

# EFFECTIVELY EXPLOITING A PARALLEL ALGORITHM FOR VIRTUAL REALITY SCENE GENERATION

RONALD POSE

*School of Computer Science and Software Engineering,  
Monash University,  
Clayton, Victoria 3168, AUSTRALIA  
E-mail: rdp@cs.monash.edu.au*

**Abstract.** Virtual reality and other immersive interactive computer graphics systems are often considered as high performance computer graphics systems without regard to the application itself. Virtual reality systems apply highly parallel graphics rendering engines in an attempt to achieve desired performance. Sadly, even state-of-the-art parallel graphics techniques provide inadequate performance. Exploiting parallel graphics technology is sound but with virtual reality much of the computation performed is valid for a very short time, and then has to be recomputed. The *Priority Rendering* algorithm avoids this problem and allows the effective exploitation of parallel graphics rendering engines.

## 1 Introduction

Virtual reality and other immersive computer graphics applications use high performance computer graphics rendering engines, often employing parallelism in both hardware and software. Even so, their performance is inadequate for low-latency virtual reality systems. Either motion fluidity or image quality usually has to be sacrificed to maintain real-time interactive performance.

The problem is that to meet desired performance parameters, one has to generate over 60 photo-realistic frames per second, each of over 1,000,000 pixels, and display them on a head-mounted display worn by the user. The system tracks the user's head in real-time to determine the view of the virtual world to show the user. Only then can the graphics rendering be performed. Delays in generating the images are perceived as a lag in response. i.e. the user moves his head and has to wait for the viewed world to catch up. This latency can lead to serious consequences such as motion sickness and disorientation.

Using multiple high performance graphics pipelines is difficult since the large graphics pipeline delays themselves may be noticeable. Parallelism has to be applied effectively to solve the problem; a brute force graphics approach is insufficient. The problem is such that it is possible to overload the available graphics performance, so the aim must be to provide the best possible user experience, given limited computational resources.

We first outline the conventional approach. Then show how this does not perform so well in virtual reality applications. A new way of thinking about the original problem leads to a different approach. The key is to consider the system as a whole rather than treat it as just another high performance computer graphics system. Our new conceptual approach is called *Delayed Viewport Mapping* and its

realization in the form of an *Address Recalculation Pipeline* (ARP) involves a great deal of parallelism in its implementation [1]. A technique of exploiting the new architecture has also been developed [2] that can also have a parallel implementation involving both hardware and software. This technique, *Priority Rendering*, effectively exploits parallel rendering in the virtual reality domain, and is described here with an emphasis on parallelism in its implementation.

## 2 Conventional approach to high performance computer graphics

Computer graphics involves much computation. With so many steps involved there is an obvious case for some kind of pipelining. Both software and hardware pipelining is exploited in high performance graphics systems.

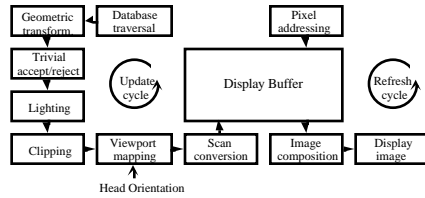
One approach is to subdivide the job based on objects in the virtual environment. i.e. use separate graphics rendering engines in parallel for each graphical object or groups of objects. Such a technique has been used successfully in animation. Various objects and background scenes are drawn separately on transparencies and can then be overlaid to give the desired composite scene. Apart from the possibilities for parallel creation of the images with this technique, it is also possible to move and animate objects within a scene without redrawing the whole scene. This is called image composition. In its simplest form the images not only have X and Y screen coordinates, but also depth, Z coordinates associated with each pixel. When combining images the system will choose the pixel at a particular screen coordinate with the lowest Z value, i.e. the one closest to the viewer. Apart from having parallelism within the graphics renderer, we can also run many renderers in parallel using image composition to form the final viewable image, and reduce the amount of graphics actually rendered by re-using previously rendered background scenes etc.

This appears to be attractive for virtual reality. The background scene is often fairly static; changes are restricted to a few objects which are the current focus of attention. However this assumes that the effective camera angle is fixed, or in the virtual reality case, that the viewer's viewpoint is fixed. That is not the case.

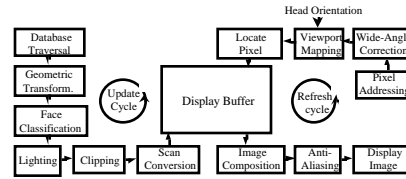
If the user moves his head, the virtual environment has not changed in itself, but what the viewer expects to see is now different. Even a slight change of head orientation invalidates every pixel in the display. So virtual reality systems using conventional, albeit parallel graphics techniques, end up having to regenerate all the graphics. Note even the background appearance has changed and is recalculated.

