Application of Parallel Processing to Rendering in a Virtual Reality System

Shaun Bangay
Peter Clayton
David Sewry

Department of Computer Science
Rhodes University
Grahamstown, 6140
South Africa
Internet: cssb@cs.ru.ac.za

Abstract

This paper describes a number of parallel rendering strategies that were tested in order to determine their suitability for use in virtual reality systems. Each strategy was implemented on a cluster of transputers and evaluated with respect to rendering speed and interaction latency. Strategies were developed that are better suited to parallel virtual reality systems than the standard graphics pipeline.

1. Introduction

The rapid rendering of images is pivotal in all applications of computer graphics, and perhaps most of all in the field of virtual reality. Research suggests that there is a limit to the improvement in performance that can be attained by the selection of the appropriate algorithms and their careful optimisation. Hardware implementation increases expense and limits the flexibility of the equipment. A relatively unexplored option is the use of parallel processing in image rendering, especially image rendering in virtual reality systems.

This paper examines the application of parallel processing to image rendering in a virtual reality system. The requirements of a virtual reality system are noted. A number of different

parallelisation techniques are proposed, and each analyzed with regard to its effectiveness in meeting these requirements. Performance values, measured from an implementation of each technique on a transputer architecture, are used to highlight the advantages and disadvantages of each technique.

2. Rendering for a virtual reality system

Rendering is a well understood application. Many systems have been created in which three-dimensional scenes have to be rendered. The use of parallelism for rendering has also been explored. However, previous research has not placed the emphasis on rendering scenes in real-time. In particular, little work has been done on using parallel processing when rendering for virtual reality applications.

Virtual reality applications are different in that the images must be rendered very rapidly, in real time, and that the delay between a user input and the corresponding change in the rendered output, the latency, must be small. In working virtual reality systems [4] [1], frame rates of 6 frames/second and a latency of less than 200 milliseconds are suggested as minimum requirements for the illusion of reality. These two factors, the speed of rendering, and the latency period are used as the criteria for evaluating the parallelisation techniques explored in this paper.

3. Rendering on a single processor

Renderers are typically constructed by creating a pipeline of several standard operations ([3], [2]). These operations include clipping, transformation, projection and hidden-surface removal. The description of the scene to be displayed is fed into the pipeline which eventually produces a sequence of primitives (two dimensional lines and polygons, for example) to be drawn. The pipeline typically ends with some routines capable of producing a visual representation of these primitives.

The renderer described in this paper is modelled as three stages. The first selects those objects in the virtual world which will be visible on the screen. The next stage transforms the objects to screen coordinates. The last stage performs hidden-surface removal, clips the objects to the dimensions of the screen and renders each primitive. This level of breakdown is adequate for the parallelisation used which concentrates more on parallelisation inside each stage, rather than between stages.

4. Parallelisation of the graphics pipeline

A number of different parallel decomposition strategies for polygon rendering are described in a paper by Whitman [5]. Four decomposition strategies were implemented, and are described in the following sections.

The various strategies were implemented on a transputer-based architecture. However, in most cases, the techniques are applicable to other architectures.

The hardware platform on which the parallelisation was carried out was a MIMD architecture consisting of a cluster of 32 T800 transputers. Each transputer is a reasonably powerful processor in its own right, capable of running a number of concurrent processes. The addition of 4 communications links allows it to transfer data to and from others transputers. Each transputer has its own private memory and all synchronisation and data transfer occurs through the links. A single bidirectional link can transfer a theoretical maximum of 2.35 Mbytes/s, or 1.74 Mbytes/s if communication is only in one direction.

A set of standard test data was used to obtain timing data from the various implementations. The speed of the rendering system on a single processor is given in Table 1. The simple test world contains 30 objects amounting to about 250 polygons (each with 4 or more sides). The complex world consists of a single object containing about 1650 triangular polygons.

4.1. Pipeline parallelism

The most obvious strategy when confronted with the graphics pipeline is to place each component of the pipeline on a separate processor. The connections between components can be constructed using transputer links. Two factors are worthy of consideration when setting up a pipeline configuration for virtual reality on a transputer architecture.

