Priority Rendering with a Virtual Reality Address Recalculation Pipeline.

Matthew Regan, Ronald Pose

Monash University *

Abstract.

Virtual reality systems are placing never before seen demands on computer graphics hardware, yet few graphics systems are designed specifically for virtual reality. An address recalculation pipeline is a graphics display controller specifically designed for use with head mounted virtual reality systems, it performs orientation viewport mapping after rendering which means the users head orientation does not need to be known accurately until less than a microsecond before the first pixel of an update frame is actually sent to the head mounted display device. As a result the user perceived latency to head rotations is minimal.

Using such a controller with image composition it is possible to render different objects within the world at different rate, thus it is possible to concentrate the available rendering power on the sections of the scene that change the most. The concentration of rendering power is known as priority rendering. Reductions of one order of magnitude in the number of objects rendered for an entire scene have been observed when using priority rendering. When non interactive background scenes which are rendered with a high quality rendering algorithm such as ray tracing are added to the world, highly realistic virtual worlds are possible with little or no latency.

CR Descriptors: I.3.1 [**Computer Graphics**]: Hardware Architecture --- *Raster display devices*; I.3.3 [**Computer Graphics**]: Picture/image generation --- *Display algorithms*; I.3.6 [**Computer Graphics**]: Methodology and Techniques;

1 Introduction.

The recent popularity of virtual reality has placed extreme pressure on conventional graphics systems to provide realistic real time graphics. In order to maintain the illusion of immersion in a virtual world the user must continually see images from his own vantage point. Any delays between changes in the user's head position and the display of images from that position are very noticeable. To minimise this delay high update rates are required. Conventional film animation rates of 24 frames per second border on adequate. An update rate of 60 or more frames per second is desirable for a good immersion effect. This delay known as latency is one of the most noticeable and undesirable features of many systems.

In order to present the user with 60 updates per second extremely powerful rendering engines are required. These rendering engines draw the scene from the user's viewing position very quickly and present the image to the user. Much academic and commercial research has been done to devise systems which are capable of maintaining high rendering rates [1][4]. One of the reasons such powerful rendering engines are needed is that if a user wearing a head mounted display rotates his head the image changes. Within many applications it is conceivable that the biggest difference between successive frames is due to the viewport orientation. If it were possible to detach the user's head orientation from the rendering process it may be possible to reduce significantly the rendering loads. So instead of drawing the entire scene at 60 frames per second we could draw only the parts of the world that change very fast at 60 frames per second. Initial experiments on a sample world indicate that as little as 1-2% of the objects in a virtual world require updating at 60 frames per second.

A graphics display architecture which will detach user head orientation from the rendering process has been devised and is called an address recalculation pipeline[9][10]. This hardware graphics system performs viewport mapping after rendering and is fundamentally different to conventional display controllers. The location of the user's head does not need to be accurately known until nanoseconds before the first pixel of a frame is displayed on the output devices in the head mounted display unit. This means the lengthy rendering time and the double buffer swap time are removed from latency the user perceives during head rotations, greatly increasing the realism of the virtual world.

There are further advantages to using an address recalculation pipeline when used in conjunction with image composition[7]. This combination allows different objects within the virtual world to be rendered at different rates. Not all objects need to be rerendered at the maximum rate. Initial experiments on a virtual world indicate that many objects within the world only require updating 3.75 to 7.5 times per second, resulting in drastic reductions in rendering loads and overall system cost. Further the combination provides a high quality mechanism for renderer overload. When the renderers are overloaded the user still receives the images using the most up-to-date orientation information from the head tracker, and the latency to head

*Department of Computer Science, Monash University, Wellington Rd, Clayton, Victoria 3168, Australia.
E-mail: regan@bruce.cs.monash.edu.au

rotations remains minimal. Only fluidity of animation and motion through the scene suffer as a result of renderer overload. Stereoscopic latency is affected by renderer overload, however latency to stereo is often permissible[6].

The subject of this paper will be to examine priority rendering which is a technique for using an address recalculation pipeline with image composition to provide low latency and low rendering loads while maintaining a highly accurate representation of high quality virtual worlds, which are the goals of many virtual reality display systems. The object rendering load for a sample application environment will also be examined.