Figure 1 depicts a conventional computer graphics system as might be employed for virtual reality. It comprises two distinct processes communicating via the display buffer. On the left is the computer graphics software and hardware. The output of which is an image composed of pixels that are placed in the display buffer. On the right is a conventional raster display that reads pixel values from the display buffer and sends them to the display device. The raster display section runs at a constant speed, typically between 60 and 100 Hz. display refresh rate. On the other hand, the computer graphics section typically runs slower with its

performance dependent on the complexity of the graphics it is rendering. It is not unusual to find high performance virtual reality systems which struggle to get beyond 10 frames per second from their graphics renderers.



**Figure 1.** Conventional Virtual Reality Display Architecture.



**Figure 2.** Delayed Viewport Mapping using an Address Recalculation Pipeline

Image composition can also be used, albeit not effectively, allowing an obvious form of parallelism. Multiple graphics renderers (the left side of figure 1) combine their outputs before feeding the resulting image into the display buffer.

### 3 Re-examining the virtual reality problem

Our difficulties with virtual reality applications have a cause clearly indicated in figure 1. The user's head orientation is input to the graphics update cycle. Thus every change in user head orientation, no matter how slight, causes all the graphics in all graphics rendering engines, to be recalculated. This negates the advantages of image composition where only those images that have changed will be re-rendered.

The virtual world may be completely static, yet all the graphics are being re-rendered. What is required is a representation in which a static virtual world has a static computer graphics analogue that doesn't need continual re-rendering.

Imagine the world is a room with four walls, floor and ceiling made of glass through which you see the world beyond. A static virtual world is indistinguishable from one painted onto the walls, floor and ceiling.

Here is a model for a graphics system that meets our requirements. Instead of a screen in front of the user's eyes, we place the viewer in a room with graphics displays completely surrounding him. That approach is taken in virtual reality *CAVEs*. We adapt that to head-mounted displays by selecting the parts of the image surrounding the user that should be displayed in front of his eyes. We call this technique *Delayed Viewport Mapping*, since we have delayed selecting the viewer's viewpoint until after graphics has already been rendered. We have also devised a highly parallel hardware implementation called an *Address Recalculation Pipeline (ARP)*[3]. The parallelism in the ARP has also been investigated [1].

### 4 The Address Recalculation Pipeline

The advantages obtained through the use of *Delayed Viewport Mapping* are not free. The easiest way to picture the ARP's operation is to begin with the viewer's eyes

and trace a path through to the encapsulating display memory that mimics the glass room we imagined earlier.

Between the viewer and the head-mounted display is a viewing lens with some distorting characteristics that spread the image so as to cover the user's full field of view. We draw a vector from the eye through a pixel of the display device, have it bent to mimic the lens distortion, and finally have it intersect the encapsulating cubic display memory surface. That point of intersection identifies the pixel in the display memory that really should be on the display device at that point. Given the viewing lens is fixed, the bending of the vectors can be described by a lookup-table. The information from the head tracker takes the form of a rotation matrix by which we can multiply the vector, thus rotating it to the viewer's current orientation.

The ARP is implemented as a pipeline with the first stage being the lens distortion correction lookup-table. The second stage doing matrix multiplication to re-orient the vector so it is aligned with the user's current viewpoint. The third stage determines the intersection with the encapsulating display surface, locating the required pixel. Note the output of the third stage is an address in the display memory, hence the name *Address Recalculation Pipeline*.

With a display device with over a million pixels and a refresh rate around 100 Hz, the ARP second stage is doing over 100,000,000 matrix multiplications per second. The third stage involves divisions that are expensive computationally.

The new system using an ARP is depicted in figure 2. The ARP is in the top right of the figure. Note the head orientation feeds into the ARP section, not into the graphics rendering section on the left of the figure. The graphics is now independent of the viewing orientation of the user. With viewport independence we now re-introduce image composition. If parts of the virtual world appear to change, only those graphics renderers handling the changing objects are invoked. Image composition hardware merges these objects with the existing static scenery.

In our implementation the parallel rendering engines each have their own display memory. i.e. the left hand side of figure 2 is replicated. A single ARP drives all the display buffers. The display buffer outputs go through image composition hardware pipeline stages, and an image smoothing stage, before the final image is displayed. The real picture is somewhat more complex [3,4].

The ARP is easily implemented with cheap, parallel hardware. Such computation is easier than complex graphics calculations. It does require hardware since the volume of computation would severely tax a software implementation.

## 5 The problem of translational movement

Translational movement of the user within the virtual environment results in relative movement between the user and the objects in the virtual world that is observable to the user. The user's own movement has the most dramatic effect, since it affects the display of even static parts of the virtual world.

