Using Integrated GPUs to Perform Image Warping for HMDs

Edward Peek
Department of Computer Science
The University of Auckland
Auckland, New Zealand
epee004@aucklanduni.ac.nz

Burkhard Wünsche
Department of Computer Science
The University of Auckland
Auckland, New Zealand
burkhard@cs.auckland.ac.nz

Christof Lutteroth
Department of Computer Science
The University of Auckland
Auckland, New Zealand
lutteroth@cs.auckland.ac.nz

ABSTRACT

In virtual reality applications, frame rate and latency are critical performance metrics for maintaining a comfortable user experience and avoiding simulator-sickness. Various methods may be used to improve frame rate and latency, however they often come at the cost of image quality or other performance metrics.

One particularly beneficial method is synthesising additional and/or lower-latency frames using image warping. However, desktop graphics subsystems lack the required level of parallelism to effectively implement a complete image warping solution on computers with a single GPU. Fortunately many computers that are treated as single-GPU systems are, in fact, multi-GPU due to the presence of a low-performance secondary GPU integrated with the CPU package.

In this paper we describe an image warping system which exploits the hardware parallelism offered by integrated GPUs to avoid the aforementioned graphics subsystem limitations. This system improves both perceived frame rate and latency: in contrast to alternative systems which only improve one or the other.

We also evaluate the performance of performing image warping on integrated GPUs. Our system is able to perform a warp in as little as 4.2 ms at 1920×1080 resolution on an Intel HD Graphics 2000 IGP at the cost of a 1.5 ms increase in application render time. This demonstrates that the overhead of our system is manageable for current VR applications even on several year old computer hardware.

IVCNZ '14, November 19 - 21 2014, Hamilton, New Zealand
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3184-5/14/11$15.00
http://dx.doi.org/10.1145/2683405.2683445

Categories and Subject Descriptors

I.3.3 [**Computer Graphics**]: Picture/Image Generation—*bitmap and framebuffer operations*; I.3.1 [**Computer Graphics**]: Hardware Architecture—*graphics processors, parallel processing*; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Design, performance

Keywords

Image warping, integrated graphics processing unit, head-mounted display, frame rate, latency

1. INTRODUCTION

In applications that deliver real-time 3D computer graphics, image frames must be rendered at a fast enough rate to induce the illusion of motion for the user. Interactive applications have the additional requirement that the delay between some interaction with the application and the visible result of that interaction be small enough so that the response appears close to instantaneous. The performance metrics corresponding to these requirements are frame rate and latency, with both being essential for achieving a satisfactory user experience.

Frame rate and latency are both highly dependent on the time it takes the computer hardware to render each image frame. In modern computer architectures, fast frame rendering is typically realised through the use of a graphical processing unit (GPU) that asynchronously executes render commands issued to it by the central processing unit (CPU). Graphics application programming interfaces (API) typically buffer some quantity of render commands to ensure the GPU does run out of commands to process and stall, which would be a waste of computational power. Batches of commands issued to the GPU are also typically executed atomically, which avoids the overhead and complexity of performing execution context switches.

These optimisations are targeted towards achieving a balance of frame rate, latency and image quality for viewing on flat-screen monitors. This balance is however suboptimal for many virtual reality (VR) applications and display systems. This is because the various types of object and body tracking in VR applications are prone to accentuate

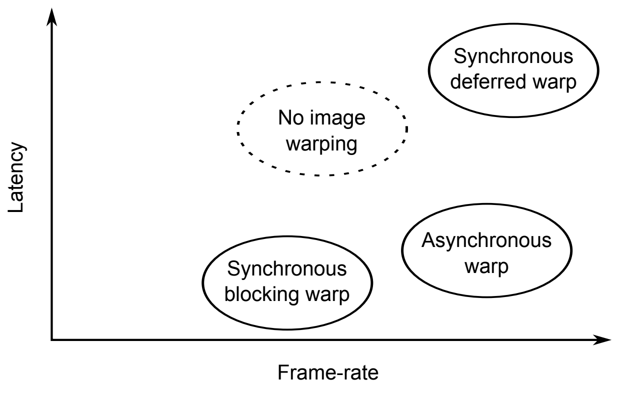

Figure 1: Spectrum of the performance trade-offs of different image warping methods. Having both low latency and high frame rate is desirable.

