

Parallelism in a Computer Architecture to Support Orientation Changes in Virtual Reality and Other Immersive Interactive Graphics Systems

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 form a category of multimedia applications which requires enormous amounts of computational power. Even the most powerful of current computers cannot generate photo-realistic images in real time at a rate to ensure smooth motion and with imperceptible lag between user interaction and seeing the results. A new conceptual view of the problem is presented which changes radically the nature of the computation required to get the desired effect. The new approach is still computationally expensive but is tractable using relatively simple and economical parallel hardware. An outline of the new parallel architectural approach is presented here.

1 Introduction

Researchers and practitioners have long been grappling with the serious problem of latency in responding visually to movement of a user in a virtual environment. The problem, while simply described as the lag between a user's head position or orientation changing and the updating of the displayed virtual view to reflect that change, has rather severe consequences, and has eluded solution by many practitioners. While it is well known how to do the computation required to generate computer graphics images appropriate to the orientation of the observer in the virtual world, the problem is computationally extremely expensive. This has meant that even the fastest of conventional graphics computers have not been able to achieve the desired performance of over 60 realistically rendered frames per second. Even if a heavily pipelined supercomputer system can generate images at the required rate, the problem of latency due to pipeline startup overheads remains.

It turns out that by solving this problem in the domain of virtual reality systems one has also provided a tool with much more general application in areas such as telerobotics and a novel form of interactive television.

In this paper I will first give an outline of the conventional approach to the problem. Then I will present a new way of thinking about the original problem that led to the revolutionary new architectural approach. The key to the success of this project has been 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* involves a great deal of parallelism in its implementation. A new technique of exploiting the new architecture has also been developed [Regan & Pose 1994] which can also have a parallel implementation involving both hardware and software, but that is not the subject of this paper.

Having set the scene by changing the way one thinks about the problem, I go on to describe the specialized parallel computer architecture which allows real time interactive response to changes in the user's viewing orientation

I conclude with a summary of the parallelism exploited in the *Address Recalculation Pipeline* and why it is more effective than a conventional highly parallel computer graphics engine.

2 Conventional Approaches to the Problem of Interactive Latency

One approach to building virtual reality systems has been to slow the update rate of the display so as to enable the computer system to cope with the processing load. This leads to a rather jerky displayed world with latency being at least the time between updates of the display. Experience with motion pictures and with television has shown that the higher the frame rate the better.

An alternate approach is to reduce the processing load by reducing the quality and resolution of the images displayed. While the display update rate is increased and latency somewhat reduced with this approach, the realism of the virtual world is sacrificed leaving one with cartoon-like imagery rather than anything approaching photo realism.

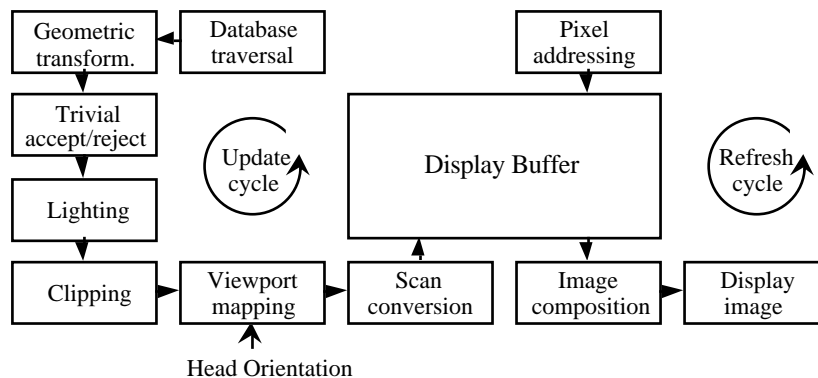


Figure 1. Conventional Virtual Reality Display Architecture.