2 The Address Recalculation Pipeline.

An address recalculation pipeline uses hardware which performs orientation viewport mapping post rendering. That is, the orientation mapping occurs after a scene has been rendered rather than as the scene is being rendered. This removes the usually lengthy rendering time from the user perceived latency for head rotations. Some previous work on image warping post rendering has been done [2][12], however these algorithms are usually multiple pass algorithms and are often not directly applicable.

A major feature of the pipeline is that the update rate for user head rotations is bound to the update rate of the display device usually 60+ Hz, instead of the rendering frame rate. Also, with an address recalculation pipeline, the latency does not include the rendering time and doesn't include double buffer swap delays. The orientation of the view the user sees does not need to be known until the first pixel is to be sent to the display device. This means the images the user sees use the most up to date head tracking information. The nature of the latency to head rotations is depicted in Figure 1.

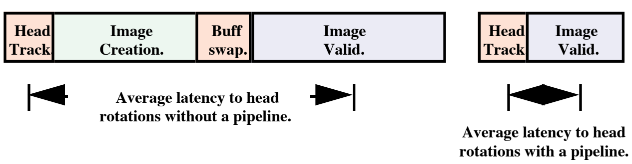


Figure 1: Latency to head rotations.

In order to perform viewport mapping after rendering, the rendered view must completely encapsulate the user's head, so when the user's head orientation changes the view from the new orientation has already been rendered and the address recalculation pipeline presents the user with the new view computed from the pre rendered image in the system's display memory. The surface of a cube was chosen as the encapsulating rendering surface after considering many possible candidates, as the cube results in a moderately simple hardware implementation and has fewer aliasing artefacts than most other surfaces. The cube's greatest advantage however is in its associated rendering simplicity. The surface of the cube may be rendered to by rendering to six standard viewport mappings. Thus most standard rendering algorithms require little or no modification. Figure 3 depicts an image rendered onto the surface of a cube and Figure 4 depicts a view of an image created by the address recalculation pipeline from the rendered image.

The architecture of the address recalculation pipeline differs from mainstream architectures in the nature of the pixel addressing mechanism. In a conventional display system, pixels to be displayed on the output device are fetched from the display memory sequentially. All adjacent pixels in the display memory appear adjacent on the display device. The address recalculation pipeline is different in that rather than fetching pixels sequentially, pixels are fetched from display memory based on the pixel's screen location, the distortion due to wide-angle viewing lenses and the orientation of the user's head. An overview of the pipeline is given in Figure 2.

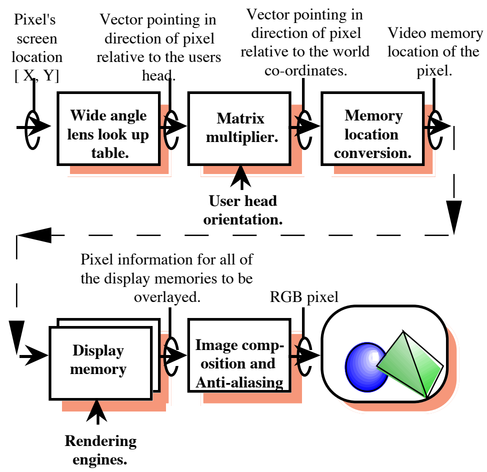


Figure 2: The address recalculation pipeline.

The pipeline runs at video display rates with the main high speed input being the x-y location of the pixel on the display device. The output from the pipeline is a stream of RGB pixels. The pipeline consists of multiple stages. Each stage performs a unique and necessary function. The first stage of the pipeline converts the screen location of a pixel into a three dimensional vector pointing in the direction at which the pixel is seen relative to the user's head as seen through the wide angle viewing lenses. The next stage of the pipeline multiplies this vector by a matrix containing user head orientation information. The output of the matrix multiplication stage is another three dimensional vector. This new vector points in the direction at which the pixel is seen relative to the world co-ordinate system. The third stage of the pipeline converts the three dimensional vector into a display memory location. Next, four adjacent pixels are fetched from the display memories. Finally the four pixel sets are composed and blended using redundant bits from the matrix multiplication stage. The resulting antialiased pixel is then sent to one of the output devices in the head mounted displays. The hardware is duplicated to obtain a stereo view.

