Image Warping for Enhancing Consumer Applications of Head-mounted Displays

Edward M. Peek **Burkhard C. Wünsche** **Christof Lutteroth**

Department of Computer Science
The University of Auckland
Email: epee004@aucklanduni.ac.nz

Abstract

Head-mounted displays (HMDs) are highly immersive display devices which are increasingly targeted towards consumer-level video games, E-learning, training and other forms of digital entertainment. Despite the hardware now being available, quality factors — particularly latency — are still issues in large part due to consumer graphics hardware being tailored for throughput instead of latency, and the expectation of a nausea-free experience even on weak hardware. In this paper we discuss the benefits and disadvantages of using image warping as a means to improve frame rate and latency in the context of consumer applications. As part of this, we suggest two appropriate algorithms for performing the image warping. These methods are compatible with other latency reduction strategies such as predictive tracking, and require minimal changes to conventional 3D rendering processes. In addition, they are implemented purely in software and are therefore suitable for use on existing consumer PCs and HMDs. Initial evaluations indicate that artefacts from both warping algorithms are minimally visible for typical environments.

Keywords: head-mounted display, latency reduction, frame rate, image warping

1 Introduction

Today, there are many general consumer applications where people interact with 3D virtual environments. Video games, computer aided design, E-learning, online virtual worlds, 3D mapping and navigation and architectural walkthroughs are just some examples. Additionally, more and more devices are becoming capable of running these applications; what was previously only attainable using high-end desktop computers is now possible on portable laptop computers, tablets and even mobile phones at remarkably detailed visual quality.

What has changed little is the way which we view these virtual environments. It is still almost universally done by showing a single rendered image on a flat panel monitor that takes up a small portion of the user's visual field. This does not provide a very immersive experience, and only recently have more immersive display technologies started to become available at reasonable cost and quality. Stereoscopy is one such technology that has appeared in computer monitors, television sets, handheld gaming consoles and smartphones, however the need to wear special 3D glasses is a factor that has hindered the uptake of the technology.

One of the more immersive classes of display are head-mounted displays (HMDs). These are worn on the user's head and have the benefit of producing stereoscopic 3D, taking up a large portion of the user's visual field and blocking out the real world. Many types also have sensors to track their orientation and/or position, allowing the user to look around the virtual environment with natural head motion. Traditionally HMDs have been expensive, or of low quality, and have therefore been limited to specialised industrial, scientific, military and enthusiast audiences. Recently however, some HMDs that are both low cost and high quality have been designed for general consumer use. A notable example is the Oculus Rift (Oculus VR, 2012) which has gained support from prolific virtual reality (VR) enthusiasts and game developers (Oculus VR, 2013), despite being pre-production hardware.

While adapting 3D applications to use sophisticated VR displays is technically straightforward, certain quality and performance factors such as rendering latency and frame rate become significantly more important for an acceptable user experience. This necessitates tighter control of these factors than previously required in these applications. When coupled with the fact that most consumers would assume that the experience for sophisticated VR displays should be at least as good as that of conventional displays, there arises a need for methods to improve these quality factors beyond what is possible by using conventional methods.

In this paper we discuss a frame rate and latency enhancement technique based on image warping, and how it may be applied to and adapted for these sorts of applications. After exploring related work, we discuss the architecture of our image warping enhancement and the benefits it provides. We then discuss the characteristics of the enhancement with respect to consumer applications.

2 Related Work

Since low latency and high frame rates can be so important for virtual reality systems, much work has been performed over the years in an attempt to enhance them.

A universal way to compensate for any type of latency is to use prediction. In the context of HMDs, by predicting the position of the head at the time the render frame will become visible, instead of the raw sampled position, the average latency of head mo-

tions can be reduced. Azuma and Bishop (1994) discuss several ways to predict head position and find it can reduce the magnitude of tracking errors by 2–10 times.

One approach to provide optimal frame rates is to guarantee that the time it takes to render a frame is always less than the deadline for that frame being scanned out to the display. Olano et al. (1995) discuss a method for achieving this for HMDs; using a cluster of graphics processors to distribute computation. The authors also dynamically offset scan lines to further reduce latency by compensating for the time it takes to scan out an image to a CRT display. For current consumer HMD systems this is impractical, as the required cluster of graphics processors is very uncommon, and modern consumer HMDs are LCD or OLED based.

Kijima and Ojika (2002) develop the idea of shifting scan lines by designing a modulator system that is inserted between an LCD panel and the driving circuitry. The modulator has a direct feed to the predicted head position which bypasses additional sources of latency. Since this process requires physical modification of the HMD, it is less suitable for use on existing consumer HMDs, as it is unrealistic to expect average users to be able or willing to perform the modification.

A third class of latency compensation techniques has been researched that, like prediction, is able to be implemented entirely through software. These techniques are based on *image warping*, i.e. taking previously rendered views and modifying them to generate new ones. Mark et al. (1997) discuss how this can be utilised to improve general frame rates, and latency when using a remote display. The authors, however, do not provide an implementation of their system, something that is accomplished in more recent work.

Smit et al. (2007) present and implement an architecture for performing this warping. This is done using dual GPUs, one for rendering the virtual environment to produce what they call *application frames*, and one for rendering warped *display frames* that are then presented to the user. This architecture is tailored for local VR systems, in particular fish-tank virtual reality which combines stereoscopy with head-coupled perspective. In their later work they improve their architecture to reduce crosstalk with stereo shutter glasses (Smit et al., 2008), adapt the system for use on a single GPU (Smit et al., 2009), and explore the quality and performance of different warping algorithms.