The state-of-the-art in conventional virtual reality systems relies on using the fastest available graphics processing and rendering equipment as depicted in Figure 1. This tends to be very expensive with the very best systems costing over a million dollars. In effect the problem is tackled indirectly through the use of massive amounts of processing power. Even so it can be observed that the trade-off between latency and image quality still has to be made, and that with reasonably realistic looking scenes the latency is still a significant problem. An extremely successful virtual reality computing system is the *PixelPlanes* system developed at the University of North Carolina [Fuchs et al., 1989]. This uses thousands of processors to give extremely impressive rendering performance and very good latency characteristics, however *PixelPlanes* is rather expensive technology which most developers could not afford. Most current high performance immersive virtual reality or interactive graphics systems make use of one or more high performance graphics pipelines such as produced by Silicon Graphics (SGI). Nothing even approaching real time photo-realistic performance is possible using current conventional systems.

3 Re-Examining the Problem

In order to deal with this problem of latency it is helpful to forget for a moment the conventional ways of computer graphics and instead look at the overall environment in which we are working.

The problem as stated earlier is one of mismatch between what is displayed to the user and what should be displayed according to the virtual world model in which the user is situated. This mismatch usually is due to the latency involved in updating the display to reflect the virtual reality. The easy solution, involving reducing image quality so one can keep up with the virtual world, creates a mismatch of a different kind, but still spoils the illusion of virtual reality.

Let us examine what is happening in the virtual world itself. It could be that some part of the virtual environment is changing or moving in some way. While indeed this does occur, it is also true that in general only a small part of the virtual world is changing at any time, and typically the virtual world contains many static objects or scenes. Thus we can deal with such activity without recomputing everything, and since the objects involved are generally independent of the user, a slight delay in reflecting their changes does not cause the serious problems mentioned above.

The other event, which is much more significant, is the movement of the user within the virtual environment. Any slight movement of the user's head leads to a change in everything the user sees, hence it looks to the user as though the observed environment has moved completely. Thus one can see that it is the relative movement between the user and the objects in the virtual world which is observable to the user, and it is the user's own movement which has the most dramatic effect, since it affects the display of even static parts of the virtual world.

Now we should look more closely at the various types of user movement, to see what the consequences are. Consider the obvious movement, translation, for example walking forward. 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 much at all. Thus for translations we have to concentrate especially on close objects, and can largely ignore the distant background. In typical scenes the majority of the complexity of the scene is in the background, hence the problem appears tractable. We are also fortunate in that people do not tend to translate so very quickly so changes tend to happen at a manageable rate.

Another user movement is that of rotation of the head. Here we have a rather different effect. Even a small rotation immediately causes everything in the observed view to appear to move, even the background and static objects. There is also the added problem that rotations are much faster and more frequent than translations; one tends to look around and observe one's environment by rotating one's head or eyes. It looks as though everything has to be recomputed and re-rendered.

However, let us not forget the idea of stepping back and looking at the overall environment. It seems that the environment is staying relatively static but the user's view of the environment is changing quite rapidly and somewhat unpredictably, mainly due to rotation. The key point here is that the environment itself is not changing dramatically, so one needs to find a way in which the representation of the displayed environment also stays relatively static, and hence can be computed without the serious problems with which we are concerned.

Note that in a conventional computer graphics implementation user orientation changes, i.e. rotations, are guaranteed to require recomputation of the rendered scene. Given that in such systems there is inevitably some slight rotation involved even in a translation movement, there is no point in distinguishing between rotation and translation, and in fact any change in user viewpoint causes the scene to be regenerated at great cost.

4 A New Conceptual Model

The ancient Greeks found a similar problem in trying to describe the motions of the planets and stars. They had a concept of planets moving relative to stars and to the Earth, and stars moving relative to the Earth. A model involving *crystal spheres* was developed. In this model the heavenly bodies were *painted* onto various layers of crystal spheres which were centred on the Earth and could move relative to one another. Objects on close spheres move more compared with objects on distant spheres. Various complex models of the movements of heavenly bodies were formulated in terms of these spherical shells surrounding the Earth. We now know that much of the observed motion of the stars is due to the rotation of the Earth, but since the observed motion is relative, a successful model centred on the Earth is possible.

