)*	An investigation into Regionally Adaptive Shading	
Date	28/09/2023	

Research Question

Can the established technique of checkerboarded rendering be combined with cutting-edge temporally adaptive techniques to decrease frame GPU time?

Elaboration

The primary cost of rendering lies in the pixel shader; not only do they often do more complex work than shaders of other varieties, but the invocation overhead is disproportionately high. As such, a wide variety of techniques serve to dynamically determine acceptable reductions in pixel shader invocations.

Checkerboarded rendering is a technique that subdivides a render target into 2x2 pixel regions. Then, imagining these regions arranged as a checkerboard, will render only the black squares and white squares on alternating frames. The rendered results for the current frame are composited with the previous frame before being displayed. There are a variety of composition methods, with some more sophisticated implementations altering the results of the previous frame slightly, based on their predicted changes between frames. With only a small amount of overhead, the number of pixel shader invocations can be halved.

Temporally adaptive shading is a technique to reduce pixel shader invocations based on predicted changes in the resultant colour. After a few frames of full screen rendering, a shading gradient is created for each pixel, based on the rate of change of its colour. Based on this, a deadline for reshading is created, before which an extrapolation from the previously calculated shading gradient is used. This can produce extreme reductions in GPU frame time.

Research into temporally adaptive shading is sparse, and all existing literature has encountered issues with handling shadows; they often see colour corruption around the edges, which is handled by forcing reshades of shadows, and many of their surrounding pixels. Examining these scenes, most areas marked for reshading are large square regions, caused by overprediction.

With most reshading happening in squares, it follows that there could be a benefit to pairing temporally adaptive shading with another technique that handles efficient shading of square regions, i.e. checkerboarded rendering. By adapting core checkerboarded rendering algorithms to be 'temporally aware', I aim to create a new form of adaptive shading, which I will speculatively call Regionally Adaptive Shading.

Project objectives

By implementing multiple forms of checkerboarded rendering, we can measure the impact of temporally adaptive elements on resultant GPU times. These will be

- A naïve form of checkerboarding that does very limited work at the composition stage
- A naïve form of checkerboarding with a more complex and visually pleasing composition stage
- A new form of checkerboarding with temporally adaptive elements at the composition and/or pre-shading stage

Loading well known test scenes such as Sponza and Robot Factory into the engine will allow us to take frame time measurements, and other useful metrics such as the number of pixer shader invocations per frame. By taking measurements with a variety of scenes in a variety of differing render modes, it can be demonstrated whether or not there is any performance to be gained.

Project deliverable(s)

This will be a modified version of Unreal Engine 5, given as the source code. All additions will be marked by files with an EP_ prefix, or where existing files are altered, with appropriate macro surroundings. This cannot be done in a plugin, as the shading stages being introduced are within the pre-frame and post frame sections, which must be changed via modifications to the source code.

Selection of an appropriate macro value will compile this source code to create a renderer with one of the specified checkerboarding modes, or the default rendering mode. For accurate efficiency measurements, runtime changing of the shading mode will not be supported.

Ethics

While there are no strong ethical considerations, human participants may be used to compare the visual quality of scenes. This will not require strenuous activity, and will not put anybody at risk of harm.

Action plan

Below are the milestone delivery dates for various parts of this application.

Wednesday 11th October

An implementation of the naïve checkerboarding element of the engine. This will not include any composition functionality.

Wednesday 18th October

An alternative implementation of the naïve checkerboarding will be implemented. This will include a more visually pleasing composition function. The basic composition functionality of the existing implementation will be complete.

Wednesday 1st November

A first pass approach at the temporally adaptive algorithm will be complete. This will be somewhat temporally aware but may not be fully optimised.

Wednesday 15th November

A significant portion of the literature review will be complete.

Wednesday 22nd November

The algorithm will be in its final stage. Writing can commence in earnest. There are no marked milestones from here because its all a bit blurry.

Appendix B – Ethics Checklist

UREC2 RESEARCH ETHICS PROFORMA FOR STUDENTS UNDERTAKING LOW RISK PROJECTS WITH HUMAN PARTICIPANTS

This form is designed to help students and their supervisors to complete an ethical scrutiny of proposed research. The University Research Ethics Policy should be consulted before completing the form. The initial questions are there to check that completion of the UREC 2 is appropriate for this study. The final responsibility for ensuring that ethical research practices are followed rests with the supervisor for student research.

