University of Wollongong

Research Online

Faculty of Informatics - Papers (Archive)

Faculty of Engineering and Information Sciences

1-1-2005

Large object segmentation with region priority rendering

Yang-Wai Chow
Monash University, caseyc@uow.edu.au

Ronald Pose
Monash University

Matthew Regan
Monash University

Follow this and additional works at: https://ro.uow.edu.au/infopapers

Part of the Physical Sciences and Mathematics Commons

Recommended Citation

Large object segmentation with region priority rendering

Abstract

The Address Recalculation Pipeline is a hardware architecture designed to reduce the end-to-end latency suffered by immersive Head Mounted Display virtual reality systems. A demand driven rendering technique known as priority rendering was devised for use in conjunction with the address recalculation pipeline. Using this technique, different sections of a scene can be updated at different rates, resulting in reductions to the rendering load.Further reductions can potentially be achieved by allowing for the segmenting of large objects. However in doing so a tearing problem surfaces, which has to be overcome before large object segmentation can be used effectively in priority rendering. This paper demonstrates a way of organizing virtual world objects for priority rendering, as well as a method to hide scene tearing artefacts due to object segmentation.

Keywords

region, rendering, priority, object, segmentation, large

Disciplines

Physical Sciences and Mathematics

Publication Details

Large Object Segmentation with Region Priority Rendering

Yang-Wai Chow, Ronald Pose, Matthew Regan

School of Computer Science and Software Engineering
Monash University
Clayton, Victoria 3800, Australia

yang.wai.chow@csse.monash.edu.au, ronald.pose@csse.monash.edu.au

Abstract

The Address Recalculation Pipeline is a hardware architecture designed to reduce the end-to-end latency suffered by immersive Head Mounted Display virtual reality systems. A demand driven rendering technique known as priority rendering was devised for use in conjunction with the address recalculation pipeline. Using this technique, different sections of a scene can be updated at different rates, resulting in reductions to the rendering load.

Further reductions can potentially be achieved by allowing for the segmenting of large objects. However in doing so a tearing problem surfaces, which has to be overcome before large object segmentation can be used effectively in priority rendering. This paper demonstrates a way of organizing virtual world objects for priority rendering, as well as a method to hide scene tearing artefacts due to object segmentation.

Keywords: Address recalculation pipeline, priority rendering, tearing, segmentation, region, warping.

1 Introduction

Latency is a major factor that plagues the designing of immersive Head Mounted Display (HMD) virtual reality systems. In order to maintain the illusion of reality, these HMD systems must continually display images from the user's point of view in real time.

Interactive rates of at least 60 updates per second based on the user's head orientation are necessary in order to achieve good immersion. Conventional graphics systems suffer from a noticeable end-to-end latency, a definition used by Meehan *et al* (2003) to mean the delay between a user's actions and when those actions are reflected by the display, because it is imperative in these approaches for the user's head orientation information to be known before rendering can be performed.

In order to shorten the latency for conventional systems, extremely powerful and fast rendering engines are required. Alternatively, realism of the virtual world can be sacrificed by reducing the level of detail in scenes,

thus decreasing the rendering load resulting in simpler scenes. However even with the fast graphics accelerators available today that can render over 100 frames per second (fps), the end-to-end latency still remains a factor to be contended with, because the update cycle is bound by the need to obtain up-to-date information regarding the user's head orientation prior to the rendering process.

A graphics display architecture known as the Address Recalculation Pipeline (ARP) has been designed in order to detach user head orientation from the rendering process (Pose and Regan 1994). This system implements a new display paradigm whereby viewport mapping is delayed until after rendering has been performed, so as to significantly reduce the overall user perceived latency. Using this approach the scene that surrounds the user's head is rendered, therefore the direction of view is only needed just before display, as the ARP maps the new view from the already rendered image of the scene sitting in display memory.

In conjunction with the address recalculation pipeline, a technique known as priority rendering has been devised to reduce the rendering load (Regan and Pose 1994). Priority rendering takes advantage of the fact that the complete scene encapsulating a user's head is rendered. In light of this, only parts of the scene that are changing from the user's perspective have to be re-rendered. From the user's point of view parts of a scene that change the most, thus require constant updating, will be parts that are located close to the user or dynamically animated parts. This is because upon the slightest movement by the user, sections of a scene located close to the user that were previously rendered to display memory will no longer be valid. However sections that are located further away from the user might still remain valid. In this manner, priority rendering reduces the overall rendering load by allowing different objects in a virtual world to be rendered at different update rates.

In order to fully utilize the reductions in rendering load offered by the priority rendering technique, it would be advantages if large objects in a virtual world could be segmented into smaller parts, this way different parts can potentially be updated at different rates. However such segmentation would introduce a tearing problem when the user translates through a scene containing segmented objects, where different sections of the same object are being updated at different rates. The reason for this is because the shared vertices originally joining the segments of an object together are now rendered at different update rates and will therefore be computed at different times, potentially resulting in tearing between or an overlapping of the polygons made up by these vertices.

This paper investigates a different way of organizing objects for priority rendering as well as a method to hide such tearing artefacts.

2 Background

This section provides some background to the address recalculation pipeline as well as previous work on priority rendering.

2.1 The Address Recalculation Pipeline

An address recalculation pipeline is a hardware architecture designed to reduce the end-to-end latency perceived by a user during head rotations. The pipeline accomplishes this by performing viewport orientation mapping after the rendering process, instead of the usual way which requires viewport mapping to be performed prior to rendering.

The delayed viewport mapping concept requires the scene that completely surrounds the user's head to be rendered. The surface of a cube was chosen to be the rendering surface surrounding the user's head. This specific shape was chosen over a number of other possible candidates, mainly because of its relative rendering simplicity. In particular the rendering surface of a cube contains six standard viewport mappings, each ninety degrees apart from the other. There are standard algorithms for cube surface rendering. An example of such rendering algorithms can be found in the use of a graphics technique known as cube environment mapping.

