

A Flexible Approach to Parallel, Real Time Graphics.

R. J. Cant, Department of Computing, The Nottingham Trent University, Burton St., Nottingham NG1 4BU, United Kingdom. Email rcc@uk.ac.ntu.doc.

Summary

Real time graphics applications are varied but are often implemented on standard hardware systems which fix the algorithms that must be used. To better match the algorithm to the application a parallel, software based, architecture is proposed. Whilst it is expected that critical parts of the software will be re-implemented for each application, a default set of algorithms is suggested, together with a work distribution scheme. The processor loading implications of various levels of performance are analysed and the implications of future developments in hardware and algorithms are considered.

Keywords

Computer Graphics, Real Time, Parallel, Rendering.

(1) Introduction

Real time graphics applications are amongst the most resource-hungry in the whole computing field. Up until now the need for processing, bandwidth and response time capability has almost always been met by specialised hardware design. For a discussion of such a design see [1]. As a result of this, software and system designers are not always able to use the most appropriate algorithms for the task in hand - since they must always use what their hardware supports. In the past, the designers of specialised graphics applications could often afford to develop their own hardware to overcome the limitations of standard systems [2] but the level of investment required and timescales have now made this impracticable.

The limited impact of graphics standards is also a consequence of this situation - since the "native library" of a graphics workstation always provides more power. The effect of this is made worse by the fact that the algorithms supported by hardware have not remained constant over time - resulting in a "temporal portability problem" as well as the usual spatial one.

For example, early versions of graphics workstations did not provide Z buffering in hardware and so real time applications would have been forced to use other methods of hidden surface removal which require structured databases. Later the hardware Z buffer became available and these databases became obsolete.

To overcome these difficulties we must take a new approach to achieving the performance levels required by real time graphics. If special purpose hardware is discarded, the only available alternative is to use general purpose hardware in a parallel architecture. In the past all attempts to move in this direction have had at least one of two major defects. Either the system created has not been capable of real time response (usually because of bandwidth limitations in interprocessor communications) or the hardware configuration adopted had the effect of pre-determining the algorithms which could be used in exactly the same way as with a special purpose hardware system. We must define a set of requirements for our design which specifically exclude these possibilities.

(2) System Requirements and Expected Benefits

We are looking for a parallel graphics architecture which is real time capable, algorithm neutral and scaleable.

Algorithm neutrality will hopefully provide a good design "match" over a wide range of applications.

The scaleability requirement is important because we recognise that algorithm neutrality will extract a heavy price in terms of reduced performance compared to "dedicated" systems. To compensate for this we need to increase the number of processors and, whilst the cost of this can be offset against savings in design effort, any absolute ceiling on system performance will be disastrous.

We would also like to use "off the shelf" hardware and to program in a high level language as far as practicable. This will provide portability and reduce the cost of migrating to newer technology as it becomes available.

Obviously such an architecture will be (in the first instance) less cost effective in terms of hardware than existing "dedicated" designs but as time progresses hardware becomes cheaper, whilst the (re)development costs saved will become more and more significant.

(3) The Implications of Algorithm Neutrality

One objection that might immediately be raised to this concept is "if one is to be algorithm neutral - than what progress can one actually make?"

There are two answers to this. The first is that the implementation of graphics algorithms in a parallel architecture in itself gives rise to a relatively limited number of ways in which work can be distributed. One of our aims is thus to define a work distribution scheme which has minimal impact on algorithm choice.

The second is that graphics algorithms follow certain common patterns even though their detailed design may differ (i.e. design structures may be the same even if the meanings of their components vary).

Given that the system is to be algorithm neutral the items which we will provide will be different from the usual software package or hardware system. There are two categories, long lived and short lived items. The long lived items are (in order of decreasing generality):

- (i) a prescription of how hardware should be connected together;
- (ii) a work distribution scheme;
- (iii) a basic graphics software package, provided in source code form which application developers can adapt to their needs.

The short lived items are:

(iv) hardware support to allow images to be displayed;

(v) software to allow (i) and (ii) to be implemented on a specific processor.

It is envisaged that the actual processing will make use of commercially available modular hardware systems. Currently there are two such systems on the market, the Transputer TRAM and the 320C40 TIM. It should be emphasised that nothing in this work will actually restrict implementation to these two systems. Any system with suitable connectivity and bandwidth would be useable.