poor frame rate and latency. While poor frame rate and latency are merely visually unappealing and frustrating on computer monitors, on head-mounted displays (HMD) they cause physical discomfort and, in some circumstances, simulator sickness. This is a problem as — if such issues are common in consumer applications — it will likely inhibit the adoption of HMDs as an alternative display mode for consumer 3D applications.

To counteract this, any number of methods may be employed to improve frame rate and/or latency. Some examples include: reducing rendering quality, CPU-GPU synchronisation to reduce command buffering, predicting tracked motion, and supplementing rendered frames with ones synthesised via image warping. All of these methods involve some sort of trade-off between image quality, frame rate and latency. The trade-offs associated with different forms of image warping are visualised in Figure 1.

Of these methods, image warping has one of the highest potentials to improve perceived frame rate and latency while only negligibly impacting image quality. Unfortunately, correctly scheduling image warping on a single GPU system is made difficult due to the minimal level of software parallelism offered by most operating systems' (OS) graphics APIs. This is because render command buffering and the inability to pre-empt executing render commands precludes the accurate timing needed for image warping. Partial implementations that improve either (but not both) frame rate or latency are still possible with a single GPU, but multiple GPUs are needed to circumvent these limitations and provide the full benefits of image warping.

Consumer graphics applications are largely designed to ignore anything other that the primary GPU of the system they run on. One reason for this is that because any particular system may not have multiple GPUs, applications must be designed to cater to the lowest common denominator of only a single GPU being present. Secondly, systems that do have multiple GPUs often have them virtualised to appear as a single (more powerful) GPU to applications. And thirdly, in most systems where multiple GPUs can't be virtualised, the vast difference in GPU performance means utilising the weaker GPU is not worth the additional synchronisation overhead.

In this paper we describe an image warping system targeted towards the third category of consumer computer systems, which is able to improve both frame rate and latency. This is a novel application of weak secondary GPUs, which benefits from their hardware parallelism and avoids being limited by their poor performance due to the low computational complexity of image warping. The ubiquity of integrated graphics processors (IGP, or iGPU when in contrast to dedicated GPUs) on computer systems that also posses a dedicated GPU (dGPU) makes our image warping approach practical for a significant proportion of consumer VR users.

2. RELATED WORK

Temporal coherence between image frames in real-time graphics applications makes methods that recycle previously rendered image data attractive for reducing the computation required to produce new frames. One such method is image warping, which is able to derive new image frames from only the data contained within previously rendered frames [3]. These methods may be used to provide a higher frame rate and lower latency for a given level of image quality.

Recent research in image warping demonstrates that it is a practical solution for improving frame rate for consumer-level 3D computer applications including video games [1]. Because warp performance is critical for it to be beneficial over conventional rendering methods, it is important to demonstrate that various warping algorithms have acceptable performance on current hardware platforms [2].

In addition to frame rate, image warping systems have also been designed that improve the interaction latency of the application. Such systems are employed by some HMD rendering frameworks, including that of the Oculus Rift, a consumer-grade HMD [4].

Asynchronous image warping systems require warping and rendering to run in parallel, which may be achieved virtually or through hardware parallelism. Smit et al. [7] discuss the trade-offs between the two approaches for their modern image warping architecture. In general, hardware parallelism incurs an overhead due to the need to copy data between the different GPUs performing warping and rendering. Their virtual parallelism method does not have this overhead, but instead rendering commands must be manually split into small chunks to allow the GPU scheduler to switch to warping at the correct time.

Our own research into image warping has investigated the practicality of using asynchronous image warping on consumer-level HMDs [5]. We have also demonstrated that asynchronous image warping is perceptually difficult for users to distinguish from conventional rendering in the context of head tracking [6].