On first examination it may appear that using an address recalculation pipeline would have six times the rendering overheads (scan conversion, clipping etc.) of a conventional system, however this is rarely the case and is only found if the scene has polygons evenly spread out in all three dimensional directions. The address recalculation pipeline must scan convert all polygons it receives which is the worst case scenario for a conventional system. Many conventional rendering systems are

Figure 3: Image in display memory.

Figure 5: A wide angle viewing lens distortion.

Figure 4: An arbitrary view created by pipeline.

Figure 6: Prototype Address Recalculation Pipeline board.

designed to cope with situations approaching the worst case scenario [8]. The rendering overheads for a conventional system may be reduced if the user is not looking at a complex part of the scene, however as the system has no control over the user's choice of direction for viewing it is fair to assume the user is looking at the most polygonally dense section of the world.

2.1 Hardware Implementation.

The current hardware implementation of the address recalculation pipeline uses 16 bit fixed point arithmetic for all computations. The prototype system is designed for medium resolution displays of the order of 640 by 480 pixels with a 60 Hz refresh rate which means each pipeline stage must complete in 40 ns. Many virtual reality systems incorporate wide angle viewing lenses [3][11]. Wide angle viewing lens distortion correction is achieved by means of a hardware look-up table. The wide angle viewing lens look-up table requires one 48 bit entry per display pixel. The resulting look-up table is 3 Mbytes in size. The system may accommodate many different wide angle lens types by down loading different distortions into the look-up table. A possible distortion is depicted in Figure 5. There is no run time rendering penalty for a distorting wide angle viewing lens.

The matrix multiplier stage is implemented with nine 16 bit by 16 bit commercially available multipliers and six 16 bit adders. Some additional 16 bit registers are used for pipeline synchronisation. The vector conversion stage which converts a three dimensional vector into display memory location requires six 16 bit divisions, some programmable logic and buffering circuitry. The divisions are implemented with a reciprocal table look-up followed by a multiplication.

The vector conversion stage produces a display memory location. The display memory itself is organised into six faces, where the faces logically fold to form a cube. Each face of the cube has a display resolution of 512 by 512 pixels. This display resolution results in 1.5 Mpixel display memories. Each display memory is Z buffered and double buffered with a private rendering engine. A stereo system employing multiple display memories for image composition requires vast amounts of memory. To implement high resolution display memories with current technology, static memory is used due to the speed requirements and the non-sequential nature of the memory access. As a result, the cost of display memory tends to dominate the cost of the system. This may change as new memory chip architectures become available. The hardware prototype of the address recalculation pipeline board is given in Figure 6.

3 Image Composition.

Image overlaying or image composition [7] is a technique often used to increase the apparent display memory bandwidth as scene from the renderer. Rather than having one display memory (or two for double buffering) the graphics has multiple display memories. Different sections of the visible scene may drawn into separate display memories then overlayed to form a final scene. In many implementations each display memory has a private rendering engine.

As pixels are being fetched from the display memory to be sent to the output device, all the display memories are simultaneously fetched from the same location. Next the Z value associated with each pixel is compared with the Z value of the pixels from the same location in the other display memories. The pixel with the smallest Z value is the winner and is the one that is sent to the output device. Figure 7 illustrates the concept of image composition. Note how one image may cut into another.

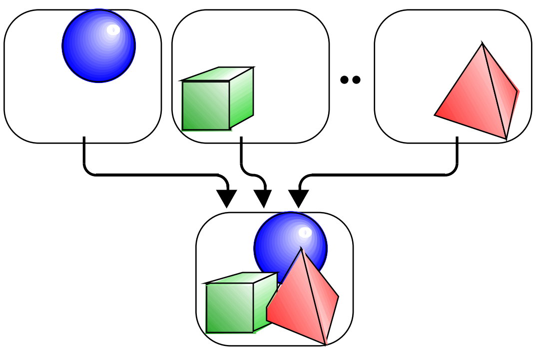


Figure 7: Image Composition.