Viewport orientation mapping is then done only when the display device requests an update. The viewport mapping is then carried out based on the head tracking device's latest directional feedback, thus presenting the user with a new view computed from the image already rendered to display memory. In doing so the pipeline separates the viewport mapping from the rendering process, effectively decoupling the rendering and double buffer swapping times from user head rotational latency, as shown in figure 1.
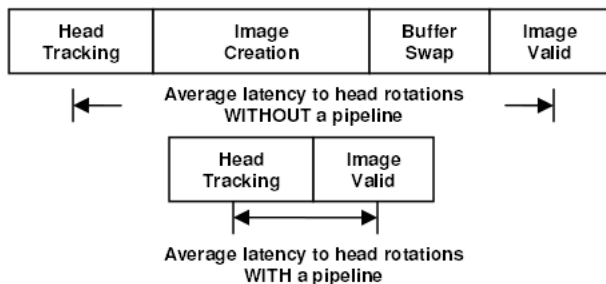
Head Tracking	Image Creation	Buffer Swap	Image Valid

Average latency to head rotations
WITHOUT a pipeline

Head Tracking	Image Valid

Average latency to head rotations
WITH a pipeline

Figure 1: Latency due to head rotations.

Due to the fact that the viewport mapping is performed by mapping relevant portions of the most up-to-date image already in display memory, display updates due to user head rotations are guaranteed an interactive response. Latency is now bound to the HMD unit's update rate and the time required to fetch pixels from display memory, and very much less dependent on the rendering frame rate. In that respect the system is therefore fairly independent of scene complexity, as compared to conventional systems where latency is tightly tied to the rendering frame rate.

Even if rendering frame rate were to momentarily drop drastically because the renderer cannot keep up with the demanded rendering load, the most recently rendered scene will still reside in display memory. Therefore the address recalculation pipeline can still display an image of the scene mapped based on the current user's direction of view, despite the viewing position not being up-to-date. On the other hand, upon renderer overload conventional systems will only be able to continually display the last rendered image which might be completely invalid, until a fully updated display image becomes ready, resulting in lengthy user perceived latency.

2.2 Priority Rendering

Image composition is a technique that has been used to accelerate computer graphics rendering (Molnar, Eyles, and Poulton 1992). The concept behind this technique is that a scene can be broken up into different sections, which can be rendered onto separate display memories. The final image can be obtained by compositing visible sections of images from the separate display memories, based on z-buffer comparisons, in order to form an image of the whole scene. This is depicted in figure 2.
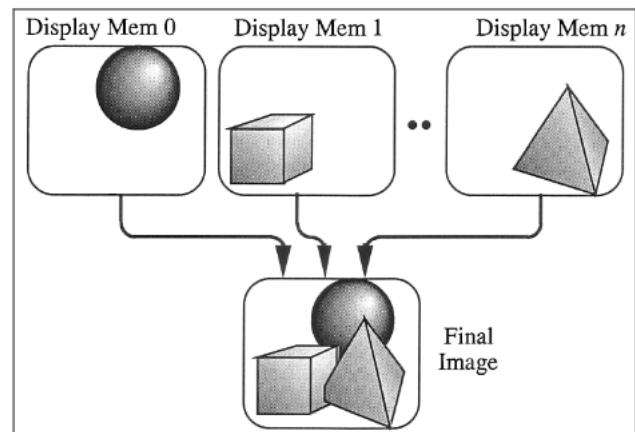

Figure 2: Image Composition.

Rendering time can be reduced by using multiple renderers in parallel when rendering to the multiple display memories. Alternatively if a section of the scene located on a particular display memory has not changed, that image already residing in display memory can be used without any need for alteration. Since the image in that display memory does not have to be refreshed, this reduces the overall rendering load.
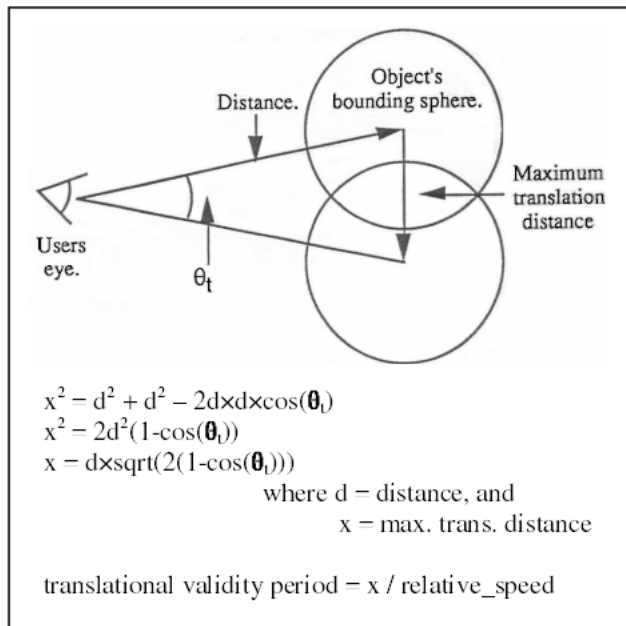
In the case of conventional virtual reality graphics systems, the later alternative does not apply very often as the images in display memories will become invalid upon the slightest movement by the user. However in the context of the address recalculation pipeline system where the scene surrounding the user's head has to be rendered, the image in display memory will still remain valid upon user head rotations. Therefore the ARP can utilize both these image composition advantages by using

multiple renderers and multiple display memories for updating at different rates.

This gives rise to a technique to be used together with the address recalculation pipeline, known as priority rendering. Priority rendering is demand driven rendering and is based on the concept that an object does not have to be updated until its current image in display memory becomes invalid. In a scene that surrounds a user, only dynamically animated objects might constantly be changing when the user rotates his/her head, static objects will remain the same. Even when a user translates through a scene, parts of the scene that are closer to the user will appear to change at a higher rate as oppose to parts that are located at a distance away from the user.

A threshold for establishing when an object requires updating has to be pre-determined for the purpose of priority rendering. In other words, if an object has changed by more than this threshold its current display in memory is no longer valid and therefore needs to be updated. In this manner, priority rendering therefore attempts to keep the image in a display memory accurate to within this threshold, which takes the form of an angle θ_t, at the highest possible update rate. A practical threshold might be the inter-pixel spacing in display memory or the inter-pixel spacing in the HMD.