In our own previous work (Peek et al., 2013) we have implemented and tested a image warping architecture specifically designed for HMDs. Our user evaluation showed that image warping significantly improves the smoothness of head tracking, and for most users is indistinguishable from ideally fast rendering; at least when head tracking is the only motion in the scene.

3 Image Warping

We extend prior work by examining how the practicality of these techniques is affected by their use in general applications, and with currently available consumer-level hardware. This builds upon our previous work in this area and provides a more detailed consideration of image warping in normal 3D applications. Within this paper, our reference scenario is an interactive video game on desktop class PC hardware using the Oculus Rift developer kit as the system display, and a keyboard and mouse for input. This is likely one of the major use cases for this wave of HMDs, and it is complicated enough to surface the issues we are looking for, while also allowing for some generalisation to other hardware platforms and applications. Considering this problem domain, the following points were considered especially important and had a large influence on the process of our investigation.

3.1 Domain Requirements

The first restriction due to this sort of usage, is that physical modification of computer hardware is unattractive to the user, so any sort of enhancement must be entirely software based. This rules out techniques that involve modification of the HMD hardware itself, such as the *Reflex HMD* proposed by Kijima and Ojika (2002). Conversely, techniques based on image warping via the GPU are practical in this case as they are typically able to be run on any modern desktop GPU.

An extension to this restriction is that is cannot be assumed that the user is willing or able to upgrade their computer hardware, specifically their CPU or GPU, in order to facilitate use of any enhancements. Since a large proportion of typical users' PCs have only a single GPU, techniques that require multiple GPUs (such as the work by Smit et al. (2007)) are impractical unless they are able to be modified to run on a single GPU (as in their later work (Smit et al., 2009)). It may be noted however that while many users have only a single *dedicated GPU* (dGPU), most modern CPUs have an *integrated GPU* (iGPU) that may be used in addition to the more powerful dGPU. However, such iGPUs are significantly slower (up to an order of magnitude), and so sophisticated techniques may not run fast enough to produce any improvement in latency or frame rate.

Furthermore, considering the range of PC configurations in current use, a very large number have only an iGPU as the sole graphics processor. In a July 2013 survey of desktop and laptop PCs used for gaming (Valve Corporation, 2013), the most common graphics processor was an iGPU (Intel HD 3000) and over 14% of systems had an iGPU as their only graphics processor. Such systems pose an ambivalent target, as while they have the most to gain from latency reduction and frame rate improvement, they are also the most difficult ones to develop for due to their limited capacities. The consequence of this is that for consumer applications, any enhancement technique should be capable of running on even modest PC hardware.

The last point of note concerns the structure of the application that is to be enhanced. With HMDs still being only a niche display type, it is unrealistic to expect application developers to enact significant changes to application architecture and rendering pipeline in order to implement an enhancement currently useful to so few users. This is especially true when considering its impact to development and testing costs, or to the quality and performance of conventional displays. The suitable response is to ensure that any proposed enhancement is compatible with popular rendering models (such as forward and deferred shading) with minimal changes, or even entirely separable in the style of NVIDIAs 3D vision (NVIDIA Corporation, 2013) or our own method for head-coupled perspective (Li et al., 2012).

3.2 Overview

This paper discusses how image warping, such as that by Smit et al. (2007), may be implemented taking into account these special requirements. We also tailor our method specifically for HMDs, where previous work (Smit et al., 2007) has targeted fish-tank virtual reality (FTVR) systems. In this section we present a high-level overview of our image warping enhancement, how it would be added to a conventional HMD software application, and two warping algorithms designed for the problem domain.

The central idea behind image warping (as a means to improve frame rate) is that for a rendered frame, subsequent frames are visually very similar and therefore may be extrapolated from the original frame with acceptably small errors. Error-free reference frames must still be regularly generated to prevent errors accumulating over time, and to account for the fact that only certain changes can be extrapolated by warping. To structure this method, we use an architecture similar to that proposed by Smit et al. (2007), in which the simulation and warping run concurrently and at separate rates. The simulation generates simulation frames using conventional rendering as fast as it can. At the same time, the warper takes the most recently produced simulation frame and uses it to generate a display frame at exactly the display's refresh rate: first by warping the simulation frame with up-to-date head orientation, and then performing the HMD lens correction. It is only the display frames that are made visible to the user via the HMD. Additionally, because HMDs are stereoscopic displays, each simulation and display frame contains a left and right eye view which are rendered and warped independently.

This set-up is able to increase frame rate because the generation of display frames is less computationally expensive than simulation frames, and thus able to be performed more frequently on the same PC hardware. However, because image warping shortens the generation of individual display frames, but requires the sequential generation of both simulation and display frames, it reduces some forms of latency while increasing others.

Basic image warping can only be used to extrapolate object and viewpoint motion between frames. Other types of motion require auxiliary data to be generated with each conventionally rendered frame. The required data for each type of motion is as follows.

Viewpoint rotation only requires the rendered image colour

Viewpoint translation requires the colour and depth of each pixel in the rendered image

Object rotation & translation requires the colour, depth and velocity of each pixel in the rendered image

Different algorithms exist for performing the actual image warp. This paper suggests and discusses two appropriate methods for our target domain in Sections 3.4 & 3.5.

3.3 Benefits

To quantify the benefits of image warping, we must compare how it relates to conventional rendering. Here we briefly discuss what variables influence the frame rate and latencies of these two rendering processes. In addition to frame rate, the two types of latency of interest are what we call tracking latency L_t and simulation latency L_s. Tracking latency specifically concerns HMDs in that it is the time delay between a rotation of the user's head and the rendered frame accounting for that rotation being visible to the user. Simulation latency is the time delay between a change in the simulation's state (including response to user input) and that new state being visible to the user. Usage of HMDs is particularly sensitive to tracking latency — an excess of which causes motion sickness — while simulation latency is more acceptable; but only to the extent that while in need not be aggressively reduced, it should also not be needlessly increased. In a GPU-bottlenecked conventional rendering process, as visualised in Figure 2, frame rate and latencies are given by the following equations.