In a conventional graphics system image composition uses a factor *n* redundancy to provide a factor *n* increase in performance. A side effect of image composition is that each of the display memories may be updated individually and at different rates. In a virtual reality system using image composition alone this side effect is generally of no use as all of the images in the display memories are rendered with the same fixed viewport mapping so when the viewport changes due to a user head rotation all of the images have an incorrect viewport mapping and need re-rendering. A factor *n* increase in speed is all that may be achieved.

Using an address recalculation pipeline it is possible to make effective use of this side effect of image composition to achieve in certain cases much better than factor n improvement for n display memories in a virtual reality display environment. This is because the images in the display memory of a graphics system with an address recalculation pipeline do not necessarily become invalid when the user's head orientation changes, thus the length of time an image in display memory is valid only loosely depends on the orientation (for a stereo view). For example a non interactive background may never require re-rendering and may thus be pre-rendered with great detail using a high quality rendering technique and a complex model.

4 Priority Rendering.

Using an address recalculation pipeline it is possible to render a scene which is largely independent of the user's head orientation. When image composition is combined with the address recalculation pipeline it is possible to render different parts of a scene at different rates. This paper examines a how virtual world may be subdivided into different rendering rates and the effect this has on the overall rendering efficiency.

The orientation independent sections of a static scene that change the most tend to occur during user translations. When the user is stationary within the scene, the renderers must only maintain stereoscopy (which is a form of translation) and animate objects which are changing themselves or change as a result of interaction.

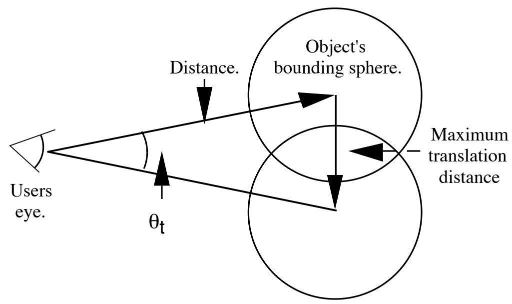
Priority rendering is demand driven rendering. An object is not redrawn until its image within the display memory has changed by a predetermined threshold. In a conventional system this strategy would not be effective as almost any head rotations would cause considerable changes to the image in display memory and the system would have to re-render everything. The images stored in the display memory of a graphics system with an address recalculation pipeline are to a great extent independent of user head orientation which means the renderer doesn't have to redraw the entire scene for head rotations.

The threshold for determining when an object has changed by more than a tolerable amount is determined by the designer of the virtual world and may typically be based on several factors. Usually this threshold is in the form of an angle (θ_t) which defines the minimum feature size of the world. Ideally this angle would be less than the minimum feature size the human eye can detect, approximately one third of an arc minute, however in reality this is impractical. If anti-aliasing of the image in display memory is not used a more sensible threshold may be the inter-pixel spacing in the display memory and if no hardware anti-aliasing is used at all, the pixel spacing in the head set worn by the user may be used. Priority rendering attempts to keep the image in display memory accurate to within θ_t at the highest possible update rate.

In order to compute the image changes for an object contained within the virtual world we compute how much the object would have changed if it were static and then add an animation component unique to the object as required.

Consider what happens to the display memory image of a static object as the user translates relative to the object. The relative location of the image changes and the image itself may change in size. The rendering strategy must compensate for image changes within display memory. It is possible to predict when these changes occur by observing certain features of a sphere which encapsulates the object.

User translations cause objects to move within the display memory. In order to keep the scene accurate to within θ_t we need to know how long the image of the object will remain valid at the user's current relative speed. This time is known as the object's translational validity period ($\tau_{translation}$). (See Figure 8) Relative speed is used to compute the object's validity period rather than relative velocity as the resulting world would have several objects caught in slow display memories if the user changes direction significantly. This would result in large temporal errors in the locations of several objects. The relative speed must include a component for the eyes' speed relative to the centre of rotation of the head if the users head is rotating.

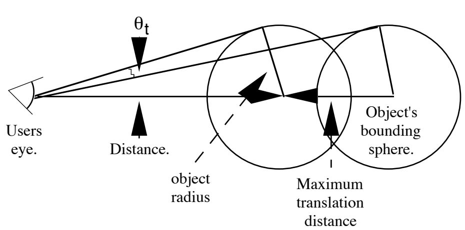


$$\tau_{translation}(i) = \frac{\text{distance(i) * sqrt(2*(1-cos }(\theta_t)))}{\text{relative\_speed(i)}}$$

Where i = object number.