In previous priority rendering experiments, display memory update rates were set to some exponential harmonic of the highest possible update rate (e.g. 60Hz, 30Hz, 15Hz, etc.). This method was used so that object swapping between display memories could be performed smoothly at fixed intervals and also as a precaution so that no objects would be duplicated on the different display memories. Objects in the virtual world were periodically assigned validity period estimates, these estimates were an indication of how long that object's image in display memory would remain valid.

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

where d = distance, and
x = max. trans. distance

translational validity period = x / relative_speed

Figure 3: Translational validity period estimation.

The validity period estimates were computed based on three components. It consisted of an animation component that had to be user defined, a translational component and a size component. The two later components were computed using the previously determined threshold, the object's distance from the user and the user's relative translational speed. The translational validity period was an estimate of how long the position of the object would remain valid, and the size validity period estimated how long the size of the object would remain valid, at the current user's relative translational speed respectively. Based upon these three components, the final or overall validity period estimate was determined as the minimum of the three values. Figure 3 shows how the translational validity period estimate's was calculated. It was found in experiments that unless the size of an object was extremely large (in the order of around ten times the distance from the user), the animation or translational validity period would tend to dominate the overall validity period computations.

With each object being assigned a validity period estimate, they were then sorted and allocated according to the length of these validity estimates, to the different display memories which had different update rates. Each object could be promoted or demoted, to higher or lower update rates depending on their computed validity estimates. Experiments have shown that priority rendering succeeded in reducing the overall rendering load (Regan and Pose 1994).

3 Experiment Framework

3.1 Large Object Segmentation

For the purposes of priority rendering, it is conceivable that a degree of further rendering load reductions can be achieved by segmenting large objects in the virtual world into smaller objects. Each of the smaller object segments can be treated as individual objects, through this way each individual object can potentially be assigned to a different update rate.

Reductions in rendering load are then possible because segments located further away from the user potentially remain valid in display memory for longer periods of time and therefore require less frequent updating. This compares favorably with the case where the whole object is treated as a single object, as even if a small section of that object is located close to the user the entire object has to be updated at the highest possible update rate. The trade off between reductions in rendering load is more appealing as opposed to the alternative of maintaining fewer objects in the virtual world, because in general the computational time needed to process more object segments is by far shorter than the rendering time required to continually render the whole object to display memory every single update cycle.

3.2 Tearing

The implementation of object segmentation will however introduce a potential tearing problem along the shared vertices between polygons of the same object that are being updated at two different update rates, whilst the user is translating through the scene. This is portrayed in

figure 4 from the user's viewpoint, the two polygons belong to the same object. At time *t*, both polygons were rendered onto separate display memories which had different update rates. At time *t+1*, the user has moved to a new location and polygon 2's display is refreshed based on the user's new position. However the display memory that polygon 1 is located on has yet to be updated, resulting in a discrepancy between the shared vertices, otherwise known as tearing. Note that overlapping would have occurred if the user translated in the opposite direction, but overlapping is not as obvious as tearing.

Such scene tearing, no matter how occasional, will completely destroy the illusion of reality that the virtual reality system attempts to present a user with. Therefore an effective solution to the problem is imperative before object segmentation can be used in priority rendering.
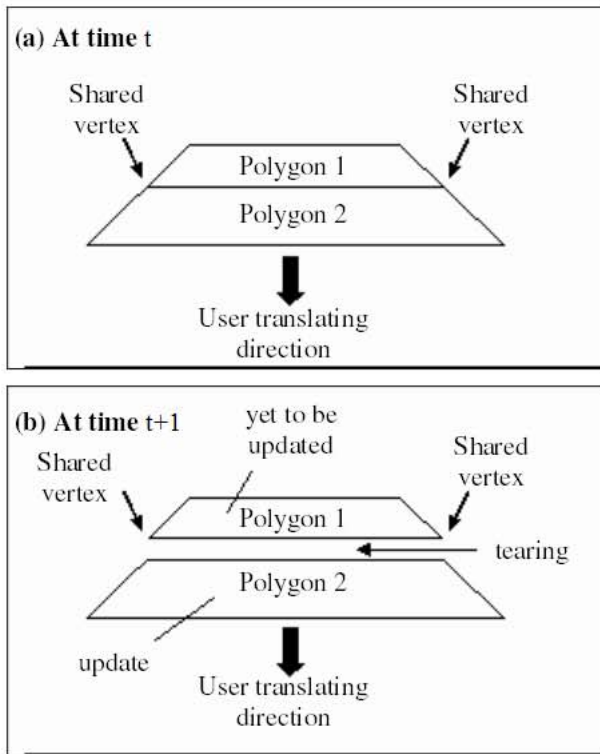

Figure 4: Tearing between the shared polygons' vertices of a segmented object.

4   Experiment Design

4.1   Region Priority Rendering

In this experiment, a different method of managing virtual world objects was implemented. The design of this implementation was based upon observations and analysis of previous priority rendering experiments.

Previous experiments showed that unless the size of an object was extremely large, the overall object's validity estimates were dominated by the animation or translational components of the equation. Since the key to this experiment was for the segmentation of large objects, size validity could then be omitted from considerations as this component would seldom or in fact never affect validity period estimates. The animation component that had to be user defined was also left out of the experiment