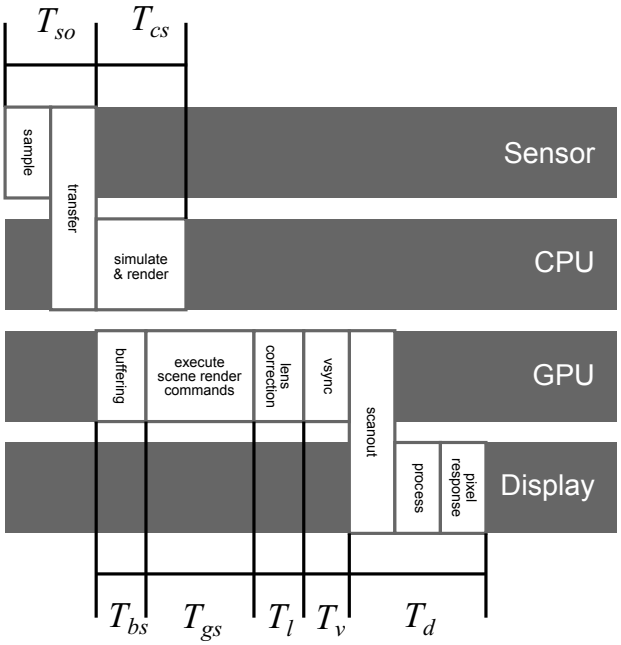


Figure 2: The constituent parts of tracking and simulation latency in the conventional rendering process

$$L_t = T_{so} + T_{bs} + T_{gs} + T_l + T_v + T_d \quad (1)$$

$$L_s = T_{bs} + T_{gs} + T_l + T_v + T_d \quad (2)$$

$$frame\ rate = \frac{1}{T_{gs}} \quad (3)$$

Using this rendering approach, frame rate and latencies are typically improved by reducing scene and shading complexity to indirectly control T_{gs}. Other delays are either impossible to affect from software, or have consequences from being changed.

The rendering pipeline modified for image warping is shown in Figure 3 and results in new equations (4–7) for frame rate and latencies

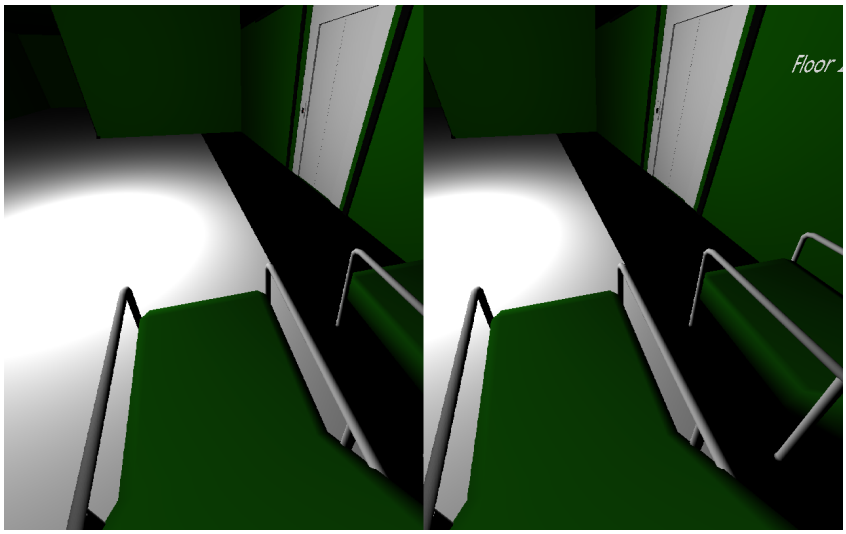
$$L_t = T_{so} + T_{bw} + T_{gw} + T_l + T_v + T_d \quad (4)$$

$$L_s = T_{bs} + T_{gs} + T_p + T_{bw} + T_{gw} + T_l + T_v + T_d \quad (5)$$

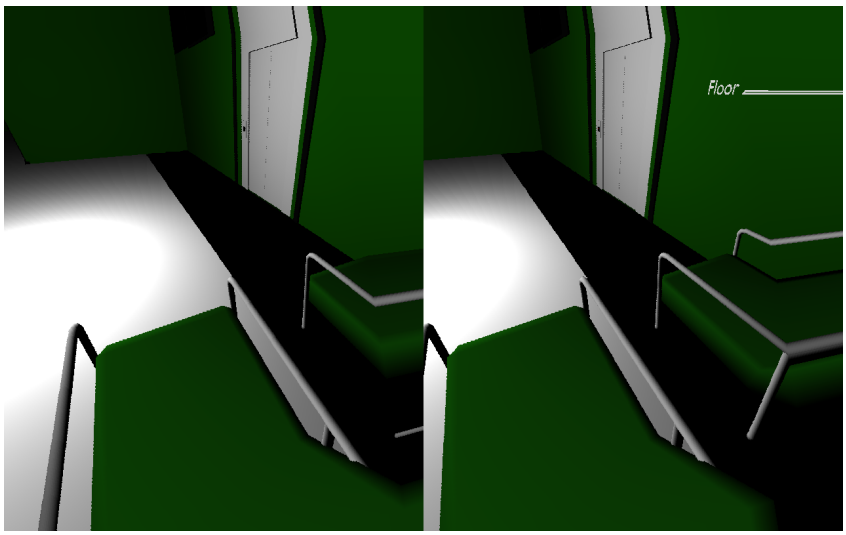
$$frame\ rate_{single\ GPU} = \frac{1 + r}{r \times T_{gw} + T_{gs}} \quad (6)$$