This paper develops upon prior work by demonstrating that dual-GPU asynchronous image warping architectures are practical on computers with asymmetrical GPUs; i.e. where one GPU is an IGP and therefore significantly slower than the other. This is in contrast to prior work which either use a high-performance dedicated GPUs for warping, or perform image warping synchronously on the same GPU as application rendering.

3. DESIGN

There are many possible approaches to architecting an image warping system, with different architectures having different performance characteristics. Our system is designed with the following performance objectives.

1. Decouple warped frame rate from application frame rate
2. Minimise warped frame latency
3. Minimise dedicated GPU stalls
4. Minimise application frame latency

Warped frame latency is given a high priority as it strongly influences the perceived application performance for HMDs [6]. Another notable point is that our system is designed to not limit, yet still be agnostic of application performance: a difference from synchronous image warping approaches.

While these priorities are what were used for the exact performance measurements, our system is flexible in that it supports several synchronisation methods with each giving a slightly different balance.

3.1 Platform Considerations

The major novelty of our research is that our system performs image warping on the computer's integrated GPU, and application rendering on the dedicated GPU. This comes with several important factors that must be considered.

Firstly, the main reason iGPUs are rarely used for co-processing is that they are vastly slower than even mid-range dedicated GPUs. This is both in terms of execution speed and memory bandwidth which can be as much as an order of magnitude less than dGPUs. This means that the additional software complexity and synchronisation overheads usually negates any benefit that offloading to the iGPU provides.

An additional consideration is that not all target systems have an iGPU available. While nearly all modern CPU ranges by Intel and AMD feature iGPUs, some specific models do not, meaning some systems may have only a dedicated GPU. On the other hand, because of the ubiquity of iGPUs and their steadily growing performance, for many users the low performance of an iGPU is sufficient for their workloads. A result of this is that many computer systems do not have a dedicated GPU, saving cost, energy and space. Furthermore, even on systems that contain both an iGPU and a dGPU, the iGPU may not be usable for various reasons. One reason this might be is that the system chipset does not support the use of iGPUs, or does not allow an iGPU to be active at the same time as a dGPU; an example of this is the Intel P67 chipset. A final reason the iGPU may not be available for warping is that it may already be used to accelerate graphics rendering. An example of this is the Hybrid CrossFire technology supported by some AMD CPU and GPUs. While it is possible to disable such technologies, doing so negates the benefits of image warping due to a reduction in base rendering performance.

Our system address these issues in the following ways. Firstly, the low complexity of image warping means that it can execute quickly even on slow hardware. Secondly, it is the hardware parallelism of the iGPU that is the key resource exploited by our system, which is not something limited by iGPU performance. Lastly, we expect the prevalence of compatibility issues to be low for the intended target audience. This is in part due to a key demographic of HMD VR being PC gamers, a large portion of which (around 80%) have dedicated GPUs [9]. Taking into account the rarity of other incompatibilities, we expect our technique to be applicable to a significant proportion of HMD users.

3.2 System Architecture

Our image warping system is designed as a pluggable library for use in applications employing the Microsoft DirectX API (specifically D3D11). DirectX was chosen over OpenGL as it has been historically more commonly used for PC video games, one of the major applications of consumer HMD virtual reality. Our system is accessed through an alternative implementation of the IDXGISwapChain interface, allowing it to be utilised with minimal changes to application code. This replacement class acts as a programmable display layer (PDL), similar to that described by Smit et al. [8], allowing for the presentation of application frames to be controlled dynamically by the application.

3.2.1 Application Rendering

Because our system provides an interface that is API-compatible with DirectX, very few changes need to be made to the application's rendering logic. The only significant runtime requirements imposed by our API is that the application must pass any frame metadata needed for warping through to the PDL. The format of the required frame metadata depends on the warping algorithm bound to the PDL. For the simple reprojection-based warp used in our evaluation, this data consists of just the view and projection matrices used for rendering.