Certain choices about algorithms will also have to be made. These choices will be made on the basis of providing maximal decoupling between one part of the algorithm and another and providing maximal generality.

For example it should be possible to change the shading algorithm without affecting the rest of the design and the number of possible shading algorithms supported by the design should be maximised.

This emphasis on generality does not mean that the sample system will attempt to be "all things to all men" at the detailed level. There will not be a large number of setup options and adjustable parameters to enable the system to be "customised" to the requirements of an individual application. This kind of generality usually creates more problems than it solves because it involves the creation of a new "language" which the user must learn before the system can be used effectively. Instead we will use algorithms which have wide applicability. Where customisation is required the intention is that it should be done by the application developer re-writing that part of the code which needs to change. This imposes a constraint of simplicity on the implementation.

(4) Parallelism and Graphics

Parallelism in graphics (and for that matter in any application) can be implemented in two ways. The task itself can be subdivided and allocated to separate processors or the data set can be subdivided and each piece given to a separate processor. In graphics there is a further choice within the "data subdivision" method in that either the input data (polygons or other geometric structures) or the output data (pixels or screen regions) can be used. It is also possible to combine any two - or all three of these subdivision principles within the same system.

The traditional, dedicated, graphics system does in fact use all three. At the top level there is an algorithm based decomposition which is implemented as a pipeline structure. Within each process there may be further - data based - parallelism. At the top level the input data set will be subdivided whilst the rendering process is likely to be accelerated by some kind of screen subdivision, with a processor being allocated to each part. In addition to this it is usual to employ specialised processors for the different parts of the pipeline in order to maximise performance. This last point is the main cause of the lack of flexibility (in terms of the ability to vary algorithms) of the traditional workstation but it is important to notice that some aspects of the parallelisation scheme also act as a constraint. In particular the algorithm based decomposition dictates the overall "shape" of the algorithm to be used and even prevents detailed changes, if they affect the overall workload of a subtask. The various data based parallelisms add to the inflexibility because of their incorporation into the the pipeline scheme. (They increase the specialisation of the pipeline steps that incorporate them.)

If we eliminate algorithm based parallelism as an option then we must choose some form of data based parallelism. The sheer size of the rendering task in real time graphics makes a purely input data based scheme difficult to construct unless the screen coverage of individual data items can be constrained in some way. This kind of constraint undermines what we are trying to achieve and so we are forced into output data based (screen based) parallelism as a necessary component of the current project.

Although this procedure is not completely "algorithm neutral" it cannot be too damaging because the process of scan conversion (the essential part of the rendering process) is itself a screen subdivision process (dividing the screen into lines and pixels) so one is merely anticipating what must happen later anyway.

This kind of parallelism on its own is inadequate because it cannot take effect until there is knowledge of where on the screen each component of the image will be. Therefore it does nothing for the early steps in the process. We must therefore speed up the early part of the graphics pipeline (geometry transformations and the like) by using input data based parallelism. At some point in the system this parallelism must be removed in order that a complete image can be displayed. If this is done early it will form a constraint on the methods that can be used since it enforces a certain "shape" on the algorithm and will probably fix the intermediate data format in which this process happens. The best way out of this problem is to leave the recombination until the very last minute - at which point we know that the data will be in the form of pixels. This technique has the further advantage that it permits different algorithms to be used for different objects within the scene.

We therefore arrive at an architecture in which processors are arranged in a rectangular array with one direction corresponding to different parts of the screen and the other corresponding to different objects or scene components. This arrangement for a simple system with the screen divided into four quadrants and three objects is shown in Fig. (1). The unconnected links at the top of the diagram go to the output hardware, which must contain some multiplexing capability to create a single image from the four separate parts. The links at the left hand side are used to provide input to the processors. Note that the upward going links must have sufficient bandwidth to allow an image to be created from their output at at least the frame update rate of the system. It is preferable to be able to output an image from the system at the field rate of the display to avoid an unnecessary transport delay being introduced.