$$frame\ rate_{dual\ GPU} = \frac{1}{T_{gw}} \quad (7)$$

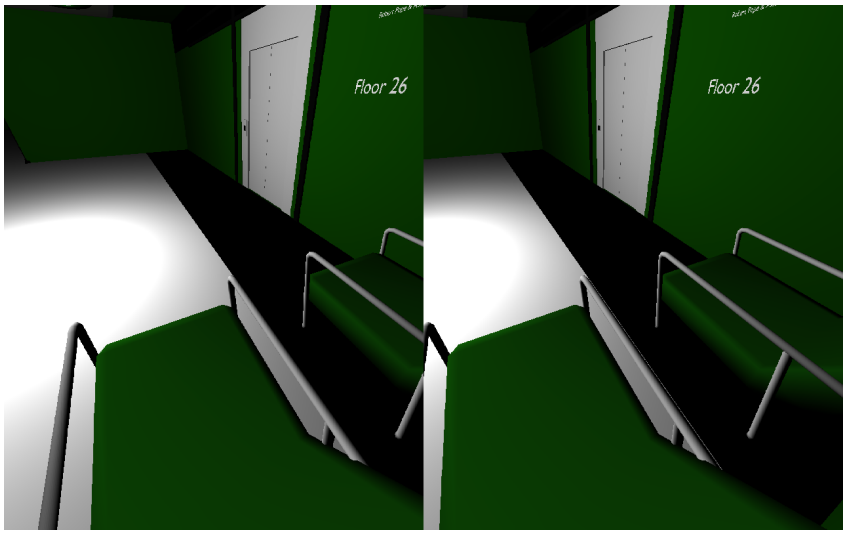
where r is the number of display frames generated per simulation frame.



(a) Simulation frame (t0), rendered before head rotation



(b) Display frame (t1), result of warping the simulation frame after head rotation. Fringe artefacts are evident in the top and right sides of the images, while a minor edge artefact is visible on the right chair arm in the right-eye image



(c) Simulation frame (t2), rendered after no additional head rotation. It corrects the errors introduced in the warped image

Figure 1: Example frame generation sequence, with an (unrealistically fast) head rotation to the upper-right. The rotation occurs between the rendering of the first simulation frame and the first display frame

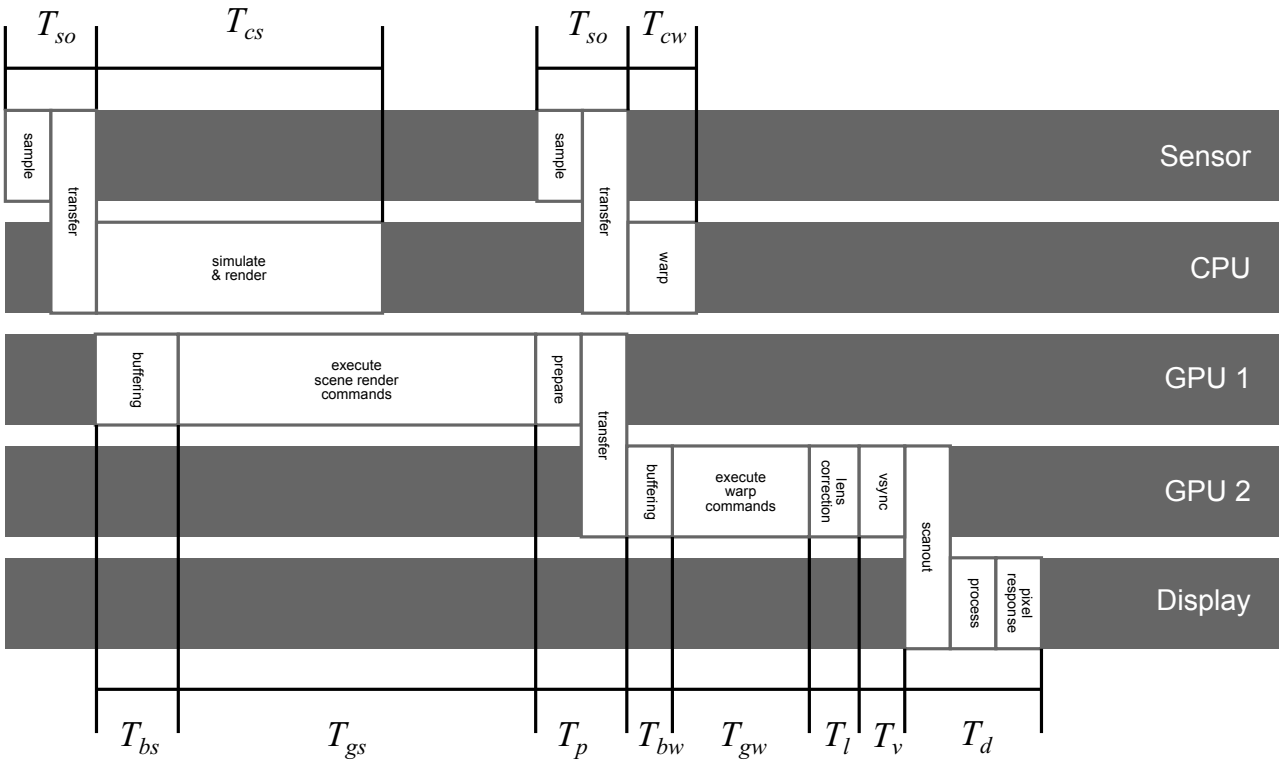


Figure 3: The sources of latency in image warping for a dual GPU system. For a single GPU system, all rendering work on the second GPU is shifted to the first and the cross-GPU transfer no longer exists