design considerations, so as to ensure the smoothest possible animations by automatically assigning any animated objects the highest update rate. This left only the translational component.
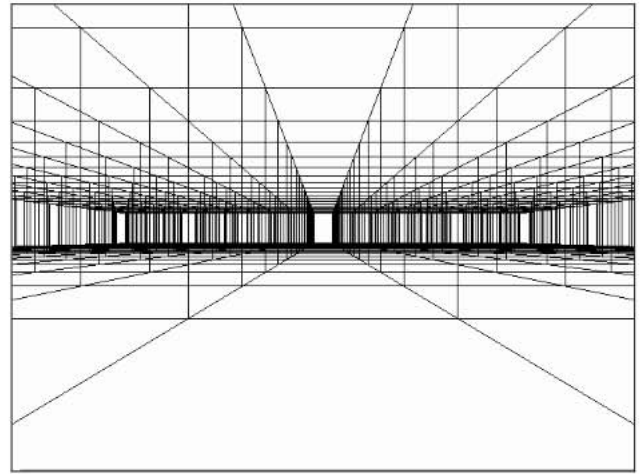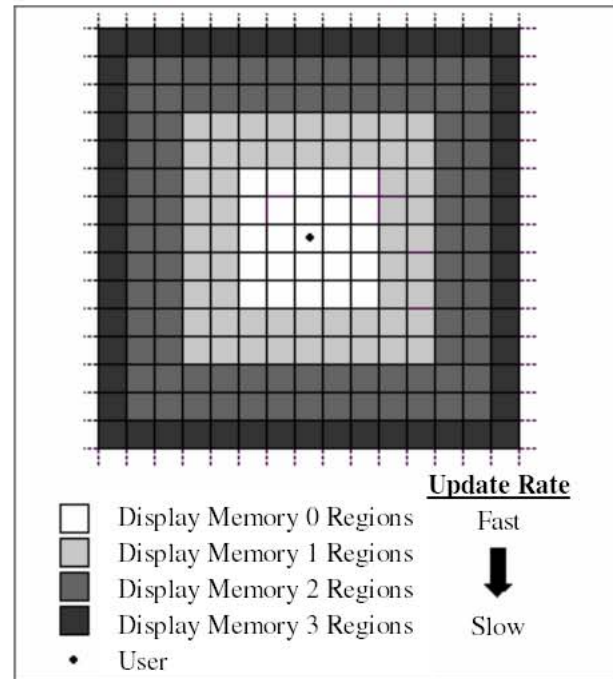

Figure 5: 3D view of the regions.

Figure 6: 2D top-down view of example region to display memory allocations.

Observations of previous work indicated that non-animated objects were assigned different update rates in clusters based on proximity, according to both the spatial and temporal locality principles. In other words, if an object was located some distance away from the user and was assigned a particular update rate, other objects within the vacinity of this object were likely to be assigned to the same update rate. Also, if an object was previously assigned to a particular update rate, it was likely to be re-assigned to the same update rate the next time around.

The design for this experiment was therefore to group objects into clusters or regions. The chosen region shape was that of a rectangular cube, with its base being a square and its height, the height of the virtual world.

Figure 5 illustrates a 3-dimensional example of the regions from a viewpoint somewhere in the world. Cubic regions are also a possibility if the height of the world is extremely high, for example, in flight simulations. These regions were to be equal-sized, and the specific shape was chosen because it can comfortably cover an entire virtual world without any leaving gaps or overlapping. Cubic regions have also been used successfully in many computer graphics techniques, for example its use in octrees (Samet and Webber 1988).

An inherent characteristic about cube regions is that they implicitly maintain relative spatial locality information. Instead of assigning all virtual world objects to display memories based on their individually computed validity estimates, the main idea behind region priority rendering was that all objects located in a particular region were assigned to a display memory according to the relative location of that region and the region where the user was located, as depicted in figure 6.

In this experiment, the separate display memories were allowed to be updated at different rates on a need-to-update basis, rather than fixing their update rates. In order for the display memories to know when their respective displays were no longer valid, information regarding the user's position when a particular display memory was last updated had to be stored. Upon each update cycle, these previous positions were compared with the user's current position. If the user had translated above a certain threshold from the time that display memory was last updated, then the image in display memory was no longer valid and the display memory would request for an update.
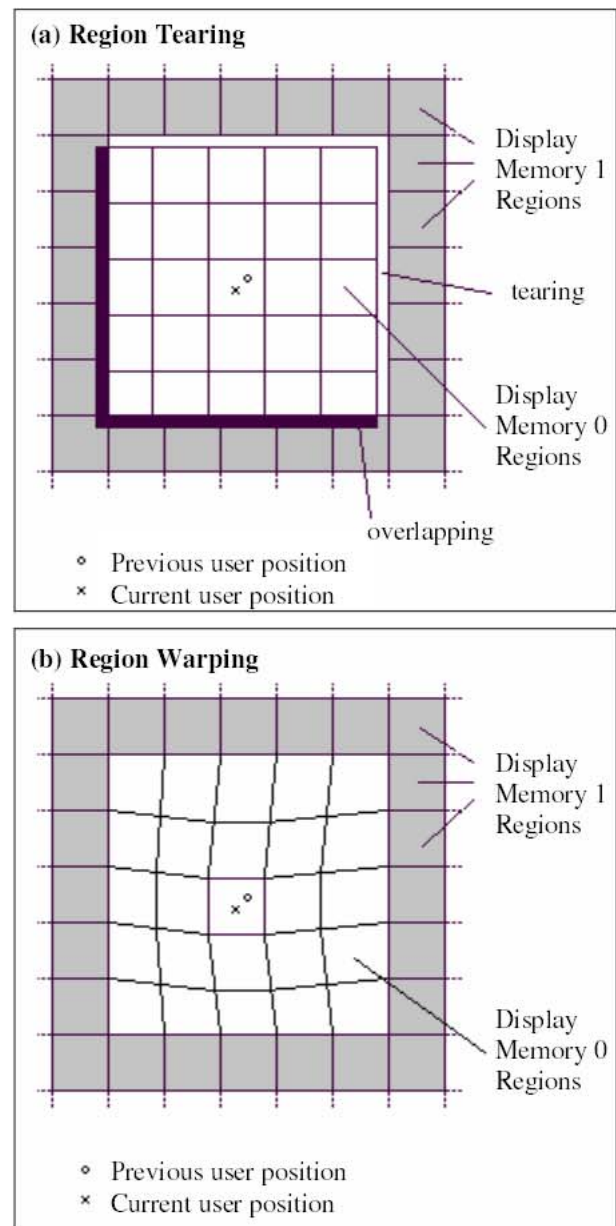
The update threshold was determined based on the translational validity period estimations (refer to figure 3). The maximum translational distance was used as the threshold to determine whether the display memory required updating. This method also catered for user accelerations and not just constant velocity translations.

In region priority rendering, the display memory swapping strategy adopted was to swap regions among the different display memories, after the user relocated to a different region, and upon the first update request of the display memory with the slowest update rate. Other strategies are possible depending on the nature of the virtual world, but the method used in this experiment avoided situations where objects in certain regions might not be represented or represented twice on two different display memories. Using this approach, each region could only be allocated to one display memory at any one time. This also meant that the user had to cross into the new region by a certain margin before the swapping of regions would be executed, thus preventing the potential problem of constant region swappings if a user were to constantly oscillate at the boundary between two regions.