Note that students and staff are responsible for making suitable arrangements to ensure compliance with the General Data Protection Act (GDPR). This involves informing participants about the legal basis for the research, including a link to the University research data privacy statement and providing details of who to complain to if participants have issues about how their data was handled or how they were treated (full details in module handbooks). In addition the act requires data to be kept securely and the identity of participants to be anonymized. They are also responsible for following SHU guidelines about data encryption and research data management. Information on the Ethics Website

The form also enables the University and College to keep a record confirming that research conducted has been subjected to ethical scrutiny.

The form may be completed by the student and the supervisor and/or module leader (as applicable). In all cases, it should be counter-signed by the supervisor and/or module leader, and kept as a record showing that ethical scrutiny has occurred. Some courses may require additional scrutiny. Students should retain a copy for inclusion in their research projects, and a copy should be uploaded to the relevant module Blackboard site.

Please note that it may be necessary to conduct a health and safety risk assessment for the proposed research. Further information can be obtained from the College Health and Safety Service.

Checklist Questions to ensure that this is the correct form

1. Health Related Research with the NHS or Her Majesty's Prison and Probation Service (HMPPS)or with participants unable to provide informed consent

Question	Yes/No
1. Does the research involve? • Patients recruited because of their past or present use of the NHS	No
• Relatives/carers of patients recruited because of their past or present use of the NHS	No
• Access to data, organs or other bodily material of past or present NHS patients	No
• Foetal material and IVF involving NHS patients	No
• The recently dead in NHS premises	No
• Prisoners or others within the criminal justice system recruited for health-related research*	No
• Police, court officials, prisoners or others within the criminal justice system*	No
• Participants who are unable to provide informed consent due to their incapacity even if the project is not health related	No
2. Is this a research project as opposed to service evaluation or audit? *For NHS definitions of research etc. please see the following website* http://www.hra.nhs.uk/documents/2013/09/defining-research.pdf	No

If you have answered **YES** to questions **1 & 2** then you **MUST** seek the appropriate external approvals from the NHS, Her Majesty's Prison and Probation Service (HMPPS) under their independent Research Governance schemes. Further information is provided below.
https://www.myresearchproject.org.uk

NB College Teaching Programme Research Ethics Committees (CTPRECS) provide Independent Scientific Review for NHS or HMPPS research and initial scrutiny for ethics applications as required for university sponsorship of the research. Applicants can use the IRAS proforma and submit this initially to their CTPREC.

1. Checks for Research with Human Participants

Question	Yes/No
1. Will any of the participants be vulnerable? *Note: Vulnerable' people include children and young people, people with learning disabilities, people who may be limited by age or sickness, people researched because of a condition they have, etc. See full definition on ethics website*	No
2. Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants or will the study involve invasive, intrusive or potentially harmful procedures of any kind?	No
3. Will tissue samples (including blood) be obtained from participants?	No
4. Is pain or more than mild discomfort likely to result from the study?	No
5. Will the study involve prolonged or repetitive testing?	No
6. Is there any reasonable and foreseeable risk of physical or emotional harm to any of the participants? *Note: Harm may be caused by distressing or intrusive interview questions, uncomfortable procedures involving the participant, invasion of privacy, topics relating to highly personal information, topics relating to illegal activity, or topics that are anxiety provoking, etc.*	No
7. Will anyone be taking part without giving their informed consent?	No
8. Is it covert research? *Note: 'Covert research' refers to research that is conducted without the knowledge of participants.*	Np
9. Will the research output allow identification of any individual who has not given their express consent to be identified?	No

If you have answered **YES** to any of these questions you are **REQUIRED** to complete and submit a UREC 3 or UREC4). Your supervisor will advise. If you have answered **NO** to all these questions then proceed with this form (UREC 2).