Firstly, greater bandwidth is obtained from the transputer links when larger messages are transmitted. Protocol overhead is less, and there is less delay waiting for each of the communicating processors to synchronise. The few bytes representing a point are not worth transmitting alone, it is better to wait until a number of points are ready to be transmitted and

Table 1 Speed of the rendering system in frames/s

Frames rates:	Simple world	Complex world
512 x 512 resolution	4.17	0.87
320 x 200 resolution	5.67	0.88

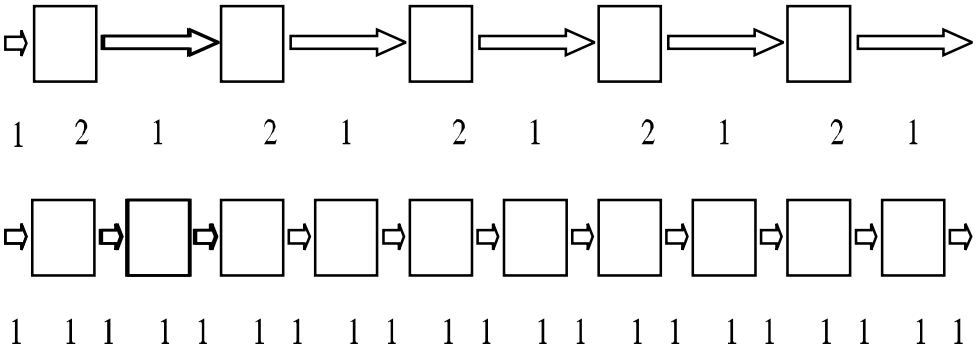


Figure 1 Illustration of latency in pipelines

then send them all together. For convenience, all the data corresponding to an entire object is sent in one packet.

The second consideration is the latency problem, the delay between input and the corresponding output. Lots of data is held in long pipelines, even with a limited amount of data at each node. All of this must be processed before new data entering the node can have any effect on the image being displayed. Since there is a certain overhead associated with the communication between pipeline components, the delay between data entering the pipeline and it reaching the output increases with pipeline length, even though the rate at which images are produced is increased.

This may be illustrated by example: Consider a job which on a single processor would accept a datum and take 10 seconds to process it before outputting the result. Assume this job is capable of being parallelised linearly. A pipeline in which each node shares equally in the workload is illustrated in Figure 1. The data is arriving as fast as possible, but will take 1 second to transmit across a link. With 5 processors the job can be divided into a pipeline of 5 parallel components, and so each will work on the datum for 2 seconds. Thus results can be output every 2 seconds. The time between a given datum being input and the corresponding output is at least 16 seconds. With 10 processors, output occurs once every second but at least 21 seconds pass between input and corresponding output.

In general:

let N be the number of processors,

let T be the time for the job on the single processor system and

Table 2 Frame rates for pipeline parallelism

Frames rates:	Simple world	Complex world
512 x 512 resolution	6.99	1.09
320 x 200 resolution	10.00	1.11

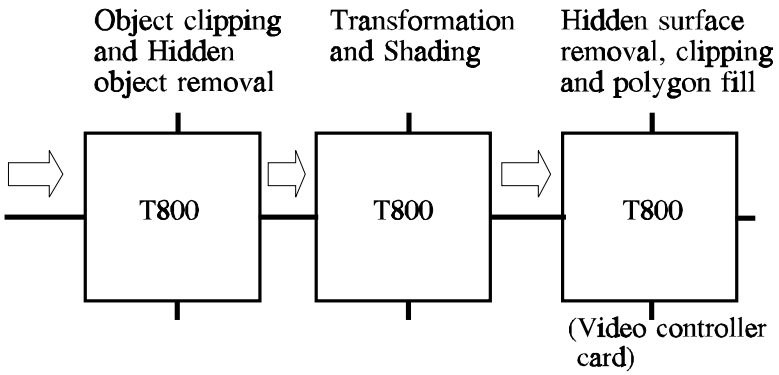


Figure 2 Processor layout for pipeline parallelism

let C be the communication time between processors.

The data output rate is then T/N.

The latency is at least (N+1)C + T seconds.

The latency increases with number of processors for non zero communication time. For zero communication time it remains constant, independent of the data output rate. For this reason pipeline parallelism is better suited to systems that require computationally intensive rendering with limited user interaction.

A limited rendering pipeline with three nodes was implemented. The processor layout is shown in Figure 2. Timing figures are given in Table 2. The differences in speedup at different screen resolutions are due to change in the load balance. The renderer has to do more work at higher resolutions and so acts as a bottleneck.