We can use a similar model for a virtual world. Concentric spheres centred on the user would have the objects of the virtual world painted on them. Close objects on inner spheres and distant objects on outer spheres. In essence all possible views from the user's position are already rendered onto the spheres, and it only remains to display appropriate views as seen by the user. A rotation merely involves displaying a different portion of the sphere. A translation will require some updating of the spheres, however outer spheres will change much less than inner spheres, and hence may require little or no updating. Of course changes within the virtual environment will have to be reflected in the spheres, but typically only a subset of the spheres will be involved.

5 Implementation of Viewport Independent Display Memory

We have devised a model in which we render the images of the virtual world onto a surface or surfaces surrounding the user. However it may appear that in so doing we have actually created a larger processing bottleneck, both in generating the images and rendering them, and in selecting the appropriate view to display. On first examination it may appear we have significantly greater rendering overheads such as scan conversion, clipping etc. than 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. With a viewport independent display memory one must scan convert all polygons received. This is the worst case scenario for a conventional system, but for guaranteed interactive response one must allow for the worst case. Many conventional rendering systems are designed to cope with situations approaching the worst case scenario [Molnar & Fuchs, 1990]. 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. The viewport mapping is indeed computationally expensive however this task has been offloaded into relatively simple and cheap dedicated hardware, the *Address Recalculation Pipeline* [Regan & Pose, 1993].

The latency is determined by how long it takes to select the required portion of the encapsulating surface and write the image to the display device. This is handled by the *Address Recalculation Pipeline* which runs at the speed of the display device and introduces negligible latency. The new improved approach is depicted in Figure 2 and a comparison of the latency components with those of conventional systems is shown in Figure 3.

The selection of the appropriate region of the sphere to display can be modelled as projecting rays from the eye onto the surface of the sphere, so defining the area to be seen. Alternatively one can view the process as one of changing coordinate systems from real-world to spherical world. This indeed requires a great deal of

computation for each pixel displayed and would be infeasible in software without a vector supercomputer, which in itself would introduce pipeline delays. However the approach is ideal for a pipelined hardware implementation, which while using fairly fast hardware has a fairly simple structure and can be built economically [Regan & Pose, 1993].

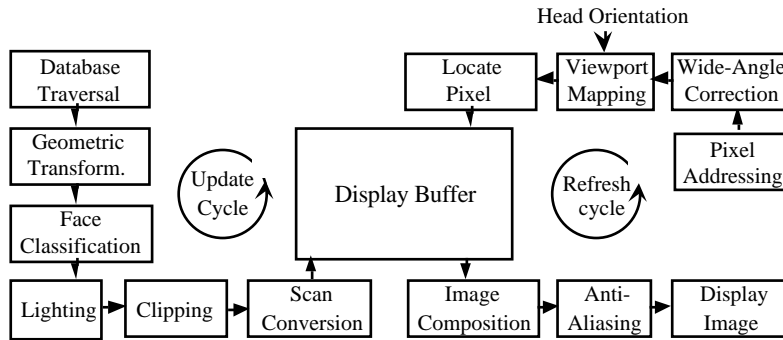


Figure 2. Delayed Viewport Mapping.

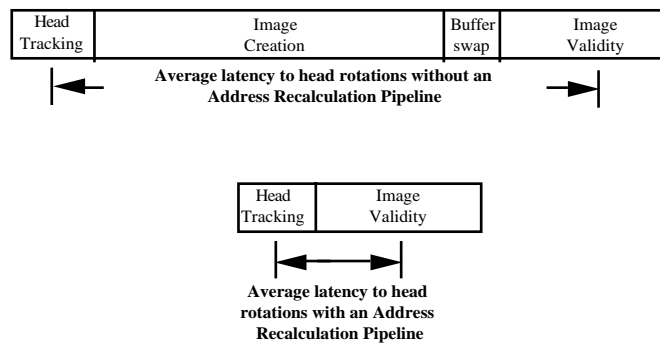


Figure 3. Comparison of rotational latency components.