General Details

Name of student	Eve Johnson Powell
SHU email address	B9009554@my.shu.ac.uk

Course or qualification (student)	MComp Computer Science For Games
Name of supervisor	Mark Featherstone
email address	
Title of proposed research	Regionally Adaptive Shading
Proposed start date	18/09/2023
Proposed end date	05/01/2023
Background to the study and scientific rationale for undertaking it.	Checkerboarding is a well-established rendering acceleration technique, optimized for square regions. Temporally adaptive shading is a very new technique, that suffers from issues with overprediction, leading to a lot of shading taking the form of square regions. A combination of the two approaches is hypothesized to reduce GPU workload, with applications primarily in games and other real time rendering applications. As with almost all forms of rendering acceleration, the final visual quality is an important metric to test for, and is best tested by human participants. As such, alongside the numerical data relating to efficiency, human participants will be used to evaluate how similar rendered scenes, but using different techniques, appear to them.
Aims & research question(s)	To evaluate a novel regionally adaptive shading technique, combining checkerboarded rendering with temporally adaptive shading. Can temporally adaptive shading principles reduce GPU workload when applied to traditional checkerboarded rendering methods?
thods to be used for: 1.recruitment of participants, 2.data collection,	The only condition for participation is that someone can see; as the nature of the experiment relates solely to visual quality this is of the utmost importance. This will not limit those who use glasses or are otherwise visually impaired; users will be asked to self-identify as to whether they believe their vision to be

3. data analysis.	appropriately strong. Users will be shown a number of short videos, across different scenes, with different rendering techniques applied. They will then be asked to rate the perceived visual quality of the video on a numerical scale. This can then be analysed by comparing the average ratings of different scenes and techniques.
Outline the nature of the data held, details of anonymisation, storage and disposal procedures as required.	Data does not need to be analysed with any respect to the participant(s) from which it was gathered. Using a commercial system such as Google Forms allows easy deletion of a given users data upon request. Data held will be purely numerical and non-identifying, with the exception of that required to facilitate the deletion of data upon request.

3. Research in Organisations

Question	**Yes/No**
1. Will the research involve working with/within an organisation (e.g. school, business, charity, museum, government department, international agency, etc.)?	No
2. If you answered YES to question 1, do you have granted access to conduct the research? *If YES, students please show evidence to your supervisor. PI should retain safely.*	N/A
3. If you answered NO to question 2, is it because: A. you have not yet asked B. you have asked and not yet received an answer C. you have asked and been refused access. *Note: You will only be able to start the research when you have been granted access.*	N/A

4. Research with Products and Artefacts

Question	**Yes/No**
1. Will the research involve working with copyrighted documents, films, broadcasts, photographs, artworks, designs, products, programmes, databases, networks, processes, existing datasets or secure data?	Y
2. If you answered YES to question 1, are the materials you intend to use in the public domain? *Notes: 'In the public domain' does not mean the same thing as 'publicly accessible'.* • *Information which is 'in the public domain' is no longer protected by copyright (i.e. copyright has either expired or been waived) and can be used without permission.* • *Information which is 'publicly accessible' (e.g. TV broadcasts, websites, artworks, newspapers) is available for anyone to consult/view. It is still protected by copyright even if there is no copyright notice. In UK law, copyright protection is automatic and does not require a copyright statement, although it is always good practice to provide one. It is necessary to check the terms and conditions of use to find out exactly how the material may be reused etc.* *If you answered YES to question 1, be aware that you may need to consider other ethics codes. For example, when conducting Internet research, consult the code of the Association of Internet Researchers; for educational research, consult the Code of Ethics of the British Educational Research Association.*	Y The research will involve alteration of Epic Games' Unreal Engine 5, which is publicly available for download and alteration.
3. If you answered NO to question 2, do you have explicit permission to use these materials as data? *If YES, please show evidence to your supervisor.*	N/A
4. If you answered NO to question 3, is it because: A. you have not yet asked permission B. you have asked and not yet received and answer C. you have asked and been refused access.	**N/A**

Note You will only be able to start the research when you have been granted permission to use the specified material.	

Adherence to SHU policy and procedures

Personal statement	
I can confirm that: • I have read the Sheffield Hallam University Research Ethics Policy and Procedures • I agree to abide by its principles.	
Student	
Name: Eve Johnson Powell	Date: 03/10/2023
Signature: EJP	
Supervisor or other person giving ethical sign-off	
I can confirm that completion of this form has not identified the need for ethical approval by the FREC or an NHS, Social Care or other external REC. The research will not commence until any approvals required under Sections 3 & 4 have been received and any necessary health and safety measures are in place.	
Name:	Date:
Signature:	
Additional Signature if required by course:	
Name:	Date:
Signature:	

Please ensure the following are included with this form if applicable, tick box to indicate:

	Yes	No	N/A
Research proposal if prepared previously	X	☐	☐
Any recruitment materials (e.g. posters, letters, etc.)	☐	☐	X
Participant information sheet	X	☐	☐
Participant consent form	X	☐	☐
Details of measures to be used (e.g. questionnaires, etc.)	X	☐	☐
Outline interview schedule / focus group schedule	☐	☐	X
Debriefing materials	☐	☐	X
Health and Safety Project Safety Plan for Procedures	☐	☐	X

Appendix C – Glossary

Rendering Hardware and Core Operations

Rasterization – The conversion of geometric primitives into a set of covered pixels.