4.2. Coarse grained parallelism

The coarse grained parallel decomposition strategy renders successive frames on separate processors. The transputer network is arranged to allow a number of independent pipelines to be created, as shown in Figure 3. Each pipeline renders one frame in memory which is then transferred directly to the graphics node. This technique promises the greatest speedup since it requires no load balancing. Each processor can start work on another frame as soon as the current one is finished. The load is also approximately equal from one frame to the next, so they are likely to stay in step. There is thus no waiting for other processors to catch up.

The delay between input and output is still the time taken for the pipeline to render the frame. No decrease in this time occurs even when the number of parallel pipelines is increased. The latency is thus constant and independent of the number of processors used.

Values for the speedup obtained from an implementation of this technique are shown in Figure 4. As may be seen, linear speedup is obtained for most of these examples. The only factor preventing linear speedup in all cases is the bandwidth restriction on the links to the graphics node. With each link capable of 1.7 Mbytes /second and four links, this allows a maximum of

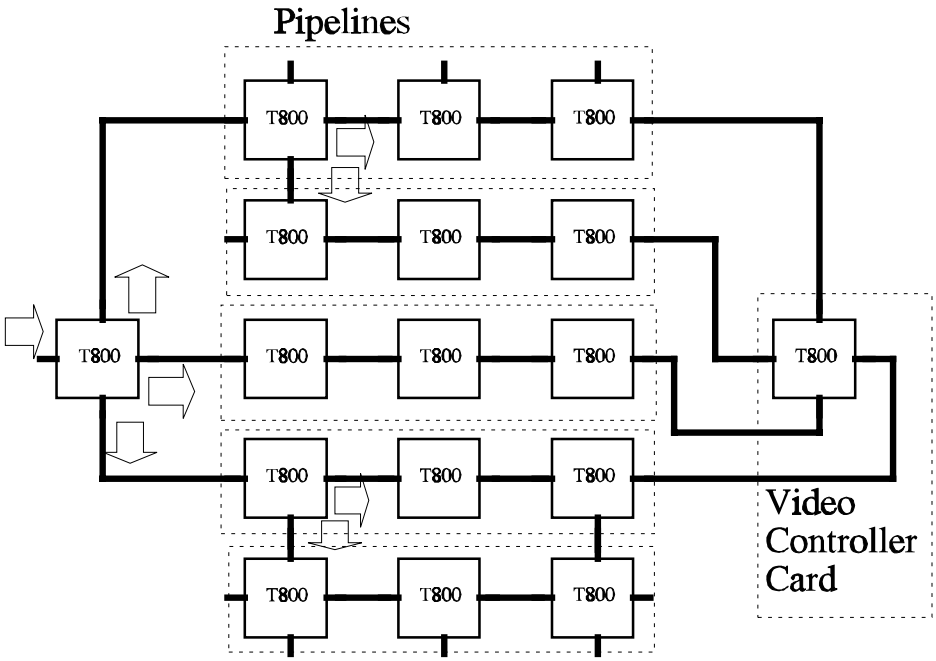


Figure 3 Processor layout for coarse grained parallelism

26.6 frames/second at 512 x 512 resolution or 108 frames/second at 320 x 200 resolution. The example with the simple world at 512 x 512 is using about half of the total bandwidth available.