When translating, objects in the world appear to move relative to you. Close objects appear to move more than distant ones. Very distant objects may not appear to move at all. Thus we have to concentrate especially on close objects.

Actually what is important is not the distance of the object from the user, but rather how long its graphics image will be valid. In the static situation this corresponds closely to the distance of the object from the user and its relative speed.

## 6 Priority Rendering

A scheme has been devised which bases the allocation of objects to display memories and hence to graphics rendering engines based on the expected validity times of their images. We call this *Priority Rendering*, since it in effect produces a prioritized list of objects in the order in which they should have their graphics images recomputed in order to maintain the illusion of virtual reality without the serious problems we set out to solve [2,5]. It not only produces a prioritized list, but indicates the rate at which the various objects should have their images updated.

Because the allocation to display memories is no longer based on depth within the scene, the hardware must maintain Z-values for each pixel and do image composition based on individual pixels' depths.

This method requires that some approximation of the validity time of objects' images be obtained. It need not and cannot be exact since it is impossible to predict if the user will move within the virtual world or whether some object in the virtual world will change or move, but it should be reasonably close. If one gets it wrong and under-estimates the validity time of an object's image, then it will end up being recomputed more frequently than required, wasting some of our processing capacity, but not visible by the user. On the other hand if one over-estimates the object's validity time, it will not be updated as often as it needs and will appear slightly out of date. However unlike traditional approaches to virtual reality, the virtual world as a whole is kept up to date, with perhaps a few objects not quite keeping up.

There is not space in this paper to provide the detailed analysis of *Priority Rendering* and its performance in various situations. We have found that we do an order of magnitude less graphics computing than a conventional system in dealing with translational movements, while still presenting identical images to the user. Human factors can also be considered [6]. Humans cannot translate or rotate at infinite speed. Applying human parameters we can reduce the rendering load by more than another factor of two. With that incorporated we are well over an order of magnitude better than a conventional system in handling translation in the virtual world. With our system we also get rotations for free in terms of graphics re-calculations, whereas in a conventional system rotations are the biggest computational load. So in effect we get identical images with several orders of magnitude less graphics computation, but requiring an ARP and its computation.

## 7 Conclusion

Using conventional graphics for virtual reality, parallelism can be employed, but not effectively. With *Priority Rendering*, and its *Address Recalculation Pipeline*, we have shown how to exploit effectively, parallel graphics rendering hardware.

The results are dramatic, with overall performance orders of magnitude better than the traditional approach. *Delayed Viewport Mapping* has enabled image composition to be used with effectively superlinear speedup.

The performance is due not to parallel graphics itself, but due to a radical new approach to the problem which transforms it into one requiring less computer graphics. Its new form employs parallel hardware in the form of an ARP and a new method of organizing the graphics computation, *Priority Rendering*, ideally suited to efficient parallel implementation

## Acknowledgements

Matthew Regan was responsible for much of this system. The prototype was funded by an Australian Research Council small research grant. Patenting of this technology was supported by a Monash Research Fund grant

## References

1. Pose, Ronald (2000) Parallelism in a Computer Architecture to Support Orientation Changes in Virtual Reality and Other Immersive Interactive Graphics Systems. In Proc. PART-2000, Sydney, Australia, November 2000.
2. Regan, Matthew and Pose, Ronald (1994) Priority Rendering with a Virtual Reality Address Recalculation Pipeline. Proceedings of SIGGRAPH 94. In Computer Graphics, Annual Conference Series 1994. pp. 155-162.
3. Regan, Matthew and Pose, Ronald (1993) An Interactive Graphics Display Architecture. Proceedings of IEEE Virtual Reality Annual International Symposium. (18-22 September 1993, Seattle USA), pp. 293-299.
4. Regan, Matthew and Pose, Ronald (1994) Display Memory Access Issues and Anti-Aliasing with a Virtual Reality Address Recalculation Pipeline. Proceedings of the 9th Eurographics Workshop on Graphics Hardware, SINTEF, Oslo, Norway, Sept. 12-13, 1994, pp. 23-27.
5. Pose, Ronald and Regan, Matthew (1998) Updating Graphical Objects Based on Object Validity Periods, US Patent Application No. 847,567, Replaces US Patent Application No. 307,330, Filed 24 April 1997, US Patent Number 5,841,439, Granted 24 November 1998.
6. Pose, Ronald and Regan, Matthew (1994) Techniques for Reducing Virtual Reality Latency with Architectural Support and Consideration of Human Factors. In Multimedia, Hypermedia & Virtual Reality, LNCS 1077 Springer-Verlag, 1996.