Implicit Rasterization – Rasterization that occurs as an automatic part of the graphics pipeline before the pixel shader.

Bottleneck – The area in which the demand exceeds the capacity; for example, a GPU bottleneck means that the GPU is still going while the CPU has run out of work.

Alpha blending – Combination of two RGB colours with respect to the alpha channel; this is a programmable automatic operation in the graphics pipeline.

Triangle culling – Discarding triangles from the graphics pipeline, often if their constituent points are arranged anti-clockwise. This is a programmable automatic operation in the graphics pipeline.

Shader – A small program or block of code specifically for transformations, lighting, or other rendering operations.

Vertex Shader – A shader specifically for the translation of vertices, such as from world space to screen space. Mandatory part of the graphics pipeline.

Pixel Shader – A shader specifically for colouring an individual pixel, perhaps including lighting. Optional but frequently used part of the graphics pipeline.

Geometry Shader – A shader for the dynamic creation or destruction of vertices or primitives. Optional part of the graphics pipeline.

Compute Shader – A shader that does not use the graphics pipeline, allowing arbitrary GPU based computation.

Render Target – The buffer to which rendered pixels are written. Most APIs allow multiple to be bound at once, for different relevant outputs.

Depth Target – A buffer storing depth values, calculated from the Z position calculated in the vertex shader, most often representing the distance of the nearest objects to the camera.

Depth Testing – A per-pixel test of its depth value against the value already in the depth target, leading either to the pixel being shaded or discarded. This is a programmable automatic operation in the graphics pipeline.

Depth Pre-Pass – A dedicated rendering pass for generating a depth image.

CPU – The central processing unit; executes commands in sequence.

GPU – The graphics programming unit; executes identical commands in parallel.

GPGPU – General Purpose GPU; the use of GPUs for work that does not fit into the traditional graphics pipeline.

Multi-Resolution Shading – A technique introduced with NVIDIA's Maxwell architecture; reduce shading quality at the edge of the screen, as this area is already warped to fit into VR lenses. (NVIDIA, 2023)

Lens-Matched Shading – An improvement on Multi-Resolution Shading that uses a render surface much closer in shape to the final lens, but still maintains a higher resolution nearer the centre. (NVIDIA, 2023)

Rendering Techniques

Nanite – Unreal Engine 5's mesh rendering system for the automatic separation and resizing of mesh data, to dynamically limit the rendering of unnecessary detail.

CUDA – Compute Unified Device Architecture; NVIDIA's proprietary tool for GPGPU programming, with an accessible C-style interface.

Forward+ - Forward rendering (a single pass that fully renders geometry), with an added light culling step to minimise data transfer and processing time.

Light culling – A pre-rendering step of removing lights from consideration for certain geometry if they are too distant to be relevant or are obscured by other geometry.

Coarse Pixel – A logical pixel, which can be made up of one or more traditional pixels.

Other Terms

SQL – Structured Query Language, a popular language for querying and updating databases.

Finite State Machine – A behavioural mechanism, comprised of a set of states (behaviours), linked by transitions (events), that define the conditions for changes in behaviour.

AI – Artificial Intelligence; in games this often refers to simulated opponents and has little in common with sophisticated techniques like Large Language Models.

Appendix D – Performance Metrics Tables

	Checked Reconstruction
Avg Execution Time	111.30us 139.40us 118.56us 117.46us 114.64us
Compute Throughput (synchronous)	97%
Compute Throughput (async)	0%
Compute Register Allocation	85%
Compute Shader Threads Launched	1,904,640
Register Allocation Stalls	15%
Warp Allocation Stalls	15%
Compute Warps Launched	59,520
Back Buffer Pixels per warp	15.48 (exp. 16)
Compute Shader Invocations	61,440
Back Buffer Pixels per invocation	15
Instructions Executed	2,976,000

Table 5 – The metrics taken from the reconstitution compute shader.