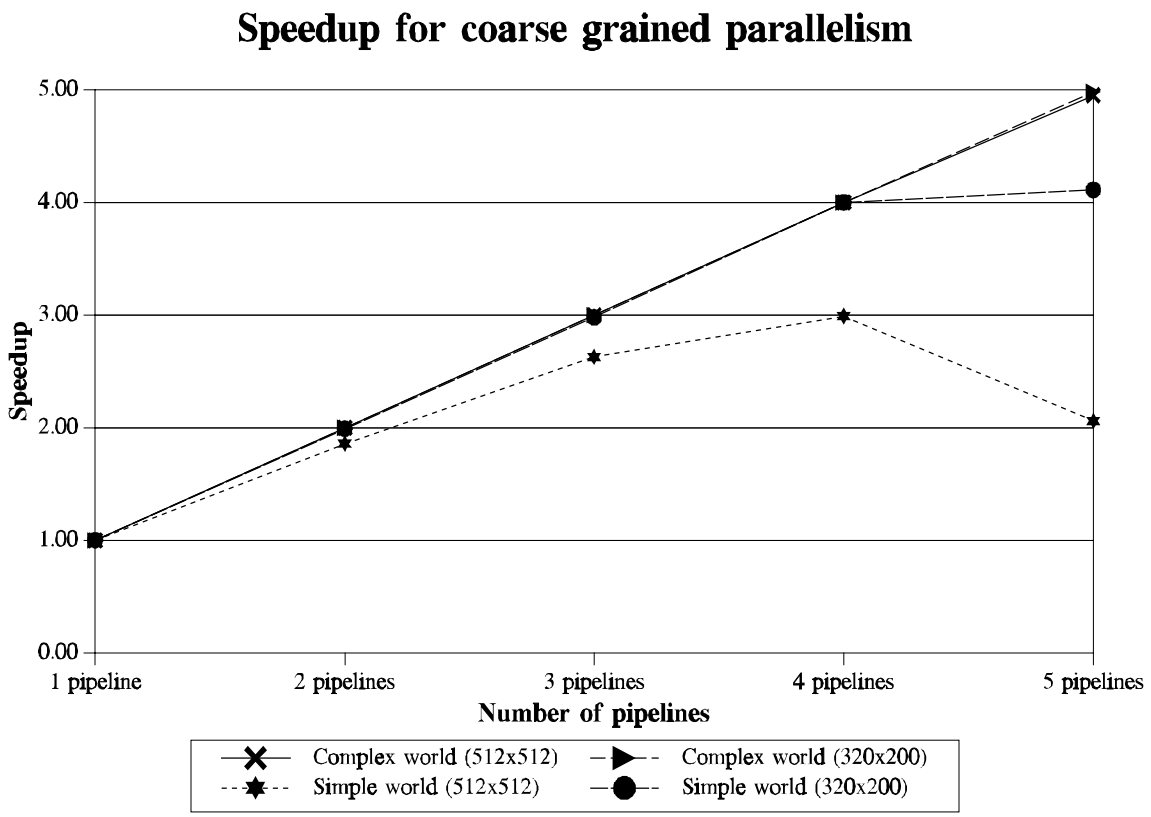
There is a general tendency for a drop in the speedup when more than four pipelines are used, especially in high traffic situations. This tends to occur with all the parallelism strategies tried. It can be ascribed to the necessary doubling of the traffic on one of the links to the video controller card.

4.3. Fine grained parallelism

The pipeline approach uses a number of transputers connected in series. It has the disadvantage of high latency. A more suitable parallel decomposition for interactive graphics applications would have the processors connected in parallel, and all working on the same frame. In this way the delay between input and corresponding output would be shortened.

A profile of the graphics routines running on a single processor reveals that the bulk of the effort is spent on the stage involving hidden surface removal and polygon rendering. Thus the bulk of the effort in parallelisation has gone into decreasing the time spent on this stage. The other stages of the graphics pipeline are left as a short pipeline on separate processors.

The fine grained parallelisation technique is an image space pixel based parallel decomposition strategy. The display area is partitioned into horizontal strips, and the hidden surface removal and rendering for each strip occurs on a separate processor. The strips are rendered in memory and transferred to the graphics node which fits each block into the correct area of screen memory. The amount of data that is transmitted to the video controller node is the



	Frames/s for 1 pipeline	Speedup for 2 pipelines	Speedup for 3 pipelines	Speedup for 4 pipelines	Speedup for 5 pipelines
Complex world (512x512)	1.09	2.00	3.00	4.00	4.95
Complex world (320x200)	1.12	2.00	3.00	4.00	4.98
Simple world (512x512)	3.98	1.86	2.63	2.99	2.06
Simple world (320x200)	8.06	1.99	2.98	4.00	4.11

Figure 4 Speedup values for coarse grained parallelism

same as with the coarse grained parallelism. The messages are shorter but more frequent. The processor layout used to implement this technique is shown in Figure 5. The shaded processors represent one possible graphics pipeline for rendering a single strip. The transformation stage can be parallelised in the same manner, with transformations restricted to those objects appearing on any particular strip. However, in practice most objects appear on every strip anyway and so this does not result in any significant saving. The transformation stage will be placed on a single processor to simplify the model.

Consider a simplified version of the graphics pipeline that consists of K stages with each stage taking time T to complete their task. By using N processors on one of the stages and assuming linear speedup, the time for that stage will be reduced to T/N. The total time taken to traverse the pipeline is (K - 1)T + T/N. The latency decreases as the number of processors is increased. The restrictions given above can be relaxed, and as long as speedup is obtained, latency will decrease.