4.2 Region Warping

In someways, much of the effort involved in region priority rendering is in dividing the virtual world into regions and also in the segmenting of objects. These computations however are to be pre-computed, and are

therefore not performed at run-time. Since the bulk of run-time computations previously required to calculate validity periods, sort and assign individual objects to different display memories is no longer necessary, this computational power can be redirected to solving the tearing problem.

(a) Region Tearing

Display Memory 1 Regions

tearing

Display Memory 0 Regions

overlapping

○ Previous user position
× Current user position

(b) Region Warping

Display Memory 1 Regions

Display Memory 0 Regions

○ Previous user position
× Current user position

Figure 7: Top-down, greatly exaggerated view of (a) region tearing (b) region warping.

A number of potential solutions were considered, from a pool of methods used by others for joining or filling in parts of scenes or images together such as in image stitching for panoramic scenes (Linhong and Hirakawa 2002, Li and Randhawa 2004), image quilting for texture synthesis (Efros and Freeman 2001), as well as methodologies aimed at filling in missing sections of scenes (Drori *et al* 2003). However, these image processing techniques were meant to be performed on still images and consequently rarely placed any importance on real time performance. Therefore the method chosen instead was one based on surface region warping/deformation as suggested by Milliron *et al*

(2002) in order to force regions to align. Applied to region priority rendering, this essentially meant slightly perturbing vertices across regions in order to force the shared vertices to line up, thus eliminating any form of tearing or overlapping.

The region methodology provided a criterion for large object segmentation. Using this methodology, large objects were segmented along the boundary lines separating the regions. In this manner tearing would be somewhat restricted and predictable, as compared to the case where large objects could be segmented anywhere causing tearing to take on a more random nature.

Initial experiments with segmentation showed that it was extremely difficult to completely compensate for such random tearing, because the location and size of each tear could only be obtained through a series of detailed, memory and time consuming computations. Moreover exact tearing location and size information was essential, as estimations were only able to reduce but not completely eliminate tearing. However by segmenting objects along region boundaries, the exact position and size of the tearing can be calculated. A greatly exaggerated depiction of this is represented in figure 7.

A vector of maximum perturbation can be calculated by simply subtracting the user's previous position from the user's current position. This is a 3-dimensional vector. All vertices along the edges of the higher update rate display memory are to be perturbed by the maximum perturbation vector. Other vertices are perturbed by some fraction of this vector depending on their distances from the edges.

The drawback of such structured segmentation is that unlike the case of random tearing, tearing will now occur at predictable sections and thus appear to be more obvious to the user. Nevertheless this is not a major concern, since the warping is meant to hide all forms of tearing anyway.

Experimental simulations of region priority rendering with object segmentation and region warping were carried out on a virtual world. The scenario used for experiments was that of a gallery. The reason for this choice lay in the human built architectural structure of a gallery. This way we could test the effects of tearing and region warping on a rigid well structured scene, which would not have showed up as clearly on a non-structured scene, due to the random characteristics of such a scene.

A fixed path through the scene, deemed to be the worst case scenario for the tearing, was chosen so that experiments along the same path could be exactly repeatable for the tearing case and the case with region warping. The user's translation speed used was 1 meter per sec, which is approximately walking speed. Every frame along the path was saved as an image at the maximum update rate (60Hz), in order to analyze the effects of the tearing and warping.

5 Results and Discussion

Some images and screenshots from the experiments are presented in this section.

Note that initial experiments were conducted on the full scene, shown in figure 14. However it was deemed necessary to remove some of the non-segmented objects from the scene as a number of them actually hid some of the tearing artefacts by covering them. Therefore in order to fully see the tearing and region warping artefacts, which was a major aim of the experiment, only the 'fixtures' of the scene were allowed to remain, as shown in figure 8.

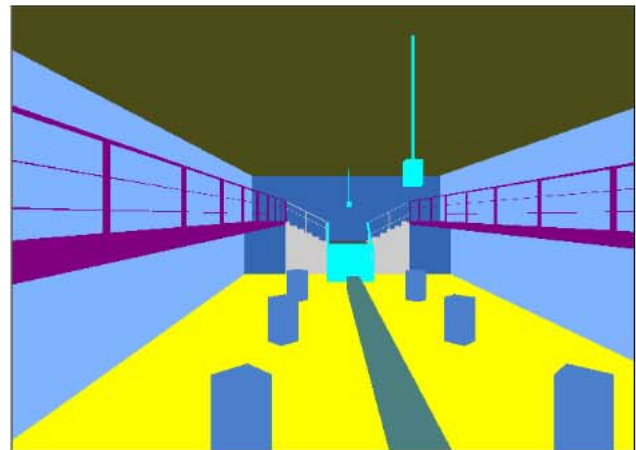**Figure 8: Scene from an arbitrary viewpoint.**
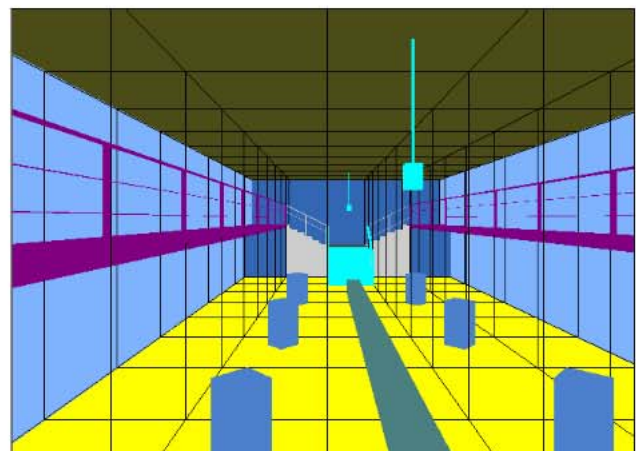

Figure 9: Virtual world objects.

Figure 10: Virtual world regions.

In figure 9 the individual non-segmented objects can be seen, as each object was drawn with a single distinct

color. Figure 10 shows how the virtual world was divided into regions, and in figure 11 the scene was rendered in wireframe in order to show the segmentation of virtual world objects.