API calls that interact with the application's back-buffer are modified by our replacement swap-chain object. The effect of these modifications is that the application renders to a PDL-managed framebuffer, replacing the default behaviour of rendering to a framebuffer that is able to be scanned-out directly to the display. Attempts by the application to cycle the swap-chain will instead notify the PDL that a new frame is available for processing.

This aspect of our system design allows the application to render in a manner mostly agnostic to whether warping is taking place, simplifying development.

3.2.2 Warping Algorithm

The PDL allows the programmer specify a custom program for frame presentation, conceptually similar to how programmable shaders allow for custom vertex and pixel processing. In this evaluation we bind our asynchronous warping algorithm — referred to as the *warper* — to the PDL. The evaluated warper configuration operates as follows.

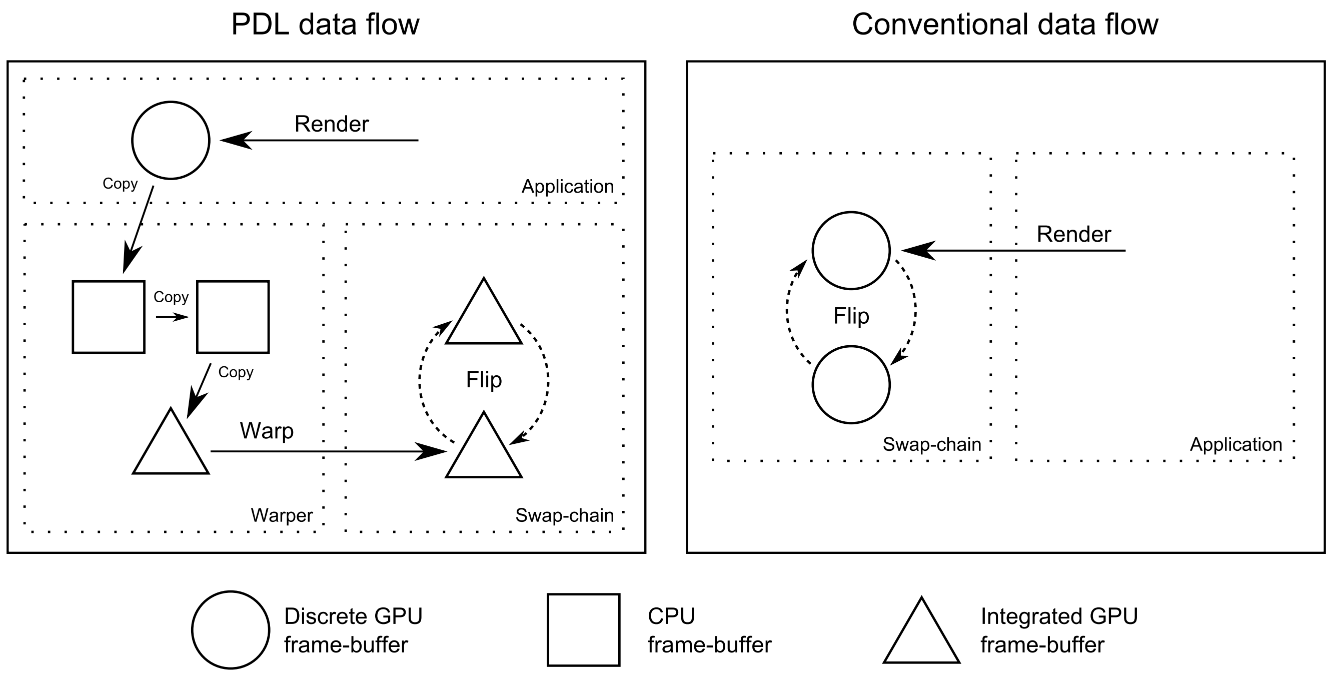

Figure 2: Data flow diagram of rendered image data through frame-buffers, comparing our PDL to a conventional double-buffered swap-chain. Buffers involved in application-specific rendering not displayed.