3.4 Reprojection Algorithm

The first warping algorithm we present is a reprojection of the rendered image as a textured screen-aligned quad. Consider the following variant of the matrix stack used to transform mesh vertices into screen-space (used ubiquitously in conventional rendering) for time t

$$\vec{v_t}' = PHBM\vec{v} \tag{8}$$

where P is the projection matrix, M is the model/world matrix, and the view matrix (normally V) is split into the simulation controlled body matrix B and the head coupled matrix H. The matrix to reproject the transformed vertex $\vec{v_t}'$ to its position at time $t+1$ is given by

$$R = PH_{t+1}H_t^{-1}P^{-1} \tag{9}$$

assuming there is no change in B or M. The full process for this method of image warping is therefore

1. Sample the head position to obtain H_t
2. Render the scene conventionally to an off-screen texture, using the transformation given in Equation 8
3. Sample the head position to obtain H_{t+1}
4. Render a quad mapped with the texture from step 2, and aligned to the rear face of the screen-space frustum to the back buffer, using the transformation matrix given in Equation 9
5. Flip the back buffer to display the rendered quad to the user

Since H is the only variable that is updated for the warping, coupled head orientation and translations are the only types of motion enhanced by this method. Changes to B are not considered in order to fully decouple the warping from scene simulation, although in future work it could be predicted. M is likewise ignored due to being simulation controlled, but is also per-object, making it significantly more complicated to predict and store than B. It should also be noted that because this method does not consider frame depth information, the head translation will be imperfectly warped, and objects will be distorted as if their depth is the same as the distance to the far clipping plane. The effect of this error is discussed in detail in Section 4.5.

This warping method runs in constant time for a given screen resolution, and is extremely fast as it consists of just 4 vertices transformed by a single matrix, and a single texture lookup per-pixel. This makes it appropriate for use on low-end GPUs (particularly iGPUs) which are too slow to perform more sophisticated warping algorithms.

3.5 Raytrace Algorithm

The second warping algorithm corrects for the translation error introduced in the reprojection method. By utilising the depth buffer information that accompanies key frames, the distance to scene objects need no longer be assumed constant. The disadvantage of this is that the image warping can no longer be performed by a forward texture lookup, and must resort to a linear search through the frame to find the new colour value. Forward lookup based methods such as point splatting do exist (Smit et al., 2007), but raytracing has attractive properties that will be discussed in later sections.

Effectively, the core difference between the two algorithms is that in reprojection, the simulation frame is rendered as a flat quadrilateral, while under raytracing it is rendered as a frustum-shaped height map. To draw this height map, each pixel of the screen

is treated as a ray that is cast into the screen and through the height map. The colour of the height map where the ray intersects with it is used to colour the pixel for which the ray was generated.

To perform this intersection, firstly each ray's start and end point is created in screen-space, and then transformed into the coordinate system of the height map by multiplying by R^{-1} (from Equation 9). A ray-plane intersection is then performed against planes at the front and back of the height map's coordinate system to give texture coordinates for the start UV_{start} and end UV_{end} of a linear search. The linear search iterates across the depth values of the simulation frame until it meets the following inequality.

$$sample_{depth}(i * UV_{start} + (1 - i) * UV_{end}) \leq i \quad (10)$$

where $0 < i < 1$ and represents the offset and depth of the search, and n_d is the number of pixels (and hence number of samples tested) between UV_{start} and UV_{end}. Once the inequality returns true, the colour at the successful UV coordinate is returned as the result of the raycast.

This technique's runtime is proportional to the amount of translation of the viewing position. This is because the amount of translation directly affects how separated the start and end points are of the linear search, which subsequently dictates the number of depth samples (n_d) needed: the major performance cost of this method. Another variable that controls n_d is the near clipping distance, which is normally recommended to be made as large as possible to prevent Z-fighting. The positive side of this is that for no translation, no depth lookups are required, giving comparable performance to reprojection. The average number of texture samples needed is approximated by

$$n_d \propto \frac{translation}{clip_{near}} \quad (11)$$

assuming $clip_{far} \gg clip_{near}$.

4 Results & Discussion

The major contribution of this paper is the consideration of how real-world factors influence the practicality of these frame rate and latency enhancement methods. This section contains the discussion of these factors.

4.1 Required Program Modifications

One of the factors influencing the practicality of image warping is the extent to which it requires changes to the architecture of a piece of software in order to accommodate the enhancement. What's desirable is requiring few changes to the conventional rendering process, as well as being compatible with a large range of lighting and shading techniques.

In this regard both methods perform extremely well. Both methods attach to the very end of the rendering process, making the integration point between the simulation software and the enhancement very small. The only process change is that the final rendered image frame is stored in an off-screen texture, rather than the back-buffer of the swap-chain. For reprojection, this is the only technical requirement, while for raytracing, there is the additional requirement that the depth buffer must also present alongside the off-screen texture and with valid data.

While depth buffers are ubiquitously used, some programs render the scene as multiple layers to improve Z precision, clearing the depth buffer in-between rendering layers. This is incompatible with the raytracing algorithm, as it expects objects to have correct depth information. This does however have the benefit of preventing excessive use of this optimisation, which can cause incorrect occlusion where layers overlap, something barely noticeable on conventional displays, but quite distracting on stereoscopic displays (including HMDs).