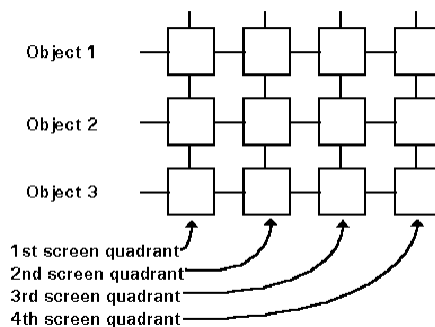


Fig. (1) Hardware Architecture.

This architecture is not particularly efficient if implemented on current hardware but it has three great virtues.

(i) It can be used with any real time graphics algorithms. Because each processor carries the complete set of algorithms any change to the system is a straightforward coding operation with no particular hardware consequences.

(ii) It is totally scaleable. An increase in rendering performance (in terms of resolution, depth complexity or rendering sophistication) is possible simply by increasing the number of processors in the horizontal direction. Conversely a greater capability in terms of numbers of objects rendered can be achieved by increasing the number of "layers" in the system.

(iii) The only major dedicated hardware required is in the output multiplexer, which in itself is a relatively "timeless" device needing to change only with advances in video display formats. Given a processor with a suitable link architecture, system porting should require only recompilation of code (with a few adjustments to "hard" addresses and the provision of an interface to the multiplexer). Systems using a shared memory architecture could also be constructed in principle but bandwidth constraints would limit scaleability.

(5) Graphics Algorithms

Although we wish to have an algorithm neutral architecture we need to provide default software for two reasons. Firstly we need to show that the idea works and secondly most users of the system will not want to supply all the code themselves, only the parts which are "mission critical" for their particular application. The algorithms which will be used fall into two categories: graphics algorithms and parallelism related (work distribution) algorithms.

Some graphics algorithms are intimately connected with the kind of image which is to be presented whilst others are more general and can be used over a wide range of applications. We will concentrate on the latter type here.

Firstly one must decide on the type of algorithm to be used. One can either use a screen based scheme (as is usually the case with ray tracing) or an object based scheme (the normal "graphics pipeline" approach). The disadvantage of the screen based scheme is that each processor must have access to the whole database. The problem of data distribution in parallel systems then emerges. The optimisation of such systems has been considered in the past [3]. In a real time system however, the implied random, high bandwidth, data movement is undesirable. Data movement in real time systems must be regular and predictable if it is to be managed successfully. The upshot of this is that we are forced to concentrate on the object based approach.

Having decided on a scheme which starts with database objects and then progresses to pixels at the end we must also fix the order in which the various sorting operations occur. This is intimately connected to the hidden surface removal algorithm to be used. Denoting the three sorting keys as P (sort by line of sight priority), X (sort by pixel in the X direction) and Y (sort by pixel in the Y direction) there are three different orderings to consider together with the possibility that the sorts could be "bundled" in some way.

(i) XYP

This group of algorithms includes the commonly used Z buffer. The idea is to scan convert each database object into individual pixel contributions before addressing the problem of hidden surface removal. The hope is that the pixel-pixel comparisons are simple and hence can be performed quickly.

(ii) PXY

This is what Sutherland [4] called the priority list approach. It includes Newell's and Schumacker's algorithms. These algorithms perform well in anti-aliased applications because the depth comparisons are not compromised by quantization. The most efficient and reliable of these algorithms rely on pre-processing the database in ways which may not be possible for all applications. Without pre-processing there is always the possibility of the sorting algorithm being caught out by a particular configuration of objects

(iii) YPX and bundled sorts

Algorithms in this group have been used sporadically throughout the history of real time graphics. In the early days they were used mainly because they afforded the possibility of avoiding the need for a frame buffer. More recently the motivation has been to eliminate the rendering of multiply covered screen areas. This category includes the span buffer method and algorithms which maintain "Y ordered lists" of face data. There are also algorithms in which all three sorts occur together, the best known of which is Warnocks algorithm.

In principal we would like to support all of these methods because different applications may be more amenable to one or the other. It is worth noting, however, that there will rarely be a direct reason to make a particular choice, and so it may be useful to construct an algorithm of our own which is a good match to the hardware architecture. In doing this we must be careful to cater for as many of the indirect reasons which normally lead to a particular choice as possible.

The influences on our choice will include: support for arbitrary database (environment) structures; support for other aspects of image quality; compatibility with parallelism; decoupling from other algorithms; low and consistent execution times and spin offs such as rangefinding, collision detection etc.