Figure 8: An object's translational validity period ($\tau_{translation}$).

As the user moves towards or away from an object the size of the image of the object changes. We must compute the time that the size of the object is valid and re-render the object when its image size has changed by the predetermined threshold θ_t. Again we use the speed of the object relative to the user rather than the velocity of the object relative to the user for our computations for the same reasons as before. The period for which the size of the image is valid is known as the object's size validity time (τ_{size}). (See Figure 9) Note that as the image size changes by θ_t there may be several aliases of the object; these have been ignored.



$$\tau_{size}\ (i) = \frac{\text{distance(i) - radius(i)/sin}(\theta_t + \text{Asin(radius(i)/distance(i)))}}{\text{relative\_speed(i)}}$$

Where i = object number.

Figure 9: An object's size validity period (τ_{size}).

The last factor we consider is any requirement for animation by the object itself. For example a bird flapping its wings requires more updating than a static object like a stationary rock. The period of update for a specific object must be tagged to the object within the database and is defined by the virtual world designer. The period of the current frame of animation for a particular object is known as the object's animation validity time.

$$\tau_{animation} = \text{user\_defined}$$

Many other factors may be considered relevant for a highly accurate representation of the scene. For example object rotation as we translate has not been computed mainly because either size, translational or animation changes tend to dominate the required update rate.

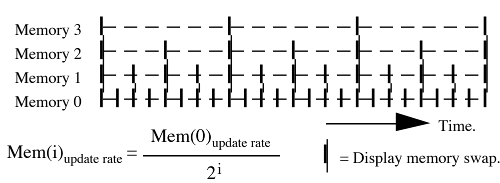
Finally we wish to determine the overall object validity period. This period gives us the amount of time we have until the next update of this object is required. This period also incidentally gives us the latency for a particular object, however, by definition the error in position of the object is less than θ_t. The overall object validity period $\tau_{overall}$ is defined as the smallest of the translational, size and animation periods. Obviously if $\tau_{overall}$ is less than the period of the maximum update rate, $\tau_{overall}$ is assigned the period of the maximum frame rate. This period defines the object's priority and the rendering power devoted to a particular object is based on this priority.

$$\tau_{overall} = \min(\tau_{translation}, \tau_{size}, \tau_{animation})$$

Accelerations have not been considered thus far, only relative speed. This may result in latency when accelerating as an object's computed validity period may not accurately reflect the actual validity period. Including acceleration into period computations is possible however the computation is made unnecessarily complex as high accelerations within a virtual world are limited as the sense of heavy acceleration may result in a form of motion sickness known as vection[5].

The previous discussions are based on being able to render all objects completely independently, this would require a pair of display memories per object (for a stereo view). As display memory pricing tends to dominate the overall system cost, providing one display memory per object is obviously impractical. An alternative is to have a limited number of display memories with fixed update periods and attempt to match objects with a particular display memory update rate. The display memory which has the highest update period which is less than the validity period of the object is chosen as the target display memory for a particular object.

Several strategies for dividing the overall system into a set of display memory update rates are possible and the optimal technique will ultimately depend on the nature of the virtual world. For our experiments we have chosen a set of update rates starting at the highest swap rate (for example 60Hz). All other swap rates are some exponential harmonic of the top rate. The display memory update swap strategy is depicted in Figure 10. Using this technique it is possible to swap an object from a low update rate into any of the higher update rates.



$$\text{Mem(i)}_{\text{update rate}} = \frac{\text{Mem(0)}_{\text{update rate}}}{2^i}$$

Figure 10: Display memory swap strategy.

The fastest display memory is swapping at 60 frames per second and all of the other display memories are swapping at some exponential harmonic of the top rate. The main reason for swapping on harmonics is so objects may be swapped to any faster update rate. The promotion or demotion of an object from one update rate to another occurs on the crossing of the harmonics, if objects are not swapped on harmonic crossings an object may not be represented or represented twice for a short period of time. This choice of exponential harmonics may not lead to maximum rendering efficiency (the number of times the object needs to be updated compared with the number of times the object is updated) and rendering loads across all display memories may not be distributed evenly. However the optimal configuration is based heavily on the nature of the scene, the rendering power available and the desired overload strategy.