While the requirement of a motion-field buffer would allow simulation motion smoothing (Smit et al., 2007), this would require a large step-up in the intrusiveness of the enhancement which is one of the reasons it was not considered in this paper.

4.2 Head Rotation and Translation

The image warping enhancement described in this paper is concerned with improving frame rate and latency specifically for head tracking. Furthermore, of the two components of tracked head motion, orientation and translation, orientation is strongly prioritised over translation. The reasons for this are threefold.

Firstly, we consider coupled head orientation to be the major immersion factor for HMDs, as it is what allows the user to naturally look around the virtual environment.

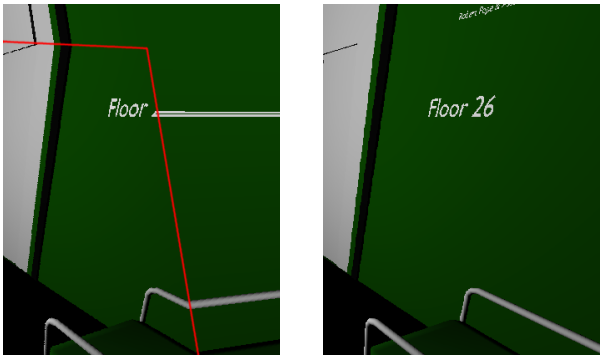
Secondly, it is the latency in orientation tracking that primarily causes motion sickness: by making the world appear to wobble, roll and lag with fast head movement, and by preventing the vestibulo-ocular reflex from correctly stabilising the user's view of the scene.

Lastly, the reference HMD (Oculus Rift developer kit) only contains an orientation tracking system, and none for positional tracking. This means that any translation in the viewpoint caused by this tracking is purely from correcting the rotation to revolve around the user's neck, and not around the midpoint of the user's eyes. This produced translation is quite small, and so becomes negligible compared to changes in orientation.

Another point of consideration is the instantaneous angular velocity of head rotation. This is important as the size of certain types of artefacts (Sect. 4.3 & 4.4), the performance of the raytrace algorithm (Sect. 3.5), and sample-and-hold blurring (Sect. 4.7) all depend on this velocity. The details of this relationship are discussed in the relevant sections, but in general, the slower the head rotation the better. This turns out to be ideal regarding actual usage, as our experience suggests long periods of small or no rotation, with occasional bursts of medium speed rotation. Rapid rotation is rare, and even suppressed due to the weight and inertia of the HMD.

4.3 Fringe Artefacts

The major visible artefact caused by image warping with HMDs is holes around the edges of the screen, what we call *fringe artefacts*. This is caused by rotation of the viewpoint, which results in sampling beyond the limits of usable data around the edge in the direction of rotation. Raytracing naturally fills these holes, while reprojection can be trivially adapted to fill them in the same manner: by transforming the *texture coordinates* of the quad's vertices by R instead of their *positions*. Both of these work by clamping the sampled texture coordinates to stay within the bounds of valid data, effectively stretching the edges of the key frame over the holes.



(a) Fringe error caused by warp; the red line separates the valid image data and the error

(b) How the frame would look if rendered without artefacts

Figure 4: Example of a fringe artefact, with comparison to an error-free rendering

Since these artefacts always appear at the edges of the screen, and due to the large field-of-view provided by the evaluated HMD, these artefacts typically appear in the user's peripheral vision. Peripheral vision is more sensitive to motion than the rest of the field-of-view, which may emphasise the flickering caused by the artefacts rapidly appearing and disappearing as they are corrected. Peripheral visual acuity is however worse than central vision (foveal), and the geometry of the Oculus Rift lenses prevents the user from looking directly at objects in the periphery of the HMD, meaning fringe artefacts are very difficult to *directly* observe.

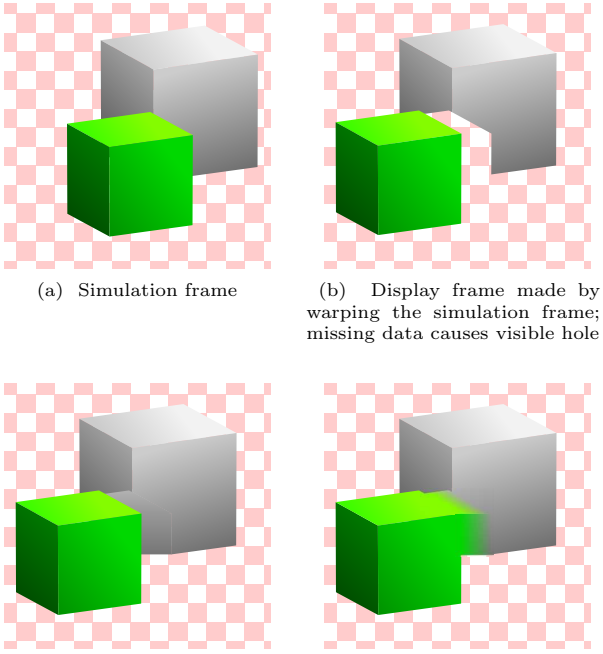
In addition, HMD rendering is frequently designed to allow for rendering beyond the normal edges of the display. This is done to compensate for a shrinking of the rendered image due to lens distortion correction. This can be reused as-is to hide fringe artefacts by pushing them further off the sides of the display, at the cost of increasing T_{gs} due to rendering more pixels than are visible.

4.4 Edge Artefacts

When translating the viewpoint position using the raytrace algorithm, near objects will be shifted by the warp more than distant objects. Where depth in the scene changes suddenly, such as when a near object occludes a far object, the two objects may be warped away from each other creating a visible artefact along the edge between them. Because our raytracing algorithm treats the height map as continuous, these artefacts do not appear as holes, but rather as a stretching of the colour at the edge.