We did investigate a hardware design based on a spherical system but found some annoying characteristics. First, in mapping a sphere onto a memory array one tends to waste a lot of memory, or else have a very complex mapping scheme. Second, the apparent sizes of pixels in a sphere vary greatly as one moves from equator to the poles. Third, the coordinate transformations involve trigonometric functions which are moderately expensive to implement. By far the dominant issue is that rendering onto the surface of a sphere is computationally more expensive than rendering onto a plane. We did not want to do anything to slow graphics performance unnecessarily.

Essentially the basic model will work for any surface surrounding the user. A sphere is intuitively obvious in that everything is equidistant, however one can go to the other extreme and look at a tetrahedron. What we eventually chose was a cube. It

has very nice properties in that no memory is wasted since its surfaces have the same topology as memory chips. The coordinate transformations are also much simpler in that they are linear and can be implemented essentially with a matrix multiplication. Pixel sizes do not vary as much as for a sphere. While there may be some concern about strange effects occurring in corners and edges these have been shown not to be significant.

A prototype display memory system based on the cube has been implemented and functions well. We call this method using viewport independent rendering, *Delayed Viewport Mapping*, and the hardware realization, an *Address Recalculation Pipeline* [Regan & Pose, 1993].

6 Image Composition

A technique which has been shown to work well in accelerating computer graphics systems is image composition [Molnar, 1991]. Instead of treating the scene as a single entity, one can break it down into component objects and combine them at the end to produce the scene. One can render these component objects separately, and if one has multiple renderers available one can get parallelism in the rendering process. The big gain is that objects that appear not to change can be left, and not rendered again if you want to change the scene. Thus one can achieve better than linear speed-up. Unfortunately this does not help much in virtual reality applications since, as we have seen, a simple rotation will cause every object in a scene to change.

We can get the benefits of image composition by employing our viewport independent approach in virtual reality applications. Since the surface encapsulating the user does not change when the user rotates, the objects already rendered can remain, and once again we get better than linear speed-up with multiple renderers. Conceptually we have gone back to the *crystal sphere* model, and our image composition involves combining images of objects painted on various concentric cubes. We use the Z-values of pixels to enable us to choose the closest pixel to the user as the one to be displayed. Hence we do not force objects onto particular cubes in relation to their apparent depth, but instead allow the hardware to resolve dynamically which pixels to display. We also provide separate rendering engines for each cube. Thus it makes sense to distribute objects over cubes in such a way that those that need to be updated at similar times are grouped together. In this way the rendering capacity is not wasted on redundantly re-rendering objects which have not changed. A technique called *Priority Rendering* has been developed to handle display updates efficiently [Regan & Pose, 1994]. Parallelism is also involved here but will not be discussed in this paper.

Image composition occurs as pixels are fetched from the display memories to be displayed on the output device. All of the pixels from the display memories which correspond to the same screen location are compared to find the closest pixel, which is then displayed.

7 The Address Recalculation Pipeline

The system as described thus far takes the form of a pipeline, which we term the *Address Recalculation Pipeline*. In it a rotation matrix derived from tracking or other devices is used together with data describing the optical characteristics of the display system, and of course the actual image data, to re-orient the images viewed by the user, in real time. This involves a matrix multiplication as each pixel is displayed on the screen, that is millions of matrix multiplications per second. This is quite manageable, but one must consider the enormous quantities of data flowing through the system. Given that the overall structure is that of a pipeline, much of the data traffic only flows from one stage to the next, so is not too difficult to handle, however there is an aspect which did require an innovative architectural solution.

Because the images on cubes inside the system do not correspond directly to any images seen by the user, it is impossible to do the usual anti-aliasing operations which help minimize the *jaggies* and other effects caused by the discrete sampling nature of a raster graphics system. Thus we cannot rely on the image rendering software to do the anti-aliasing. Instead we need to run a smoothing filter as the images are being displayed, in real-time. Even a fairly simple anti-aliasing filter, such as bilinear interpolation filtering, requires four adjacent pixels to be averaged to create a pixel value to be displayed. In our system we do not even know which pixel is going to be displayed until just before it appears on the screen, hence we needed to find a way to simultaneously get the four adjacent pixels from our display memories.