On notification by the PDL that an application frame has been submitted, the warper blocks the application thread until the GPU has completed rendering, after which it immediately copies the frame from dGPU memory and maps its contents allowing it to be read by the CPU.

Simultaneously, in another thread, the warper repeatedly renders warped frames according to the following sequence. Firstly it checks to see if a new application frame has been mapped by the process described in the above paragraph. If a new frame exists, it is copied by the CPU to a framebuffer accessible by the iGPU, then converted into the required tile format. The warper then samples the HMD sensors to get up-to-date parameters for the warp. Using these parameters, the actual warp itself is then rendered on the iGPU, using the reprojection warp described in our previous work [6]. It lastly blocks until the iGPU is idle to prevent future warp frames from being delayed by command buffering.

4. PERFORMANCE EVALUATION

In addition to being simple to integrate, our system is also intended to be fast enough to be beneficial on mid-range consumer hardware configurations. Our performance evaluation demonstrates that this is the case, through addressing the following performance concerns. Firstly, by ensuring that the overhead of warping architecture does not introduce an excessive amount of resource contention with the application's rendering. Secondly, to ensuring that the poor performance and smaller memory bandwidth of iGPUs is still sufficient to perform image warping at reasonable speed.

The key performance metrics of interest are frame rate and latency, both of which our system aims to improve over conventional warp-less rendering. However with image warping, there are two different ways to measure frame rate and latency. Because only particular types of motion are compensated when producing warped frames, the compensated and uncompensated motion will each appear to have their own distinct frame rate and latency. The specifics of this effect are described in the subsequent sections .

In our evaluation we only consider the software-introduced components of frame rate and latency, ignoring contributions that are impossible to eliminate via software. For example, in practice the usable frame rate is constrained by the display's refresh rate, while additional latencies are introduced by frame scan-out and pixel response.

Quantitative performance values were measured on a desktop computer including an Intel i7-2600[1] CPU, an Intel HD Graphics 2000 iGPU, 8GiB of 1333MHz dual-channel DDR3 RAM and an NVIDIA GTX 580[2] dGPU connected over a 16-lane PCI Express (PCIe) v2 bus, with all components at stock speed. While the CPU and dGPU may be high-performance components, the following sections explain how we believe them to have little impact on warping performance.

Tests were conducted by embedding our image warping system into a test application and measuring the performance during runtime. The chosen application was the *Deinterleaved Texturing* sample from NVIDIA's GameWorks D3D Samples[3]. It was chosen because the source code was simple to integrate with, while still being demanding on the dGPU.

[1]http://ark.intel.com/products/52213
[2]http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-580
[3]https://github.com/NVIDIAGameWorks/D3DSamples

Except where otherwise specified, the load was adjusted to give an application frame rate of around 30 frames per second (FPS). The type of warping algorithm used is the simple frame reprojection described in our previous work [5, 6] which is able to compensate for camera rotation and — to a small degree — translation. All rendering and warping was done at 1920 by 1080 resolution.

4.1 Latency

In 3D applications with a conventional rendering pipeline, the component of latency introduced by software is the difference in time from when the application logic samples and processes a user's input, to when the first frame that incorporates that input begins to be scanned-out to the display. We refer to this as application latency, and it exists under the same definition within our warping system.

As a result of warping, a second method of measuring latency is introduced which we call warp latency. Warp latency is the difference in time between the warping subsystem sampling a user's input and when the corresponding warped frame begins scanning-out. It is important to note that any warping system will only sample a subset of the input types that the application does.

The effect of this split is that the delay a user sees when interacting with the application depends on the particular type of interaction. For example, the perceived latency of changing view direction might be 10 ms, while that of moving the player character might be 100 ms.

4.1.1 Warp Latency

In our test set-up warp latency follows a bimodal distribution with modes at 4.2 ms and 6.8 ms. These latencies are consistent and predictable, making motion prediction accurate and reliable.