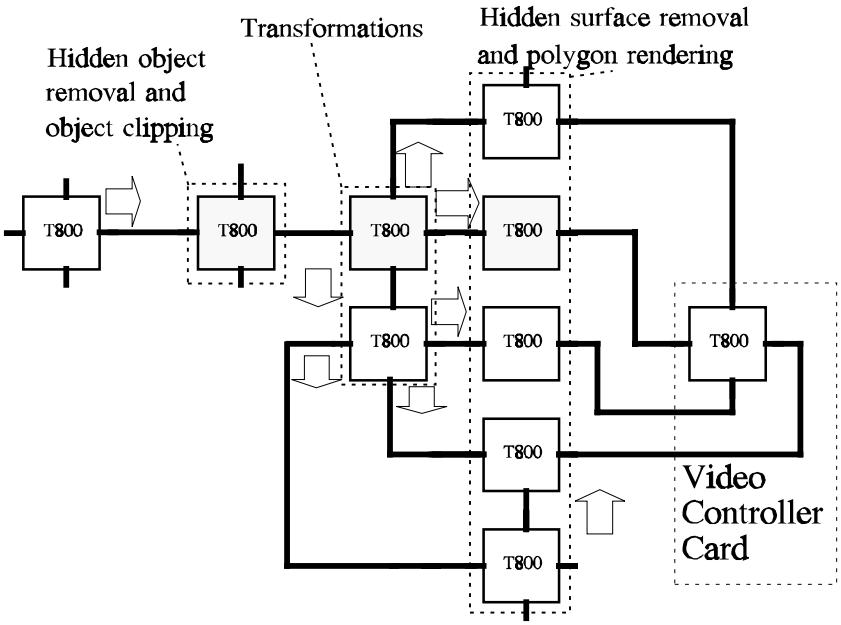
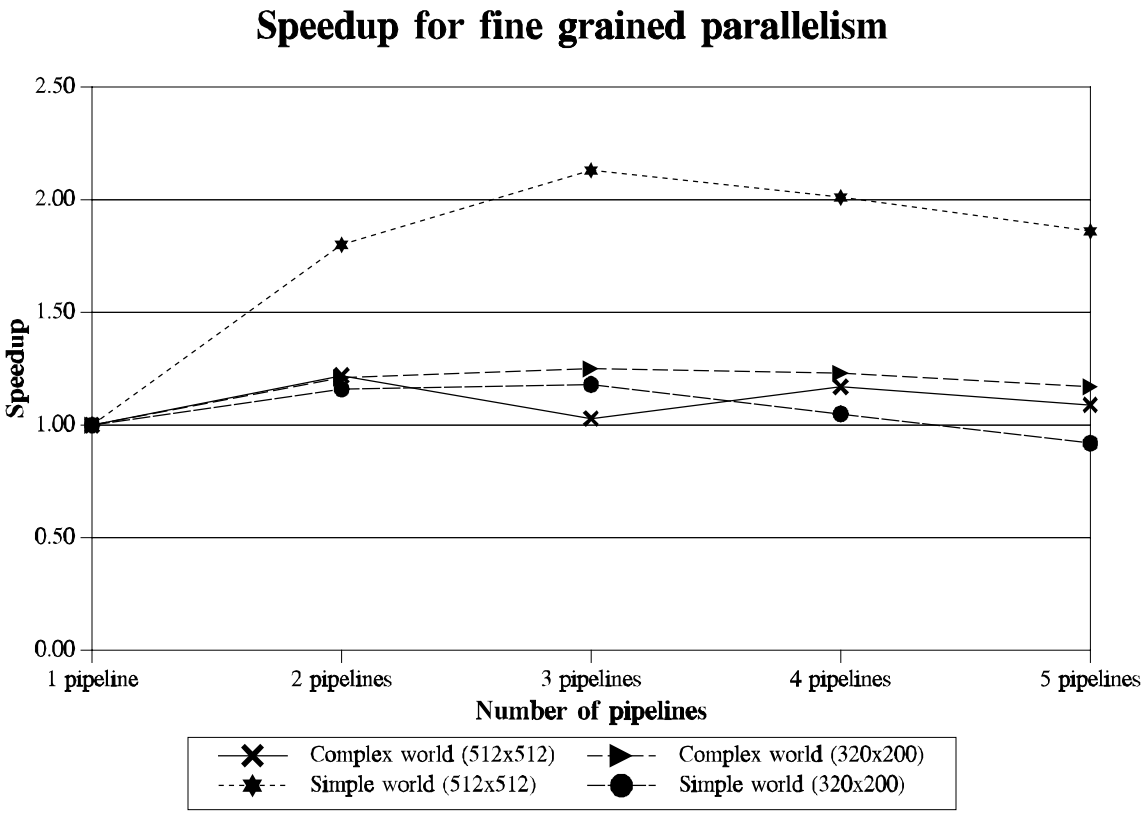