An ingenious solution was found which involves dividing the display memory into six independent segments [Regan & Pose 1994a], and allocating pixels to the segments such that it is guaranteed that the four pixels required for the bilinear interpolation are each in different segments. It then remains to choose which four of six pixel values are required. Each segment is of course assigned to different memory banks so that they may all be accessed in parallel.

8 Parallelism in the Address Recalculation Pipeline

The first stage of the pipeline uses a lookup table with one entry per pixel on the display device. The output of this lookup table is a vector which notionally extends from the displayed pixel through the surface (cube) of the display memories. A lookup table is used to reflect the potential distortions of the optical system which ensures a wide viewing angle. The viewer's current orientation is captured by a head tracking device which produces a rotation matrix. The second stage of the pipeline multiplies the vector by this rotation matrix to get the vector oriented in the right direction. This involves 9 multipliers in a 3x3 array with adders between them. The next stage calculates the intersection with the display surface which gives us the pixel we want. This calculation involves some trial divisions which are implemented using reciprocal lookup tables and 6 multipliers. At this stage we now have the addresses of

the required pixels in the display memories. We then fetch the required four adjacent pixels in parallel from the display memories which are organized as further pipeline stages. Each display memory board takes in the pixels fetched from the previous board and does a depth comparison, forwarding to the next stage the pixels closest to the user. After all the display memories the next stage in the pipeline is the smoothing filter which does a bilinear interpolation of the four adjacent pixels. Finally the pixel value is sent to the display device.

Summarizing the parallelism within the pipeline stages. Stage 2 has 9 multipliers and a number of adders running in parallel, and the multipliers themselves are pipelined Wallace-trees. Stage 3 has 6 multipliers running in parallel. Each of the display memories which follow in the pipeline is arranged with 6 memory banks being accessed in parallel and some depth comparison circuitry. The smoothing stage which follows runs a pipelined set of adders and lookup tables for the bilinear interpolation. Of course for a stereoscopic system the whole pipeline is replicated and runs in parallel. Figure 4 depicts the pipelined system.

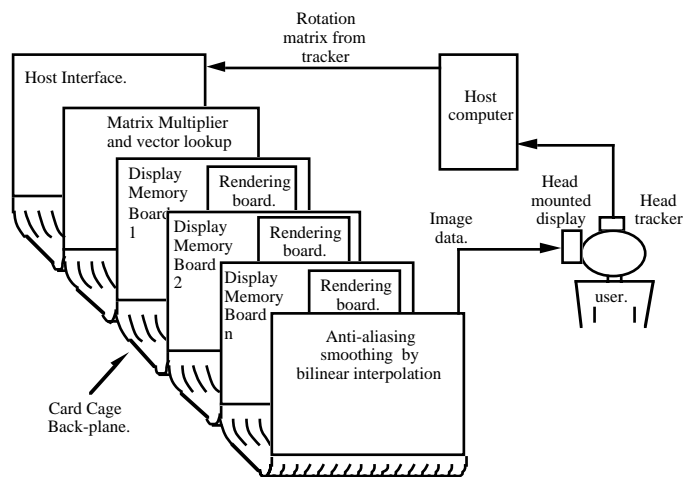


Figure 4. Address Recalculation Pipeline System.

9 System Performance

The virtual reality system described here involves two distinct processes. This paper concentrated on the *Address Recalculation Pipeline* which in real-time allows a change in user orientation to cause the appropriate images to be displayed. This, as we have seen, is a highly pipelined hardware implementation with several multiplications, additions and memory look-ups occurring in parallel within each pipeline stage. Thus a large amount of parallel computation is going on and none of this is related to computer graphics, but rather to selecting, smoothing and removing distortions from pre-computed graphics images. This enables user orientation

changes within a static virtual environment to be handled with imperceptible latency. Thus it is providing essentially infinite performance improvement over a conventional system, even one using highly parallel computer graphics engines such as Pixelfusion. This is because no computer graphics at all is taking place although much other computation is occurring.

Of course a real virtual reality system has to allow users to move within the virtual world, not just change their head orientation, and of course objects within the virtual world must be allowed to move, change and interact with one another and with the user.