In the current context the last consideration can probably be ignored but the others must all be addressed.

The best algorithm for flexibility is the Z buffer since it makes no demands on the general structure of the other parts of the graphics pipeline. Unfortunately, as has been remarked many times before, it is very difficult to combine with proper sampling. PYX algorithms support sampling well but either require extensive database preprocessing or have poor or unpredictable performance when presented with complex environments. They also require that the ordering of objects is maintained as they are rendered - creating problems for a parallel system. Some YPX and bundled algorithms also suffer from this latter problem and they all introduce considerable extra complexity into the scan conversion process.

It follows that none of the commonly used algorithms provides good support for all of the requirements in a cost free way. We must thus think in terms of modifying one of the algorithms to achieve a better fit. The choice lies between modifying the Z buffer to enable it to cope with sampling properly and modifying the space partitioning version of the PYX algorithm to allow it to cope with a more arbitrary database.

The Z buffer suffers from two specific problems which make it incompatible with proper sampling. The first problem is caused by the fact that the depth is itself only sampled at one point. The second problem arises from the fact that only a single "running total" is kept of the contributions to each pixel. If there are two contributions to a pixel then their shade values will be irretrievably mixed and cannot be unscrambled should a third contribution arrive which is intermediate in depth.

To overcome these difficulties two approaches can be taken. The first approach is to antialias by supersampling. This is simple but inefficient since it scales the rendering workload up by a noticeable factor. It is only really viable in the context of a dedicated hardware system. It is also inadequate in the case where there are a large number of small surfaces within one pixel. The second approach is to generate and retain more information about each pixel contribution and to hold on to more contributions until the status of each is clear. Experimental coding in this area has shown that a "perfect" implementation of this idea is also inefficient but that in many practical situations an adequate solution can be found which is likely to be more efficient than supersampling.

The basic problem of the space partitioning PYX algorithm is the need to pre-structure the database. This is impracticable in applications such as CAD where the database is continually changing but can be done relatively easily in simple flight simulators where the viewpoint moves around in a pre-determined environment. There are also a number of in between cases where pre-defined objects move around a pre-defined environment and only the object interactions need special treatment. It is possible to extend the algorithm to cover such cases, with each object being separately pre-structured. Provided the geometric relationship between objects remains simple then the algorithm will remain efficient.

Since neither of these algorithms seems to be universal on its own perhaps the best compromise is to combine them. We propose therefore to divide the database up into objects which must be internally pre-structured to allow space partitioning methods to be used for intra-object hidden surface removal. We will also calculate and record depth values so that priority comparisons between objects can be performed on a pixel by pixel basis. Since many of the problems of the space partitioning PYX algorithm relate to the movement of objects whilst most of the Z buffer problems arise from the detail within objects the two algorithms cover each other's weaknesses. The resulting software will also be easily modifiable to pure Z buffer or pure space-partitioning if required. By allocating "objects" directly to the layers in our parallel scheme we also achieve a neat fit with the architecture.

(6) Work Distribution Algorithms

The distribution of work between processors must be done in two directions in our proposed architecture. Firstly we must consider the allocation of tasks to the horizontal layers of processors associated with object space parallelism. To maintain the advantages of our proposed default hidden surface removal method then all of the data processed by a given layer must belong to a single pre-structured object. This will not normally result in an efficient distribution of work however since objects are likely to be uneven in "size". We would like to be able to use a farm strategy to allocate

work to the layers. If we are to do this it means that the objects must be smaller so that each layer can deal with several of them. Processing several objects within one layer implies that contributions that belong to different objects may need to be kept apart until the end of the rendering cycle when all the object contributions are put together.

Allocation of work within the layers is more straightforward but there are still choices to be made between the advantages of coherence which come from allocating each processor to a contiguous area of screen and the more even work distribution that results from a scatter decomposition. The object farming scheme described above makes a scatter decomposition more attractive since an even workload across the screen will not normally be possible for individual objects even though it may exist for the scene as a whole. This must be balanced against the extra programming complexity which it requires and hence the issue cannot be resolved in general. The output multiplexer must thus support both schemes.