The rendering hardware may have more display memories available than the virtual world requires for high efficiency. In this event, multiple renderers and display memories may be assigned to the one update rate thus devoting more hardware resources to a particular update rate, helping to balance the load.

The previous computations do not take into account stereoscopy. Fortunately the closest objects are most likely be in high speed buffers and it is these objects that are most affected by stereo updates. It may be possible to include a period factor which considers how far the head may rotate within a set period of time, however this is deemed unnecessary as some latency to stereoscopy is acceptable.

5 Experimental virtual environment.

Priority rendering may be used to reduce the overall rendering load on the rendering subsystem. The rendering load is based on several features of the scene, where the actual number of polygons is just one of the factors. One of our virtual world applications is a walk through of a forest and is the subject of this investigation. This simulation was performed in order to determine the rendering load on various display memories with various update rates.

The virtual world under investigation contained one thousand trees. Each tree is bounded by a sphere of radius five metres, which implies a maximum tree height of ten metres. The actual number of polygons contained within each tree is arbitrary as we are only considering object rendering load, the number of polygons per tree will eventually be determined by our real rendering power. The trees are randomly placed in a circular forest at a density such that the leaves of the trees may form a continuous canopy. The resulting forest has a radius of one hundred and fifty metres.

The simulation investigates the object rendering loads on various display memories with different update rates. The experiment is conducted as a walk through the world from one side to the other at one metre per second (approximate walking speed), passing through the centre of the world. This gives us statistics on rendering loads for circumstances ranging from being completely surrounded by the objects to being at the edge of the objects.
The chosen allowable error θ_t is the smallest inter-pixel spacing between the smallest pixels in the display memory. The system is to have a display memory resolution of 512 by 512 pixels per face, this means the smallest distance between any two pixels is approximately six arc minutes. This is an order of magnitude higher than the resolution of the human eye.

All of our comparisons are based on the number of objects that must be redrawn by the rendering system to maintain the approximately the same illusion with an effective update rate of 60 frames per second. We compare how many objects a system with an address recalculation pipeline must redraw against the number of objects a system without the pipeline must redraw for the entire length of the simulation (both systems are assumed to have multiple display memories and multiple renders). The Relative Object Rendering Load (RORL) is a percentage measurement of this ratio.

$$\text{RORL} = \frac{\text{Total number of object updates (with pipeline)}}{\text{Total number of object updates (without pipeline)}} * 100\ \%$$

With an address recalculation pipeline for the simulation described above the RORL is 15%. That is the system with the pipeline only had to redraw 15 objects for every 100 objects the system without the pipeline had to redraw for a similar illusion. If we make the minimum feature size (maximum error size) θ_t larger, the RORL reduces further. When θ_t is increased to be the smallest arc distance between the largest pixels in display memory the RORL is less than 8% of the object rendering load of a conventional system. Figure 11 depicts the relationship between the object rendering load (relative to a conventional system) and minimum feature size.

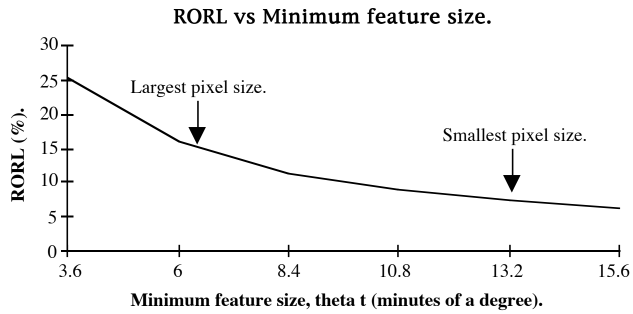


Figure 11: RORL against feature size.

The combination of the address recalculation pipeline with image composition and priority rendering has cut our total object rendering load to 15% of the equivalent object rendering load without the hardware. The main reason for this significant saving is depicted in Figures 12 and 13. Of the total number of object updates required for the walk through, nearly 40% of them were assigned to display memory 3 which is swapping at 7.5 frames per second. So even though display memory 0 is swapping 8 times faster than display memory 3, it is only doing one quarter the work display memory 3 is doing. This means memory 3 is updating 40%/10% * 8 = 32 times the number of objects display memory 0 is updating (at an eighth the rate).