Our *Address Recalculation Pipeline* has decoupled the rotational orientation changes from these translational movements which indeed do require computer graphics calculations. This decoupling has however allowed a successful employment of image composition whereby we have spread the images of the objects in our virtual world over multiple display memories, each equipped with its own graphics rendering engine, which itself may be highly parallel. Without rotational movement now being a concern, we can see that very distant objects in the background have images which will not appear to change unless a very large user translational movement occurs. Relative movement between the user and the objects in the virtual world is what causes the appearance of the objects to change, and that is greatest for close objects and those with large relative speeds with respect to the user. Thus not all objects need their images recomputed at the display refresh rate since it is pointless recomputing something which will give the same values again.

Our experiments show that about an order of magnitude reduction in computer graphics rendering compared with a conventional purely computer graphics approach can still provide the user with identical images. The method we have developed for exploiting the *Address Recalculation Pipeline* architecture in this way is called *Priority Rendering* [Pose & Regan, 1998], and an analysis of its parallelism has also been undertaken [Pose 2000].

In summary with an order of magnitude reduction in required graphics computation for translational movement, and a complete elimination of graphics computation for rotational user movement, we have an overall performance improvement of several orders of magnitude in comparison with conventional computer graphics systems doing virtual reality. Even highly parallel computer graphics rendering engines cannot achieve this performance, but they could be used as graphics renderers in our system.

Conclusion

By changing dramatically the way one views the virtual world, one can take advantage of techniques such as *Delayed Viewport Mapping*, image composition and *Priority Rendering* to increase the performance of virtual reality computer systems. A novel but affordable parallel hardware architecture, the *Address Recalculation*

Pipeline, was employed to implement an extremely low latency virtual reality display system.

In this paper an outline of the new display architecture was presented. This set the scene for a discussion of the parallelism involved in the implementation of the system. Certainly a great deal of computation is done, of the order of a hundred million matrix multiplications per second and other calculations as well as some conventional computer graphics computations, to generate the images for the display memories. However a specialized parallel computer architecture can do this much more economically and efficiently than a vector supercomputer.

It is also interesting to note that the computation performed in this architecture is on pixel addresses rather than on the values of the pixels themselves. Conventional computer graphics systems compute the pixel values. The parallelism involved in the *Address Recalculation Pipeline* takes the form of a high speed pipeline with individual pipeline stages themselves having a high degree of parallelism.

Our system essentially solves the problem of latency to user orientation changes in immersive computer graphics systems such as employed for virtual reality. It achieves this by decoupling the updating of the displayed images due to user orientation changes from the generation of the images themselves.

Acknowledgements

Matthew Regan was responsible for much of the design and implementation of this system.

The prototype hardware was funded by an Australian Research Council small research grant. Patenting and commercialization of this technology was supported by a Monash Research Fund grant

References

- [Fuchs et al., 1989] Fuchs, Henry et al. (1989) Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor Enhanced Memories. Proceedings of SIGGRAPH 89. In *Computer Graphics*, Annual Conference Series 1989. pp. 79-88.
- [Molnar & Fuchs, 1990] Molnar, Steven and Fuchs, Henry. (1990) Advanced Raster Graphics Architectures. Chapter 18, *Computer Graphics*, Foley and van Dam, pp. 872-873.
- [Molnar, 1991] Molnar, Steven, (1991) Image Composition Architectures for Real-Time image Generation. Ph.D. dissertation, University of North Carolina, 1991.
- [Regan & Pose, 1993] 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.
- [Regan & Pose, 1994] 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.
- [Regan & Pose, 1994a] 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.

[Pose & Regan, 1998] Pose, Ronald and Regan, Matthew (1998) *Updating Graphical Objects Based on Object Validity*, US Patent Number 5,841,439, Granted 24 November 1998.

[Pose, 2000] Pose, Ronald (2000) *Effectively Exploiting a Parallel Algorithm for Virtual Reality Scene Generation*, Proceedings of the 4th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP'2000, Hong Kong, 11-13 December 2000.