Figure 12 shows the images rendered onto the different display memories at that particular position, for the cases with object segmentation and without object segmentation. The reductions in rendering load due to the segmenting of objects can be seen from this picture. The final composite image of these display memories is shown in figure 14. The results in table 1 and table 2 indicate the reductions in rendering load between the two cases. Take display memory 0 as an example, without object segmentation 160 object sections are rendered at the highest update rate, as compared to 87 object sections for the case with object segmentation. With object segmentation, the distribution of objects sections among the separate display memories reduces the amount of rendering demanded at 60 updates per second.
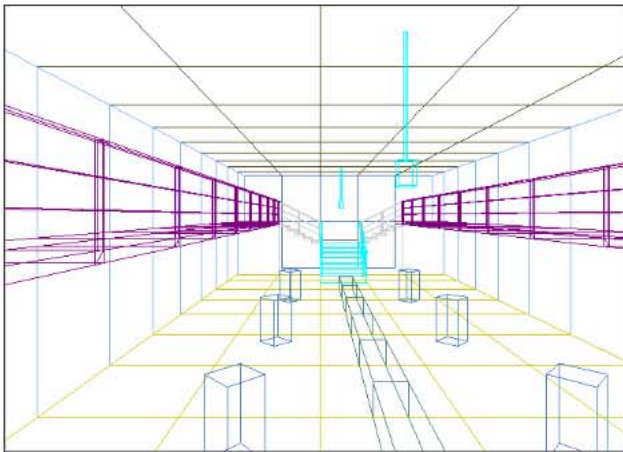

Figure 11: Virtual world object segmentations.

Display Memory	Average Update Rate (Hz)
0	60.00
1	39.25
2	28.21
3	16.15

Table 1: Average update rates for the different display memories.

Display Memory	Number of Rendered Object Segments	
	With Segmentation	Without Segmentation
0	87	160
1	44	8
2	20	3
3	62	42

Table 2: Number of rendered object segments for the different display memories shown in figure 12.

Although the results shown in table 1 and table 2 are not exact calculations of the reductions in rendering load, as the rendering time for each object section is different. Nevertheless, it does give evidence of the rendering load reductions achievable through the segmentation of large objects.

The tearing artefacts can be seen from the images shown in figure 13, these images were taken a few frames apart. The tearing appears as white in the frames because the frame buffer was cleared to white, therefore any discrepancies between the objects will show up as white in the scene. Figure 16 shows a close up of the tearing, and figure 17 in turn shows the effects of region warping, which successfully hides the tearing. As can be seen, the tearing is only a few pixels wide, therefore the warping of the region only stretches the regions by a few pixels (this can be seen from the enhanced difference image shown in figure 15, grey indicates no error) and will not be very obvious to a user translating through the scene.

6   Conclusion and Future Work

This paper has shown that reductions in rendering load can be achieved by allowing for the segmentation of large objects in conjunction with priority rendering. A method of organizing virtual world objects into regions for priority rendering as well as a way of hiding tearing artefacts, which result from the introduction of objects segmentation, in real time has also been described and shown.

In order to improve region priority rendering and region warping, future work will focus on measuring the exact computational load involved in region warping as well as developing a way to determine the optimal region size for a virtual world and its relationship to world size, user translating speed, priority rendering considerations, etc. Further experiments in order to measure the amount of distortions incurred in region warping will also have to be carried out.

7   Acknowledgments

8   References

Efros, A. and Freeman, W.T. (2001): Image Quilting for Texture Synthesis and Transfer. *In Proc. ACM SIGGRAPH '01*, Los Angeles, California, 341-346.

Li, J.S.J. and Randhawa, S. (2004): Improved Video Mosaic Construction by Selecting a Suitable Subset of Video Images. *In Proc. Twenty-Seventh Australasian Computer Science Conference (ACSC2004)*, Dunedin, New Zealand, CRPIT, **26**:143-149.

Linhong, Y. and Hirakawa, M. (2002): A Stitching Algorithm of Still Pictures with Camera Translation. *In Proc. First International Symposium on Cyber Worlds*, Tokyo, Japan, 176-182.

Meehan, M., Razzaque, S., Whitton, M.C. and Brooks, F.P. (2003): Effect of Latency on Presence in Stressful

Virtual Environments. *In Proc. IEEE Virtual Reality*, Los Angeles, California, 141-148.

Molnar, S., Eyles, J. and Poulton, J. (1992): PixelFlow: High-Speed Rendering Using Image Composition. *Proc. ACM SIGGRAPH '92, in Computer Graphics Annual Conference Series,* Chicago, Illinois, **26**(2):231-240.

Pose, R. and Regan, M. (1994): Techniques for Reducing Latency with Architectural Support. *In Proc. of East-West International Conference on Multimedia, Hypermedia and Virtual Reality,* Moscow, Russia, 153-160.

Regan, M. and Pose, R. (1994): Priority Rendering with a Virtual Reality Address Recalculation Pipeline. *Proc. ACM SIGGRAPH '94, in Computer Graphics, Annual Conference Series*, Orlando, Florida, 155-162.

Samet, H. and Webber, R.E. (1988): Hierarchical Data Structures and Algorithms for Computer Graphics II: Applications. *IEEE Computer Graphics and Applications.* 59-75.

Paul Bourke's Personal Pages: Texture Library. http://astronomy.swin.edu.au/~pbourke/texture/ Last accessed 4 July 2004.

Drori, I., Cohen-Or, D. and Yeshurun, H. (2003): Fragment-Based Image Completion. *ACM Transactions on Graphics*, **22**(3):303-312.