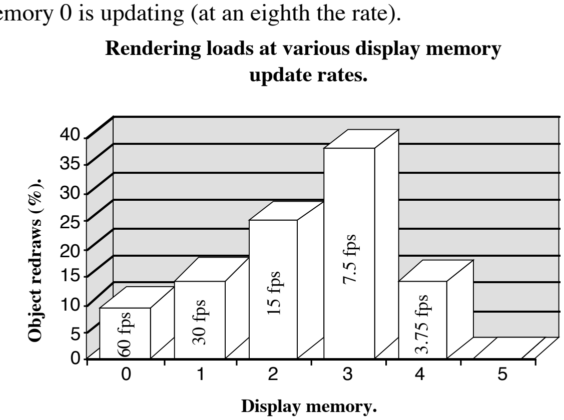


Figure 12: Graphical depiction of objects assigned to display memories.

White:	User.
Red:	Display Memory 0 (60 fps).
Cyan:	Display Memory 1 (30 fps).
Yellow:	Display Memory 2 (15 fps).
Green:	Display Memory 3 (7.5 fps).
Blue:	Display Memory 4 (3.75 fps).

Figure 13 Display memory assignment of trees during a walk through..

Providing a stereo view of the world is highly desirable within a head mounted graphics display system to help with the sense of presence within the world. With an address recalculation pipeline the display memories are not actually centred around the point of rotation of the user's head, rather they are centred around the user's eyes. This means when the user's head rotates while the user is stationary a small amount of translation occurs. This implies the need to re-render some objects which are affected by the translation caused by the head rotation. Although the speed at which the eyes translate during a head rotation must be included into the priority computation for $\tau_{translation}$ and τ_{size} it is interesting to note the total number of objects that become invalid to head rotations of various angles. Figure 14 shows how many objects become invalid for a particular head rotation. The upper line is when θ_t = 6 arc minutes (corresponding to the smallest pixel in display memory) while the lower line for when θ_t = 13 arc minutes' (corresponding to the largest pixel in display memory). From this graph we see that head rotations smaller than 45 degrees require few objects to be updated. These figures were generated whilst standing in the middle of the above mentioned scene.

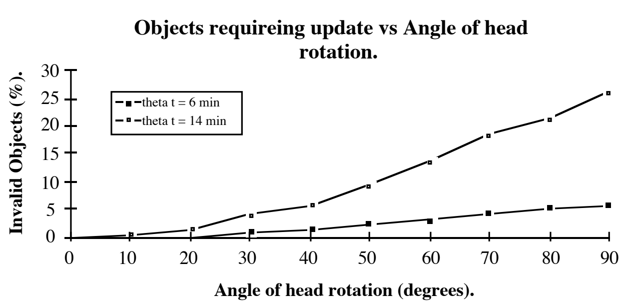


Figure 14: Invalid objects against Angle of head rotation.

6 Conclusion.

We have described a novel display architecture optimised for use in virtual reality systems. This architecture allows us to take advantage of a prioritized rendering technique which was the focus of this paper. Simulation studies have shown that these techniques can provide order of magnitude performance improvements over conventional graphics subsystems applied to virtual reality applications.

Using the address recalculation pipeline it is possible to reduce latency for head rotations to close to theoretical limits. In fact latency to user head rotations is imperceptible since these are handled within the display controller itself without need for re-rendering.

The use of image composition and parallel prioritized rendering techniques in conjunction with the address recalculation pipeline enables one to handle translation through scenes of great complexity without sacrificing frame rate or incurring unacceptable latency. Comparison with conventional approaches illustrated where the performance gains were taking place.

While the address recalculation pipeline itself can provide a significant advantage for virtual reality applications, its use in conjunction with multiple renderers and prioritized rendering techniques forms a significant advance in virtual reality implementation technology.

7 Acknowledgements.

Matthew Regan acknowledges the support of an Australian Postgraduate Award (Priority). This research was conducted under an Australian Research Council Small Grant. Datasets courtesy Iain Sinclair.

References.