	Checked	Non-Checked
Average Frame Time (ms)	49.6ms	50.4ms
CPU Time Presenting Swap Chain (%)	76.4%	91.0%
Avg Deferred Mesh Draw Time	300.51us 300.23us 267.29us 292.50us 291.06us	294.37us 292.10us 323.04us 309.85us 313.55us
Avg Motion Pass Time	136.79us 133.59us 127.41us 135.29us 132.63	134.39us 139.19us 138.38us 142.71us 133.80us
Avg Pre-Frame SetupMain Time		44.50us

Avg BlitGBuffer Time	32.92us	~
Avg Gamma Time		52.51us
Deferred Draw Mesh Stats (Starting View)		
VS Invocations	4,185,983	4,185,983
PS Invocations	479,923	959,773
Samples Rejected (DepthOcc)	1,101,438	584,216
Samples Rendered (DepthOcc)	517,259	1,034,481
Clipper Primitives In	3,738,168	3,738,168
Clipper Primitives Out	1,122,640	1,122,640
GPU Activity	100%	100%
Front End Stalled	30%	29%
Early Z Pixels Killed	673,408	102,656
Early Z Samples Input	1,153,616	1,045,106
Early Z Samples Killed	674,226	66,829
Coarse Resolution Depth Testing Samples Input	17,384,960	17,384,960
Coarse Resolution Depth Testing Samples Rejected	6,155,264	4,989,952
EOP to EOP Duration	39,278,592	39,621,632
TOP to EOP Duration		39,654,976
Pixel Shader Activity	0%	0%
Vertex Shader Activity	0%	0%
Warp Instructions Issued	15,658,686	18,833,738
Warp Instructions Executed by Vertex Shader	11,247,052	11,247,052
Warp Instructions Executed by Pixel Shader	4,362,540	7,540,260
Warp Instructions Executed	15,612,252	18,797,672
Pixel Shader Warps Activity	0%	0%
Unused Warp Slots on Active SM	9%	9%
Unused Warp Slots on Idle SM	90%	89%
Vertex Warps Active	0%	0%
Pixel Shader Warps Launched	31,072	53,953
Vertex Shader Warps Launched	135,745	135,745
Warps Launched	166,757	189,637
Vertex ISBE Alloc Throughput	94%	94%

Vertex Shader Thread Launched	4,185,983	4,185,983
Pixel Shader Thread Launched	718,868	1,437,308

Table 6 – The metrics taken from the Deferred Mesh Drawing GPU region, encompassing the rendering of mesh data to a multi-target G-buffer.

Appendix E – Human Participant Form

Checkerboarded Rendering Study

The next page comprises the participant information sheet; this will inform you of everything you can expect from this study and your participation in it. The page after will contain the participant consent form. Please ensure you read and understand the participant information sheet fully before answering the questions in the participant consent form.

Please ensure that this study is completed on a tablet, laptop, or desktop computer. Do not proceed if using a phone.

Your Google login has been used to ensure that responses to the study are limited to one per person; this information will be deleted once the study concludes.

* Indicates required question

1. Email *

Participant Information Sheet

Participant Research Information

An Evaluation of Depth-Based Checkerboarding

You have been invited to participate in this study to evaluate the visual quality of a pair of videos, with the aim of testing the quality of various graphical techniques. This is part of a wider effort to improve the quality and speed of real-time rendering, with applications in gaming and simulations.

Legal Basis for Research for Studies

The university undertakes research as part of its function for the community under its legal status. Data protection allows us to use personal data for research with appropriate safeguards in place under the legal basis of public tasks that are in the public interest. A full statement of your rights can be found at https://www.shu.ac.uk/about-this-website/privacy-policy/privacy-notices/privacy-notice-for-research. However, all University research is reviewed to endure that participants are treated appropriately and their rights respected. This study was approved by UREC. Further information at https://www.shu.ac.uk/research/ethics-integrity-and-practice.

Why have you asked me to take part?

This study is open to anyone who believes they have the visual ability to judge the quality of an image. There are no selection criteria beyond this.

Do I have to take part?

It is up to you to decide if you want to take part. A copy of the information provided here is yours to keep, along with the consent form if you do decide to take part. You can still decide to withdraw at any time without giving a reason, or you can decide not to answer a particular question.

What will I be required to do?

You will be required to view a pair of videos. After each video has been viewed, you will be asked to rate them on a numerical scale ranging from “very poor quality” to “very good quality”. This will happen 2 times, for a total of 2 different videos. Each video will be no more than 60 seconds, and the experiment is expected to take no more than 10 minutes.