The algorithm may be trivially adapted to stretch image data across the gap in one of two ways. The first approach fills the gap using the farther side of the edge, while the second approach is a linear interpolation of the near and far sides. The benefit of the first method is that it more realistically handles object occlusions, which are the most common cause of edge artefacts, albeit at the expense of being incapable of handling multisampled anti-aliasing. The linear interpolation approach removes this incompatibility, but at the cost of having less realistic fill appearance. A comparison of the the two filling methods can be seen in Figures 5c & 5d.

The size of edge artefacts is directly related to the amount of coupled head translation, which is in turn related to the velocity of head rotation (Sect. 4.2), and to the ratio of the object depths as given by Equation 12.



(a) Simulation frame

(b) Display frame made by warping the simulation frame; missing data causes visible hole

(c) Hole filled using far edge

(d) Hole filled by interpolating between both near and far edges

Figure 5: Example of an edge artefact caused by moving the viewpoint to the right, as well as the two filling methods that could be used to mask it

$$error \propto translation \times \left(\frac{distance_{far} - distance_{near}}{distance_{far} \times distance_{near}} \right) \quad (12)$$

From this equation, it can be seen that as the objects get farther away, or closer together, the error approaches zero. Slow head rotation, coupled with objects at reasonably far distance and frequent key frame generation all serve to minimise the visibility of edge artefacts. Due to the many variables involved in their size and contrast, it is difficult to give exact values for these variables beyond which these artefacts become negligible; however the presence of sample-and-hold blurring significantly reduces their visibility, which is discussed in Section 4.7.

4.5 Translation Errors in Reprojection

It was mentioned within the description of the reprojection algorithm, that while the method attempts to correct for viewpoint translations, it is unable to do so without error due to ignoring depth information. The effect of this is that translation is under-compensated for near objects, but approximately correct for far objects. The amount of error can be determined through Equation 12 where $distance_{near}$ is the distance to the object, and $distance_{far}$ is the distance to the far clip plane. The appearance of this error is that, during head rotation, nearby objects jerk slightly in the opposite direction of the rotation.

Considering the points discussed in Section 4.2 along with the fact that for typical virtual environments most objects are at reasonable distances, the visibility of these errors is quite small.

4.6 Head-coupled Scene Objects

One of the major assumptions of these image warping methods is that no scene objects are coupled to the head position or orientation. Unfortunately there are a few common scenarios that violate this assumption. The most common examples of this are the head-up display and other types of 2D user interfaces. These are drawn at fixed positions on-screen and so may be modelled as scene objects coupled to the camera's position. Examples of other common coupled 3D objects are parts of the player's avatar, particularly the hands and held objects.

The violation in the warping model occurs because these types of objects are not transformed by a matrix stack of the form given by Equation 8. The visible effect of this is that these objects will judder in the direction of any head motion, instead of remaining stationary (relative to the user's head).

The simplest solution to this problem is to avoid using these types of head-coupled objects. Some video games already follow this model and do not show any 2D HUD, and opt to display the equivalent information entirely using in-game objects.

Another option is to shift the rendering of head-coupled objects themselves into the warping pass. There are three main disadvantage to this. It deteriorates the performance of the warping pass due to the increase in rendering complexity, although not significantly as most HUDs tend to be reasonably simple. It more tightly couples the warping pass to the simulation logic, making it more difficult to run them at independent rates. And finally it increases the intrusiveness of the enhancement by requiring more deep changes to the conventional rendering process.

A middle ground could be to modify the warping methods to detect head-coupled objects and then avoid warping them. This would require some way to differentiate between head-coupled and normal objects. Since head-coupled objects are typically rendered in front of all other scene objects, a simple solution would be to apply a threshold to the depth buffer, where objects below the threshold are not warped, while those behind are. While this is trivial to implement, it does introduce edge and transparency artefacts which, due to the great difference in depth between HUD and scene, would be significantly more visible than their manifestations in ordinary scene objects.

4.7 Sample-and-hold Blurring

One of the most interesting factors that deserves attention is a blurring effect due to the finite frame rate and sample-and-hold nature of the HMDs display. This appears as a linear blur during head rotation when fixated on a stationary point in the scene. While a similar blur also occurs when fixated on moving objects, whether on a conventional monitor or HMD, it is how the blur caused by head rotation interacts with and masks warping artefacts that is of particular interest.

The reason this occurs is that while the vestibulo-ocular reflex causes continuous tracking of the object on the display, the motion of the object on the display itself occurs in discrete steps. This means that the focused image of the object follows a rapid saw-tooth pattern across the surface of the retina, causing it to be perceived as blurred since it moves faster than the threshold of persistence of vision.

It has been noted previously that many types of artefacts depend on head translation, and subsequently head rotation. Because the size of this blur also depends on the amount of head rotation, it will naturally mask these artefacts by blurring them by an amount directly proportional to their size. We find the size of this blur effect due to head rotation to be approximated by the following equation

$$blur\ width = \frac{\omega}{frame\ rate} \tag{13}$$

where ω is the angular velocity of the user's head in radians per second, and *blur width* is the visual angle of the perceived blur in radians.

The blur is itself a visual artefact however, albeit one that cannot be reduced much using software. The blur can be minimised by ensuring that the tracking frame rate is at least equal to the display's refresh rate, 60Hz in the case of the Oculus Rift. Otherwise the blur may only be reduced though redesigning the HMD hardware to support a higher refresh rate, and/or strobe its image.