Figure 5 Processor layout for fine grained parallelism

The speedup values obtained from implementations of this strategy are shown in Figure 6. The times measured are for the entire graphics pipeline, and not just the hidden surface removal and polygon rendering. This latter part of the pipeline does most of the work however, and controls the overall speed. This technique does not produce a large speedup, nor does the speedup increase very much with the number of processors used. Two factors are found to contribute to these effects.

When rendering the horizontal strips, the polygons that fall partially outside these strips must be clipped. As the number of strips increases, each strip becomes smaller and the number of polygons needing to be clipped increases. In one test with the screen divided into 4 strips the time taken for clipping increased from about 25% of the time taken for rendering to about 500% of the rendering time. The rendering time did decrease by a factor of 4 due to the parallelism, but the clipping overhead was still significant. The clipping is a significant factor with the single object in the complex world with its many polygons and limits the speedup the most for this case.

The second factor involves load balancing. The picture is not always evenly distributed over the screen. Sometimes all the objects cluster in one region of the screen. In this case one processor is required to render most of the objects while others render almost nothing. This situation is almost equivalent to a single pipeline. The effect of this can be seen in the speedup values for the complex world with 512 x 512 resolution. The single object occupies the middle half of the screen. Thus for three pipelines, the processors rendering the top and bottom slices have little work to do and the processor rendering the middle slice does most of the rendering. This explains the drop in speedup over the two pipeline version where the load is more evenly balanced. The object fills the screen in the 320 x 200 resolution and clipping limits the speedup



	Frames/s for 1 pipeline	Speedup for 2 pipelines	Speedup for 3 pipelines	Speedup for 4 pipelines	Speedup for 5 pipelines
Complex world (512x512)	1.06	1.22	1.03	1.17	1.09
Complex world (320x200)	1.08	1.21	1.25	1.23	1.17
Simple world (512x512)	3.64	1.80	2.13	2.01	1.86
Simple world (320x200)	8.62	1.16	1.18	1.05	0.92

Figure 6 Speedup values for fine grained parallelism

in this case.

Alternative techniques for distributing sections of the display area over N processors to improve the load balancing include:

- Placing successive scan-lines on different processors.
- Dividing the screen into N^2 horizontal slices and placing successive slices on different processors.

These alternatives, however, incur greater overheads. The second alternative would result in more clipping, making it more expensive. The first alternative proved possible to implement with surprisingly few alterations to the rendering routines. Tests of the method show a more consistent tendency to increase in speed as the number of processors is increased, as opposed to the more erratic behaviour evidenced by the original routine. This can be attributed to the improved load balancing. Despite the improvement in scaling, the added complexity results in the improved algorithm running about 10% slower than the original.

The frame rates obtained with these methods were not high enough for bandwidth limitations to be a constraint.

4.4. Object-based parallelism

An object space parallelisation strategy was designed in which rendering of different objects occurs on different processors. For the case of N processors, the objects are sorted into N groups based on their depth. This is easily done as part of the depth sort involved in the hidden object removal routine. Each group is then rendered on one of the processors. The images produced are sent to the graphics node where the images are combined. The combination technique is similar to the Painter's algorithm, used for hidden surface removal, where images corresponding to nearer objects are painted over those belonging to more distant ones. The transputer has a specialised block move instruction that allows this to be done relatively quickly and easily. Both the transformation and hidden surface/polygon rendering stages can be done in parallel with this method.

The transputer network used to implement this strategy is shown in Figure 7. The shaded nodes show the graphics pipeline for one group of objects. This parallelisation strategy has one immediate disadvantage. If the scene consists of only one object, then there is little point in using more than one processor. Consequently a world containing 16 objects, each with 276 triangles was used instead of the complex world used for previous tests.