[1] Akeley, Kurt. Reality Engine Graphics, Proceedings of SIGGRAPH 93. In *Computer Graphics*, Annual Conference Series 1993. 109-116.

[2] Catmull, Ed. and Smith, Alvy. 3-D Transformations of Images in Scanline Order. Proceedings of SIGGRAPH 80. In *Computer Graphics*, Annual Conference Series 1980. 279-285.

[3] Deering, Michael. High Resolution Virtual Reality, Proceedings of SIGGRAPH 92 In *Computer Graphics*, Annual Conference Series 1992. 195-202.

[4] Fuchs, Henry. et al. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor Enhanced Memories. Proceedings of SIGGRAPH 89. In *Computer Graphics*, Annual Conference Series 1989. 79-88.

[5] Hettinger, Lawrence and Riccio, Gary. Visually Induced Motion Sickness in Virtual Reality Systems: Implications for training and Mission Rehearsal, Presented at a DoD sponsored Inter agency Tech Simulation, 1-3 Oct, 1991.

[6] Lipton, L. Temporal Artefacts in Field-Sequential Stereoscopic Displays. Proceedings of SID '91 (Anaheim, California, May 6-10, 1991). In Proceedings of the SID 22 (May 1991), 834-835.

[7] Molnar, Steven, Image Composition Architectures for Real-Time image Generation. Ph.D dissertation, University of North Carolina, 1991.
[8] Molnar, Steven and Fuchs, Henry. Advanced Raster Graphics Architectures. Chapter 18, Computer Graphics, Foley and VanDam, 872-873.
[9] Regan, Matthew and Pose, Ronald. A Low Latency Virtual Reality Display System. Tech Report 166, Department of Computer Science, Monash University . September 1992.
[10] Regan, Matthew and Pose, Ronald. An Interactive Graphics Display Architecture. Proceedings of IEEE Virtual Reality Annual International Synposium. (18-22 September 1993, Seattle USA), 293-299.
11] Robinett, Warren and Roland, Jannack. A Computational Model for the Stereoscopic Optics for Head Mounted Display. In Presence 1, (winter 1992), 45-62.
[12] Smith, Alvy. Planar 2-Pass Texture Mapping and Warping. Proceedings of SIGGRAPH 87. In Computer Graphics, Annual Conference Series 1987. 263-272.

Appendix

Derivation of $\tau_{translation}$ and τ_{size}

For a given θ_t, the smallest maximum translational distance *x* of an object at distance *d* will occur when the distance after the translation is also *d* (ie. no change in size). This means we may use the cosine rule to compute *x* and then from *x*, given the objects relative speed we may compute $\tau_{translation}$.

$$x^2 = d^2 + d^2 - 2d*d*\cos(\theta_t)$$
$$x^2 = 2d^2(1-\cos(\theta_t)$$
$$x = d*\mathrm{sqrt}(2(1-\cos(\theta_t)))$$

$$\tau_{translation} = \text{maximum\_translation} / \text{relative\_speed}$$
$$\tau_{translation} = d*\mathrm{sqrt}(2(1-\cos(\theta_t)))/ \text{relative\_speed}$$

When an object changes size the smallest maximum translation distance occurs when the object moves closer. The maximum translational distance occurs when the angle to the edge of the object θ changes by θ_t .

$$\theta \text{ at position } 1 = \mathrm{Asin}\ (\text{radius}/d)$$
$$\theta \text{ at position } 2 = \mathrm{Asin}\ (\text{radius}/(d-x))$$

$$\theta_t = \theta_{(pos2)} - \theta_{(pos1)}$$
$$\theta_t = \mathrm{Asin}\ (\text{radius}/(d-x)) - \mathrm{Asin}\ (\text{radius}/d)$$
$$\sin(\theta_t + \mathrm{Asin}(\text{radius}/d)) = \text{radius}/(d - x)$$
$$x = d-\text{radius}/(\sin(\theta_t + \mathrm{Asin}(\text{radius}/d)))$$

$$\tau_{translation} = \text{maximum\_translation} / \text{relative\_speed}$$
$$\tau_{translation} = (d-\text{radius}/(\sin(\theta_t+\mathrm{Asin}(\text{radius}/d))))/\text{speed}$$