5 Future Work

While the raytrace warp (Sect. 3.5) has fast performance for minimal head movement, the fact that performance deteriorates linearly with the speed of head motion makes the method unusable on low and mid-range hardware. It does not help that the major performance cost is texture bandwidth, which is particularly limited on iGPUs as it must be shared with the CPU. Therefore more work is needed to either optimise this algorithm, or to find another of better performance characteristics that supplements the reprojection algorithm as a higher quality warping method for faster systems.

Another issue with the proposed enhancement architecture is timing issues caused by GPU command buffering. In order to avoid excessive user-space to kernel mode switches on the CPU, and to prevent the GPU from stalling with no work to do, consumer GPU drivers and hardware will buffer rendering commands. This helps to improve *throughput* at the expense of *latency*. This is preferable on conventional display types which are latency-insensitive, but highly undesirable for VR systems such as HMDs which are not. This causes further issues on a single GPU system where the concurrent simulation and warping logic interleave their rendering commands, making controlling the timing of them very difficult. Future work is needed to find ways to avoid command buffering during the generation of display frames (while it remains acceptable for simulation frames), which might be possible though finding a way to give higher priority to warping commands and by manually flushing the command buffers.

In this paper we have given a technical discussion on the perception of the benefits and artefacts of the warping algorithms, and personal experience supports our conclusions. However, a proper user study is required to ensure the proposed enhancements really do give a better user experience with actual users and in representative applications.

The last major point in need of further attention is the practicality of the proposed enhancements as injectable algorithms. Since the enhancement is so minimally intrusive, it might be possible to modify it so that it can be retroactively applied to existing HMD applications in the vein of NVIDIAs 3D Vision (NVIDIA Corporation, 2013) or our method for conversion of desktop VR to support head-coupled perspective (Li et al., 2012).

6 Conclusion

We have presented and discussed the use of and image warping based enhancement for improving frame rate and head tracking latency, that is appropriate for use with consumer HMDs (particularly the Oculus Rift) and applications.

Two algorithms are suggested for performing the actual warping, one based on reprojection and the other on raytracing, that are suitable for use in PC systems with either one or two GPUs, and of variable processing power. The reprojection algorithm has the constant performance of rendering a single textured quad, as does the raytracing method for slow head rotations, although the raytracing computation cost increases linearly with head rotation speed. These warping methods exploit properties of real-world HMD usage such as minimal head translation and require only small changes to the conventional rendering process for HMDs.

Furthermore, we have described two types of error that are visible at the edges of the display and along object edges during head motion, and how the sample-and-hold nature of HMD displays produces a blur which helps to mask them.

Lastly we believe this type of enhancement in their current form should prove valuable for improving user experience in many consumer HMD applications, and even more with the suggested slight modifications.

References

Azuma, R. and Bishop, G. (1994), Improving static and dynamic registration in an optical see-through hmd, *in* 'Proceedings of the 21st annual conference on Computer graphics and interactive techniques', SIGGRAPH '94, ACM, New York, NY, USA, pp. 197–204.

Kijima, R. and Ojika, T. (2002), Reflex hmd to compensate lag and correction of derivative deformation, *in* 'Virtual Reality, 2002. Proceedings. IEEE', pp. 172–179.

Li, I. K. Y., Peek, E. M., Wünsche, B. C. and Lutteroth, C. (2012), Enhancing 3d applications using stereoscopic 3d and motion parallax, *in* 'Proceedings of the Thirteenth Australasian User Interface Conference - Volume 126', AUIC '12, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 59–68.

Mark, W. R., McMillan, L. and Bishop, G. (1997), Post-rendering 3d warping, *in* 'Proceedings of the 1997 symposium on Interactive 3D graphics', I3D '97, ACM, New York, NY, USA, pp. 7–ff.

NVIDIA Corporation (2013), 'Nvidia 3d vision'.
URL: *http://www.nvidia.com/object/3d-vision-main.html*

Oculus VR (2012), 'Oculus rift - virtual reality headset for 3d gaming'.
URL: *http://www.oculusvr.com/*

Oculus VR (2013), 'John carmack joins oculus as cto'.
URL: *http://www.oculusvr.com/blog/john-carmack-joins-oculus-as-cto/*

Olano, M., Cohen, J., Mine, M. and Bishop, G. (1995), Combatting rendering latency, *in* 'Proceedings of the 1995 symposium on Interactive 3D graphics', I3D '95, ACM, New York, NY, USA, pp. 19–ff.

Peek, E. M., Wünsche, B. C. and Lutteroth, C. (2013), More for less: Fast image warping for improving the appearance of head tracking on hmds, *in* 'Image and Vision Computing New Zealand, 2013. IVCNZ '13. 28th International Conference'.

Smit, F. A., van Liere, R. and Fröhlich, B. (2007), The design and implementation of a vr-architecture for smooth motion, *in* 'Proceedings of the 2007 ACM symposium on Virtual reality software and technology', VRST '07, ACM, New York, NY, USA, pp. 153–156.

Smit, F. A., van Liere, R. and Fröhlich, B. (2008), An image-warping vr-architecture: design, implementation and applications, *in* 'Proceedings of the 2008 ACM symposium on Virtual reality software and technology', VRST '08, ACM, New York, NY, USA, pp. 115–122.

Smit, F., Van Liere, R., Beck, S. and Froehlich, B. (2009), An image-warping architecture for vr: Low latency versus image quality, *in* 'Virtual Reality Conference, 2009. VR 2009. IEEE', pp. 27–34.

Valve Corporation (2013), 'Steam hardware & software survey: July 2013'.
URL: *http://store.steampowered.com/hwsurvey*