Values obtained for the speedup for an implementation of this strategy are shown in Figure 8. The bandwidth limitation is particularly severe in this case. In the case where the

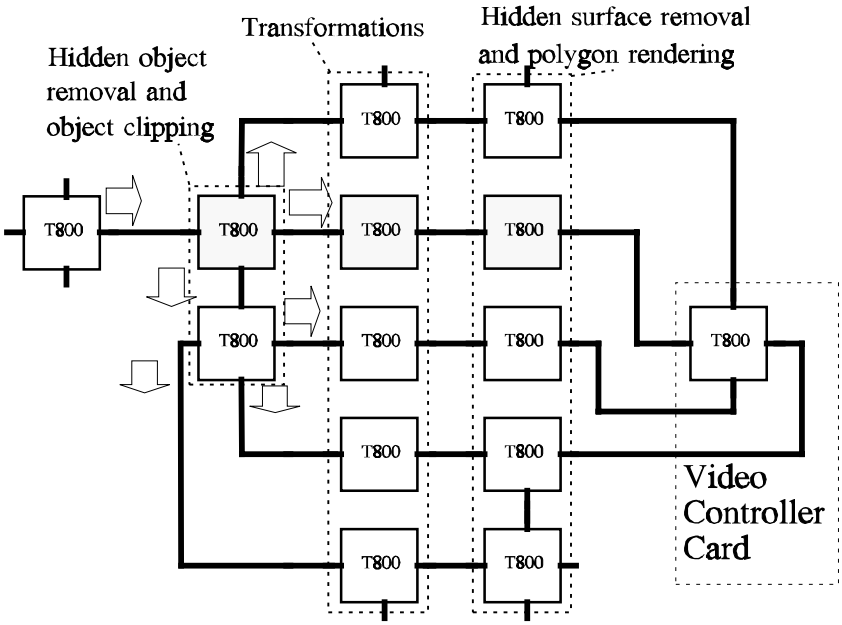
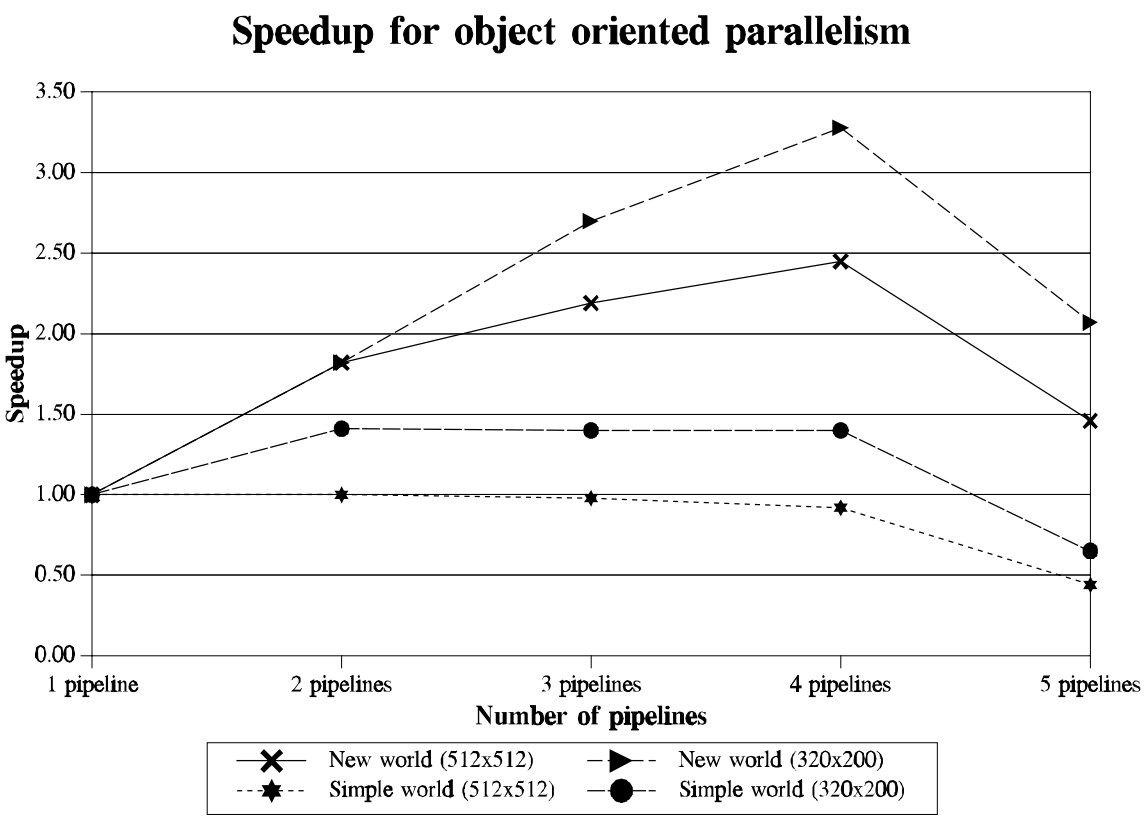