(7) System Performance

We have analysed a number of different rendering schemes in terms of the performance which they require from the system - for a given level of scene complexity. The results presented here are a mixture of figures derived from experience of existing real time graphics systems, figures derived from experimental coding and estimates. For each rendering scheme figures have been calculated for the per-pixel, per-scanline and per-face workload. The total workload, in terms of processors equivalent to the Texas 320C40, has then been calculated.

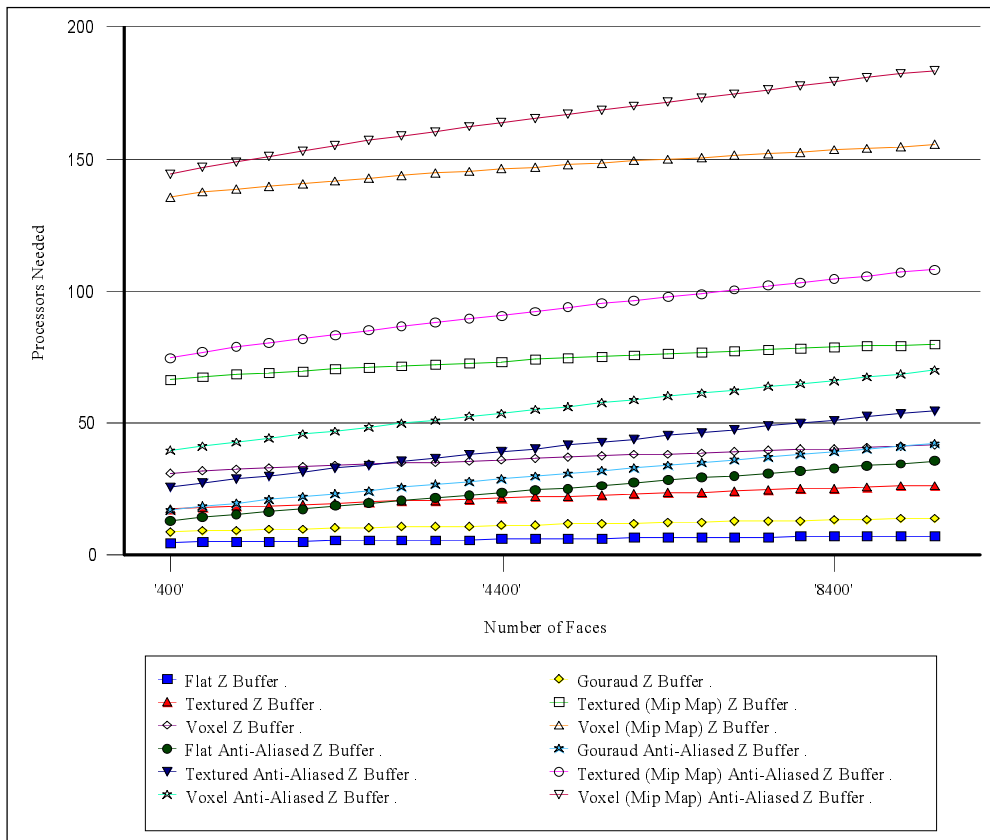


Fig. (2) Processors required as a function of number of faces.

Two hidden surface schemes have been considered, the Z buffer and an antialiased scheme using a Z buffer for inter-object comparisons as described above. Six different shading schemes have been considered: simple flat shading; Gouraud shading; texture, in simple form and with a mip-map antialiasing scheme and finally, as an example of the kind of specialised rendering scheme which can be supported by the present approach, a voxel based method of displaying spotlights with complicated beam patterns [5]. This last scheme is also shown in simple and mip-map form.

Fig. (2) shows processors needed as a function of the number of faces for a 500x500 screen with an average depth complexity of 3. In Fig. (3) the number of faces has been held at 4000 and depth complexity has been varied instead. This is equivalent to varying the resolution. From Fig. (2) it is clear that the processing requirement depends critically on the algorithm but is insensitive to the number of faces. Fig. (3) shows that processor loading is very sensitive to depth complexity or resolution. This is because non-specialised processors are good at geometry but relatively poor at rendering.