The videos will come from Google Drive, and should be downloaded before viewing, to avoid the potential effects of compression.

Where will this take place?

The experiment will take place via the internet, on Google Forms. You will be required to follow links to download videos from Google Drive.

How often will I have to take part, and for how long?

This experiment only requires participation once. It is expected to take no more than 10 minutes.

Is deception involved in this study?

There is no deception involved in this study.

Are there any possible risks or disadvantages in taking part?

There are no identifiable risks in taking part.

Are there any possible benefits of taking part?

There are no identifiable benefits in taking part.

When will I have the opportunity to discuss my participation?

The researcher can be contacted by email at any time, at b9009554@my.shu.ac.uk.

Will anyone be able to connect me with what is recorded and reported?

Your responses are entirely confidential. Your stored information can be given to you upon request, and any data held on you deleted upon request.

Who will be responsible for all of the information when the study is over?

The researcher will be responsible for the information once the study is complete. Anonymised data may be included as an appendix with the final research, after which the originally recorded data will be deleted.

Who will have access to it?

Only the researcher will have access to the information recorded. Anonymised data may be included with the final research paper.

How will you use what you find out?

Data collected will only be used comparatively with other participants anonymised data. This will be used in the analysis of the efficacy of given real time rendering techniques. A full set of anonymised rating data will be included as an appendix with the final research.

How long is the whole study likely to last?

The testing phase of the project is expected to last no longer than 3 days.

How can I find out about the results of the study?

Contact the researcher if you wish to request a copy of the final study.

Researcher Details

Name: Eve Johnson Powell

Email: b9009554@my.shu.ac.uk

Participant Consent Form

Please answer the following questions carefully by ticking the response that applies.

2. I have read the information sheet for this study, and have had the details of the study explained to me. *

Mark only one oval.

- ◯ Yes
- ◯ No *Skip to section 13 (Thank you)*

3. My questions about the study have been answered to my satisfaction and I understand that I may ask further questions at any point. *

Mark only one oval.

- ◯ Yes
- ◯ No *Skip to section 13 (Thank you)*

4. I understand that I am free to withdraw from the study within the time limits outlined in the Information Sheet, without giving a reason for my withdrawal or to decline to answer any particular questions in the study without any consequences to my future treatment by the researcher. *

Mark only one oval.

- ◯ Yes
- ◯ No *Skip to section 13 (Thank you)*

5. I agree to provide information to the researchers under the conditions of confidentiality set out in the information sheet. *

Mark only one oval.

- ◯ Yes
- ◯ No *Skip to section 13 (Thank you)*

6. I wish to participate in the study under the conditions set out in the Information Sheet. *

Mark only one oval.

- ◯ Yes
- ◯ No *Skip to section 13 (Thank you)*

7. I consent to the information collected for the purposes of this research study once anonymised (so that I cannot be identified), to be used for any other research purposes. *

Mark only one oval.

- ◯ Yes
- ◯ No *Skip to section 13 (Thank you)*

8. Your Full Name *

_____

Skip to section 9 (Main Study)

Main Study

Thank you for signing the consent form. On the following page will be a video of a scene rendered with one of two algorithms; you should watch this video once, and then answer the following questions on your impressions of that video. The following page will then show another video of the same scene but rendered with a different algorithm. You will be asked the same questions about this scene. You will then be asked a final question on the next page.

First Video

Please read all the questions on this page and then view "**Test_Effect_1**" from the link below. For best results, please download the video before viewing.

Videos Folder

9. Rate the visual quality of the scene, on a scale of 1-9. *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	
Very	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very Good Quality

10. Please indicate your reasoning for this rating.

Second Video

Please read all the questions on this page and then view "**Test_Effect_2**" from the link below. For best results, please download the video before viewing.

Videos Folder

11. Rate the visual quality of the scene, on a scale of 1-9. *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	
Very	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very Good Quality

12. Please indicate your reasoning for this rating.

Concluding Remarks

13. Do you have any other comments regarding the two videos you have just seen?

14. What device did you use to view the videos? *

Mark only one oval.

- Laptop
- Desktop
- Phone
- Tablet
- Games Console
- Other: _____

Skip to section 13 (Thank you)

Thank you

Thank you for your participation. If you signed the consent form, you will receive a copy of the participant information sheet and the consent questions to the email address provided.

Untitled section

This content is neither created nor endorsed by Google.

Google Forms