Milliron, T. Jensen. R.J. Barzel, R. and Finkelstein, A. (2002): A Framework for Geometric Warps and Deformations. *ACM Transactions on Graphics.* **21**(1):20-51.
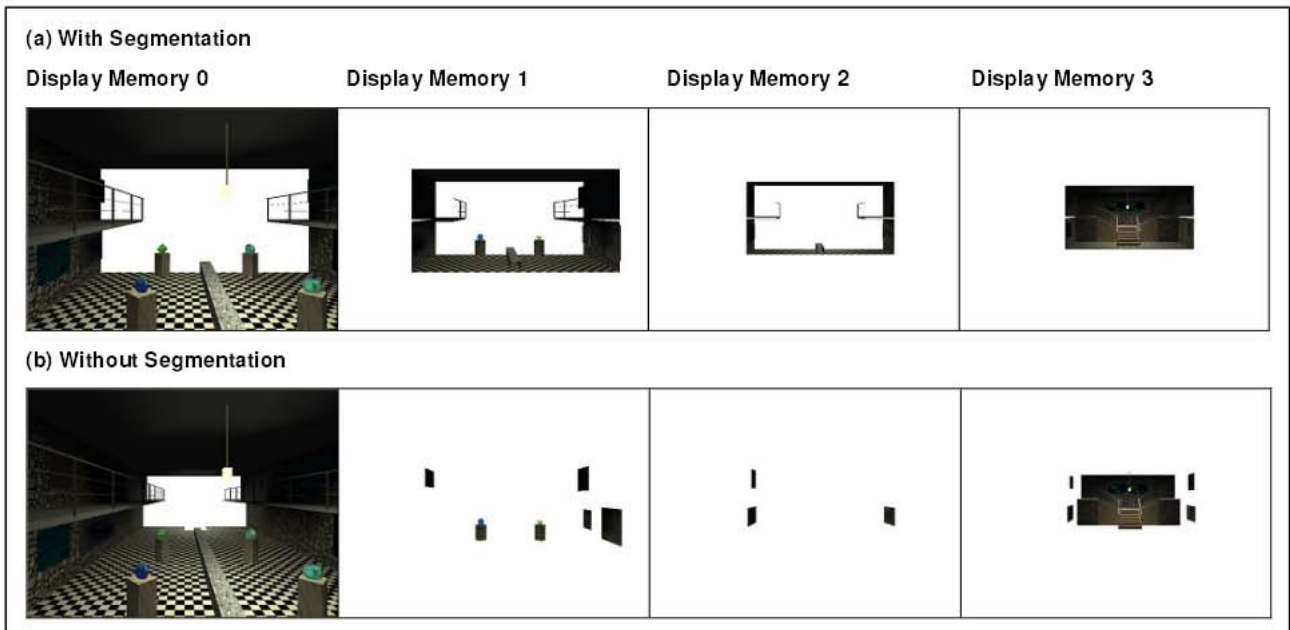
Figure 12: Images from the user's viewpoint, mapped from the different display memories.
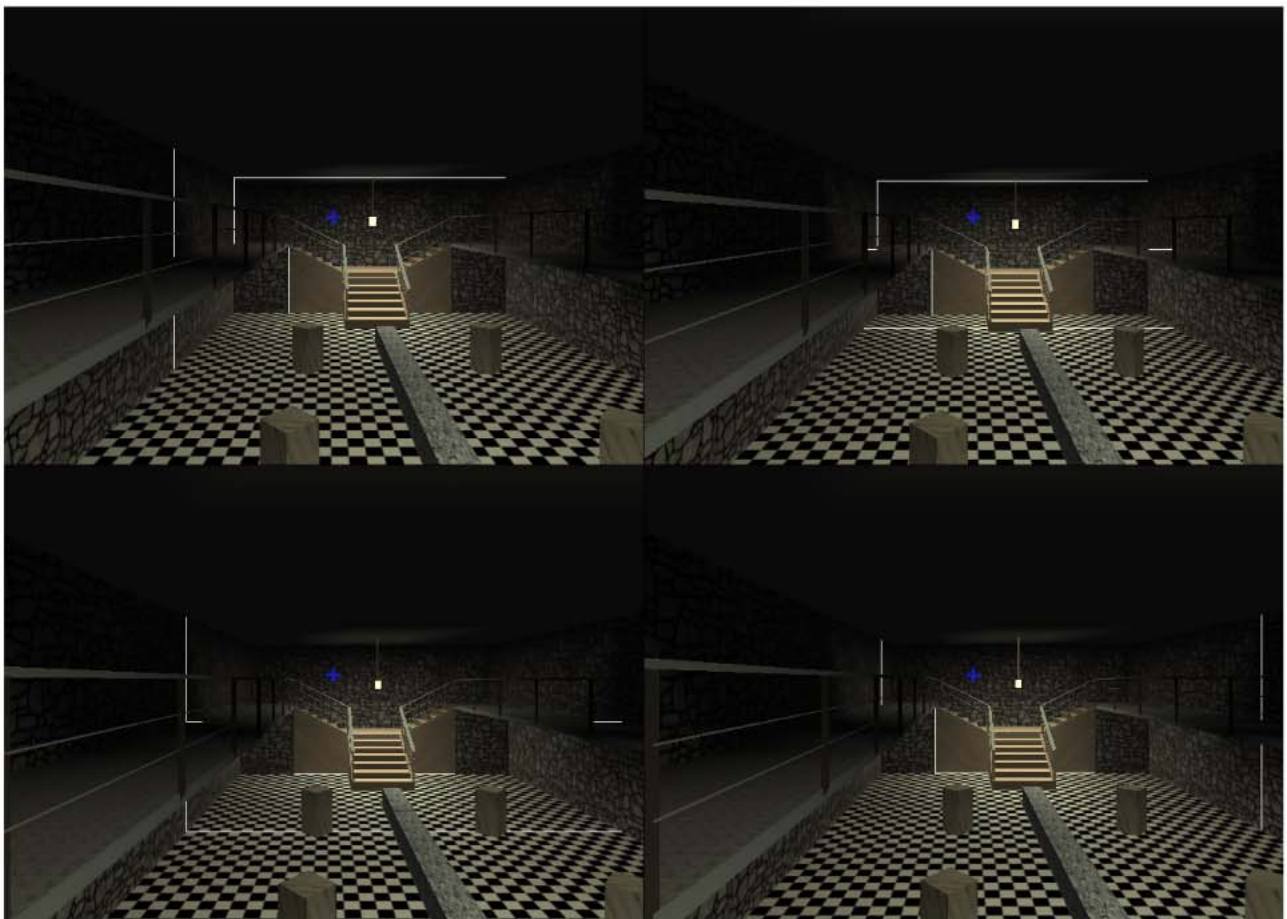

Figure 13: Tearing artefacts.