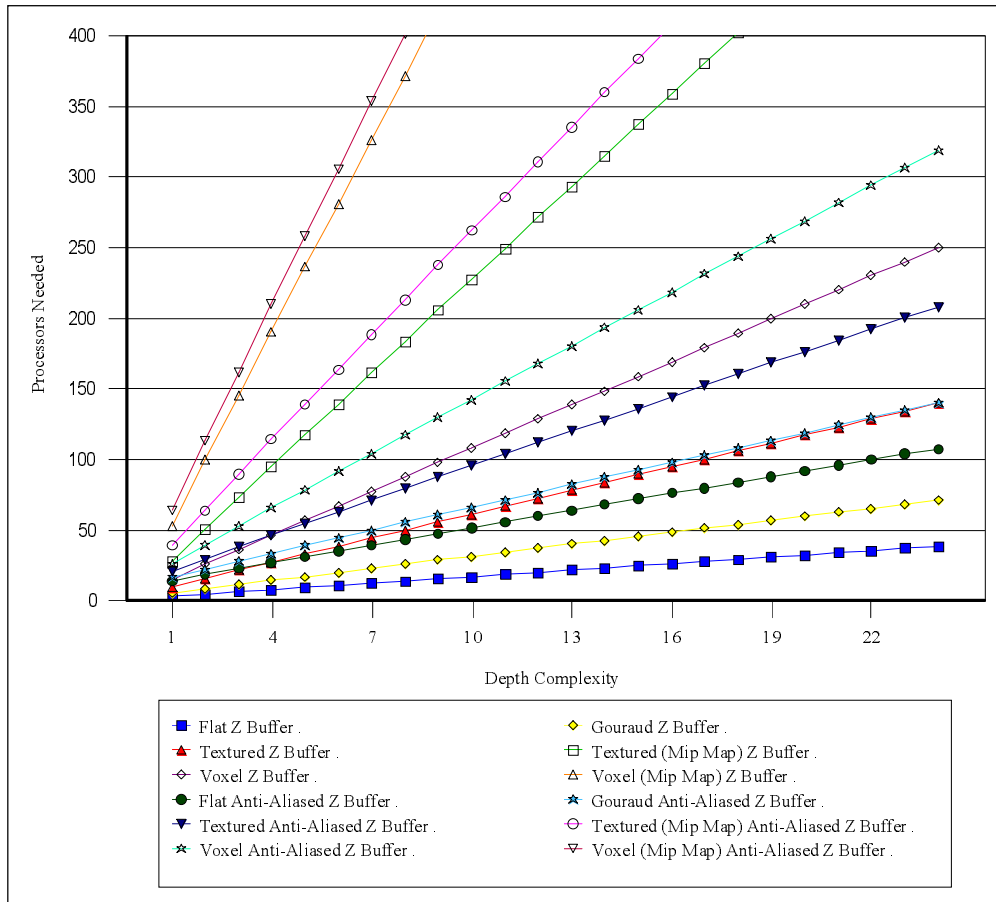


Fig. (3) Processors required as a function of depth complexity.

(8) Conclusions

Clearly a system which incorporates the most sophisticated rendering scheme at a high resolution will be impossibly large with current technology but by confining the more sophisticated techniques to those parts of the image which the application needs, costs can be contained. A 64 processor system could be cost competitive with high end dedicated graphics systems and would allow non-standard rendering techniques to be used in critical areas. Such a system will not need re-developing as technology progresses and can take advantage of higher processor performance either by improved system capability or by reduced cost.

References

- [1] Molnar, S. E., J. Eyles and J. Poulton "PixelFlow: High Speed Rendering Using Image Composition," SIGGRAPH '92, Vol 26, No.2, P23.
- [2] Cant, R.J. and P. Sherlock, "CIG system for periscope observer training," Proceedings of the 9th Interservice/ Industry Training Systems Conference (1987) P 311.
- [3] Green, S. A. and D. J. Paddon, "Handling Graphical Databases in Parallel Architectures" Proceedings of BCS Seminar "Parallel Processing for Display" (1989) P1.
- [4] Sutherland I.E. , R. F. Sproull and R. A. Schumacker " A Characterization of Ten Hidden Surface Algorithms", Tutorial Computer Graphics , (1982) P387.
- [5] Burton N.D. "Algorithms for Advanced Light Sourcing in Real Time," The Nottingham Trent University Department of Computing Project Report (1994).