Figure 7 Processor layout for object oriented parallelism



	Frames/s for 1 pipeline	Speedup for 2 pipelines	Speedup for 3 pipelines	Speedup for 4 pipelines	Speedup for 5 pipelines
New world (512x512)	0.50	1.82	2.19	2.45	1.46
New world (320x200)	0.50	1.82	2.70	3.28	2.07
Simple world (512x512)	3.65	1.00	0.98	0.92	0.44
Simple world (320x200)	8.85	1.41	1.40	1.40	0.65

Figure 8 Speedup values for object oriented parallelism

number of processors used is less than five, an entire image needs to be sent across each of the transputer's links. This places an upper limit on the frame rate of 6.6 frames per second for 512 x 512 resolution or 27 frames per second for 320 x 200 resolution. The speedup values tend to level off as this barrier is reached. Adding a fifth processor only exacerbates the situation.

A further problem with this technique is load balancing. Some objects are more complex than others, and just distributing equal numbers of objects may result in some processors having to do more work. A finer, more balanced approach would be to distribute on a polygon basis. This could be implemented relatively easily. However such an approach is not worth exploring until higher bandwidth systems are available.

The parallelism would also suffer when the number of visible objects is less than the number of processors. Polygon distribution will be valuable for this situation as well.

The simplified version of the graphics pipeline used in the previous latency calculation, gives the total time taken to traverse the pipeline as (K - 2)T + 2T/N, since two stages of the

pipeline are parallelised. The latency is less than that for the fine grained decomposition and also decreases as the number of processors is increased.

4.5. Parallel rendering in practice

With each of the techniques implemented, analysis of run-time profiling information showed that processor time was being wasted in waiting for other processors to catch up. Processors were blocking, waiting for others to communicate. This problem was overcome by buffering incoming messages, enabling the process sending the data to continue with other processing. A notable speedup in the frame rate was achieved.

This solution had a corresponding disadvantage. Adding extra processes on the communication links has the effect of lengthening the pipeline. This increases the latency. The size of the buffers on each link also affects the latency adversely.

In order to get the maximum benefit from this technique the size of the buffers should be related to the complexity of the scene. Small buffers are better suited for scenes with a few complex objects, while larger ones smooth over the delays for large numbers of simple objects.

5. Conclusions

This paper describes a number of different parallelisation strategies that were tested in an attempt to increase the rendering speed and decrease the interaction latency in virtual reality systems. The strategies explored tended to exhibit a tradeoff between speedup and latency. A frame per processor approach gave almost linear speedup, but left the latency unaffected. Strategies which distributed the calculations for a single frame over a number of processors resulted in latency decreasing as the number of processors increased. These techniques did, however, have a lesser speedup.

A significant factor in the tests carried out was the limited bandwidth of the transputer links. An increase in the communication speed could contribute toward producing acceptable rendering speeds with some of the lower latency strategies. In particular, the technique which rendered slices of the image on different processors proved to be the most satisfying to use in practice. The object per processor strategy will become acceptable once the communication bandwidth increases.

References

[1] Airey, J.M., **Rohlf**, J.H., and **Brooks**, F.P. Jr., "Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments", *ACM SIGGRAPH Symposium on Interactive 3D Graphics*, **24**(2), March 1990, 41-50.

[2] Foley, J.D., and **Van Dam**, A., "Fundamentals of Interactive Computer Graphics", Addison-Wesley, 1984.

[3] Hearn, D. and **Baker**, M.P., "Computer Graphics", Prentice-Hall (New Jersey), 1986.

[4] Pausch, R., "Virtual Reality on Five Dollars a Day", *Proceedings of the ACM SIGCHI Human Factors in Computer Science Conference*, New Orleans, April 1991.

[5] Whitman, S., "Parallel Graphics Rendering Algorithms", *Proceedings of the 3rd Eurographics Workshop on Rendering*, May 1992.