The existence of two modes is due to a fraction of warp frames being slowed due their rendering including an extra GPU tile format conversion. The fraction of frames including this conversion is equal to the application frame rate divided by the warp frame rate. The delay caused by this conversion may be avoided by extra iGPU synchronisation: at small cost to warp frame rate; or by doing the conversion asynchronously on the CPU using direct resource access (DRA): which is only supported on some iGPUs and at cost to CPU utilisation.

Warp latency is dominated by the time it takes to render the warp on the integrated GPU, which consistently made up 3.4 ms of the latency in all frames. Warp render time is influenced by the warping algorithm used, the iGPU hardware performance and the resolution at which warping is being performed.

4.1.2 Application Latency

Application latency is, as expected, slightly degraded over rendering without warping. With our configuration, application latency was between 45.2 ms to 49.5 ms, in contrast to 34.9 ms for warpless, synchronised conventional rendering. The increase in latency due to warping is therefore between 10.3 ms to 14.6 ms depending on the phasing of rendering and warping.

The increase to application latency is due to the various overheads introduced by warping. In conventional rendering, rendered frames may be scanned-out as soon as they finish rendering, while in our warping system they must additionally:

1. Be copied from the dGPU into CPU-addressable memory in linear memory order.
2. Be copied between the buffers owned by the different D3D devices.
3. Be copied into tiled memory order in iGPU-addressable memory.
4. Wait for the correct time to warp.
5. Wait for the warped frame to render.

The separation of items 1–3 is necessary as the D3D11 API does not allow direct copies between different different physical GPUs. Figure 2 helps to illustrate the copies involved in this process.

4.2 Frame Rate

In conventional real-time 3D applications, frame rate is the sustainable frequency at which the application is able to render complete image frames. Image warping complicates things, as some image frames are produced by application rendering, while others are produced through warping. Our evaluation considers each frame type independently: resulting in both an application frame rate and a warp frame rate.

4.2.1 Application Frame Rate

As with application latency, application frame rate is also slightly degraded due to the overhead associated with warping.

In contrast to latency however, the only applicable sources of overhead are the time taken to copy finished frames out of dGPU memory, and the optional anti-buffering synchronisation. For our test system, the copy accounted for 1.5 ms of overhead while synchronisation accounted for 1.9 ms.

The copy consists of reading a frame out of the tiled format on the dGPU, copying it over the PCIe bus, and writing it sequentially into shared system memory. The duration of copy may then be expected to scale with how much data needs to be copied (i.e. the frame size) and with the transfer speed (i.e. whichever bandwidth is smallest).

The 1.9 ms of synchronisation overhead is due to our system's greedy approach to copying finished application frames. This value is determined by the time between the application's last render issue for a frame, and the first render issue for the next frame. As such, it is the most implementation-dependent performance metric in our system. Our system may also be configured with a lazy synchronisation approach which eliminates this overhead at the expense of increasing application latency (by the time it takes to render one application frame).

4.2.2 Warp Frame Rate

Due to the lightweight nature of warping, the warp frame rate is able to be significantly higher than what can be achieved through conventional rendering methods, even on modest hardware.

With our system, warp frame rate is essentially bound by the rate at which the iGPU can perform the warp process. The significant factors in this process are the upload of application frames to the iGPU and the time it takes for the iGPU to render the warp itself. Because frame uploads only occur once for each rendered application frame, the warp frame rate is therefore inversely proportional to the application frame rate. Our system prevents the application frame rate from exceeding the warp frame rate, forming a lower bound when they are equal. This occurs at 116 FPS for our test configuration which is already towards the upper end of common display refresh rates (60 Hz to 120 Hz). The limit of our testing showed warp frame rate increasing to 224 FPS when the application frame rate is reduced to 28 FPS

The frame upload penalty may be avoided by asynchronously uploading the application frames using the CPU using DRA. In this case, warp frame rate is no longer dependent on application frame rate, and is instead maintained at what would otherwise be its upper bound.