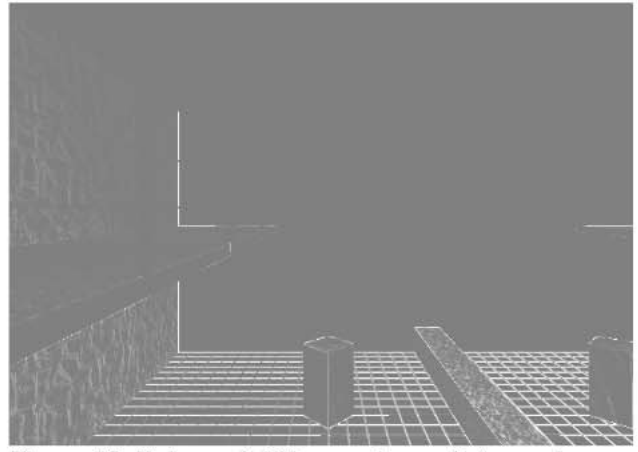**Figure 14: Composite image of the display memories shown in figure 12.**

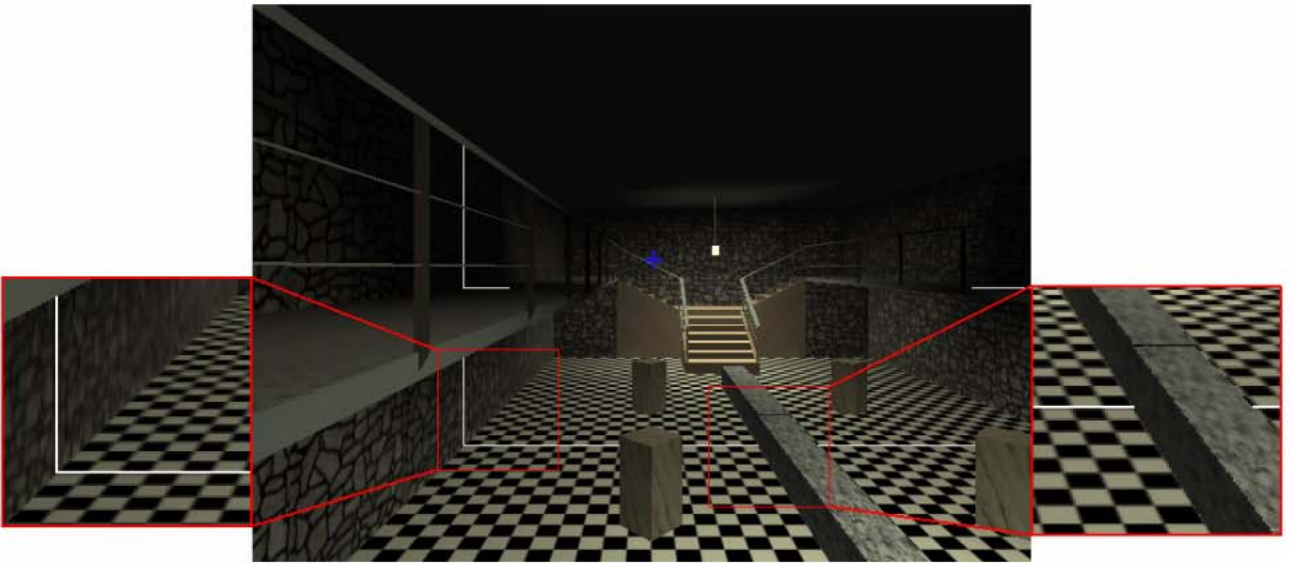**Figure 15: Enhanced difference image between frames shown in figures 16 and 17.**
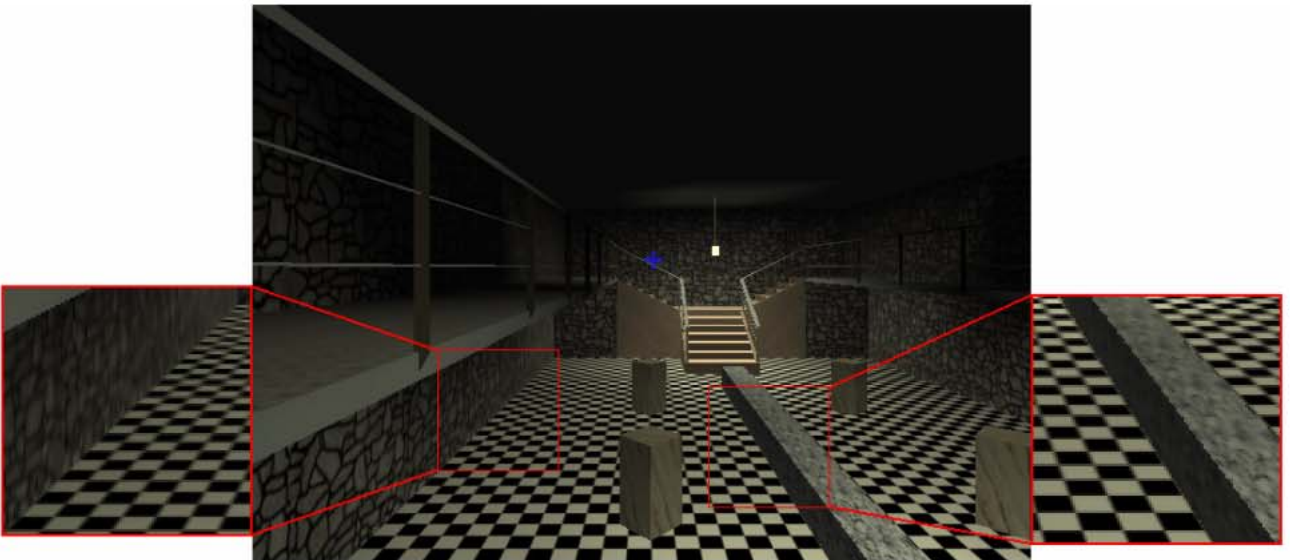

Figure 16: Magnification of tearing.

Figure 17: Magnification showing the region warping results.