5. CONCLUSIONS

We have demonstrated that our dual-GPU asynchronous image warping architecture successfully circumvents the parallelism limitations imposed by the DirectX graphics API on computers running Microsoft Windows. Because of this, it is able to perceptually improve both frame rate and latency, in contrast to synchronous image warping systems which are only able to improve one or the other.

Our system does however provide less individual benefit to frame rate and latency compared to synchronous approaches due to the overhead associated with copying between GPUs. This takes effect as a smaller improvement to warp frame rate and latency, and as a larger degradation to application frame rate and latency.

Performance tests with a mid-range hardware configuration and synthetic load application resulted in worst-case warp frame rate of 116 FPS and warp latency of 6.8 ms: indicating the system is still highly beneficial despite its overheads.

The major hardware bottlenecks that influence warping performance are iGPU rendering speed, system RAM bandwidth and PCIe bandwidth. A simplification is that: iGPU rendering speed and system RAM bandwidth influence warp latency and frame rate, while PCIe bandwidth influences the amount of application frame rate degradation.

The system's CPU to GPU synchronisation may be configured to prioritise either application frame rate or latency. Alternatively, the application developer may perform the synchronisation themselves in order to achieve maximum hardware utilisation and efficiency.

Several performance bottlenecks may be eliminated through direct CPU access to iGPU memory buffers, a non-standard feature that is starting to become available. This allows one buffer copy to be avoided and also allows another copy to be performed in parallel with warped frame rendering.

6. FUTURE WORK

A significant limitation in trying to perform timing-sensitive computation entirely in software is that the reliability of the timing is influenced by several non-deterministic factors. These include the usage of system resources by other processes, and the scheduling of tasks by the operating system. While we observed satisfactory consistency in our evaluations, the variability of background conditions may disrupt our system in more heavily loaded systems. These issues may not be completely avoided through software, but the degree to which they cause problems should be identified to fully determine the suitability of software-based warping.

Another point for future research is regarding resource contention with the warping. The major resources contended by both warping and other systems are memory bandwidth and CPU cache occupancy. Early results from our testing confirms that iGPU warp performance can be affected by other CPU tasks, but we have yet to establish the degree to which this affects performance.

References

[1] D. Andreev. Real-time frame rate up-conversion for video games: Or how to get from 30 to 60 fps for "free". In *ACM SIGGRAPH 2010 Talks*, SIGGRAPH '10, pages 16:1–16:1, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0394-1. doi: 10.1145/1837026.1837047.

[2] H. Bowles, K. Mitchell, R. W. Sumner, J. Moore, and M. Gross. Iterative image warping. *Computer Graphics Forum*, 31(2pt1):237–246, 2012. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2012.03002.x.

[3] W. R. Mark. *Post-rendering 3 D image warping: visibility, reconstruction, and performance for depth-image warping.* PhD thesis, Citeseer, 1999.

[4] Oculus VR. Oculus rift - virtual reality headset for 3d gaming, 2012. URL `http://www.oculusvr.com/`.

[5] E. Peek, B. Wünsche, and C. Lutteroth. Image warping for enhancing consumer applications of head-mounted displays. In *Australasian User Interface Conference (AUIC 2014)*, volume 150 of *CRPIT*, pages 47–56, Auckland, New Zealand, 2014. ACS.

[6] E. M. Peek, B. C. Wünsche, and C. Lutteroth. 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*, 2013.

[7] F. Smit, R. Van Liere, S. Beck, and B. Froehlich. An image-warping architecture for vr: Low latency versus image quality. In *Virtual Reality Conference, 2009. VR 2009. IEEE*, pages 27–34, 2009.

[8] F. A. Smit, R. van Liere, and B. Fröhlich. An image-warping vr-architecture: design, implementation and applications. In *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, VRST '08, pages 115–122, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-951-7.

[9] Valve Corporation. Steam hardware & software survey: July 2013, July 2013. URL `http://store.steampowered.com/hwsurvey`.