An Architecture for Interactive Raster Graphics

A.A.M. Kuijk, E.H. Blake and P.J.W. ten Hagen

CWI
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
Email: fons@cwi.nl

ABSTRACT

A radical reappraisal of the 3-D interactive raster graphics pipeline has resulted in an experimental architecture for a workstation which is currently being evaluated at the CWI. The principal features of this architecture are that it:
- concentrates exclusively on real-time interactive 3-D graphics (initially for CAD).
- uses object space rather than image space methods where possible.
- avoids using a frame buffer.
- only uses custom VLSI where commercial products are unlikely to suffice in the near term.

Four years into the project the system design is complete and the major components have been acquired and the custom VLSI chips have been packaged and tested. The current experience with the system is based on detailed simulations which gave a fairly clear idea on its strengths and limitations. A complete, but reduced resolution, experimental prototype system is now being assembled.

1991 Mathematics Subject Classification: 68Q10, 68Q80, 68U05.

Key Words and Phrases: raster graphics, interactive, architecture, display controller, systolic array, object space hidden surface removal, angular interpolation.

1. Introduction

A number of different feedback levels in the image synthesis pipeline can be identified if one takes a new look at the basics of high quality three-dimensional (3-D) raster graphics [Hagen87] for CAD. A user interacts with the visible parts of a 3-D model at each of the levels (Figure 1.1). We provide direct access to graphics objects to support pointing and identification. These are the fundamental actions that underlie every change a user makes.

Changing pictures form the key to the architecture. Actual pixels are not needed for interaction. If we take this observation seriously and ruthlessly pare away other elements we get a radical prescription for a graphics architecture. One where the visible surfaces of objects are explicitly identified and without any mandate for a frame buffer. We believe that our research shows that such a machine, which harkens back to the calligraphic roots of graphics displays, can be built [Akman88].

We report here on an ongoing project to build such a system. In the past four years the design has been completed, the critical components have been made [Jayasinghe91b],

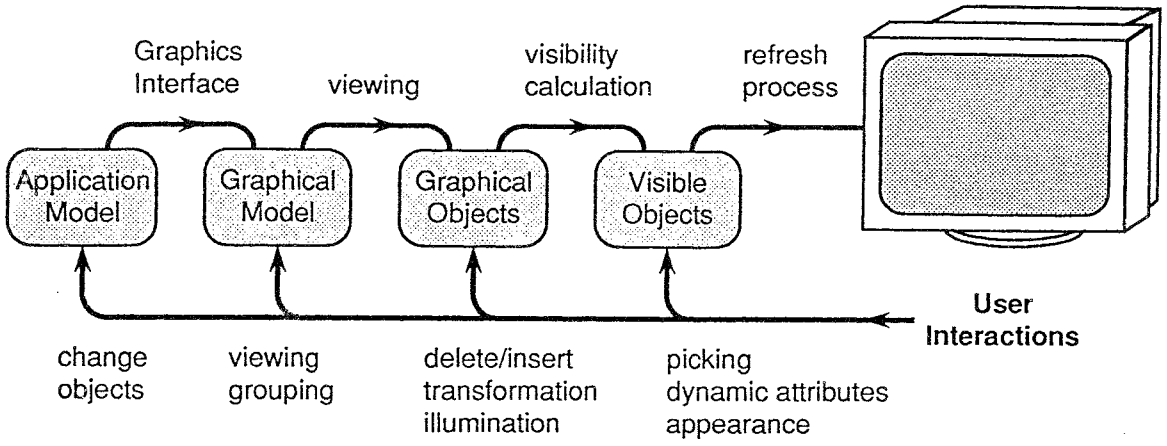

Figure 1.1: Graphical data that are relevant for interaction.

and functions simulated in detail [Guravage92]. This paper is a description of the complete design and early experiences. The rest of this section presents an outline of the architecture. The major new components follow from the bottom up: first, the display controller comprising the pixel generator (or X-processor – Section 2.1) and the shading processor (or Y-processor – Section 2.2), and then hidden surface removal (Section 3). In Section 4 the system is evaluated in the light of our experience so far and with reference to its antecedents and current alternatives.

The functional elements of the display architecture are shown in Figure 1.2. In our architecture the frame buffer is replaced by a structured list of objects which can be 'pointed' at refresh rate. To this end a powerful VLSI-based hardware block containing a systolic array of processors produces a full colour pixel stream at video refresh rate. This block is fed directly from the structured object list. As a result a selected object can be highlighted or moved instantaneously. The processors are also capable of producing Phong shaded 3-D objects or 2-D textures at this rate.

The size of the structured list dependent on the image complexity. For reasonable complex images the size is about the same as a conventional frame buffer, with the advantage that the object list is resolution independent (it does not become larger for higher resolutions). The structured list is also a much more organized data structure and can support more sophisticated operations than a frame buffer.

At higher levels of the architecture the objects become more complex (but also fewer – less fragmented) with information about light sources, textures and viewing. Here representations for incremental changes typical of real-time interaction are favoured. These requirements appear to be satisfiable in the short term by powerful but off-the-shelf parallel hardware. At the lowest level custom components are needed: these have already been built.

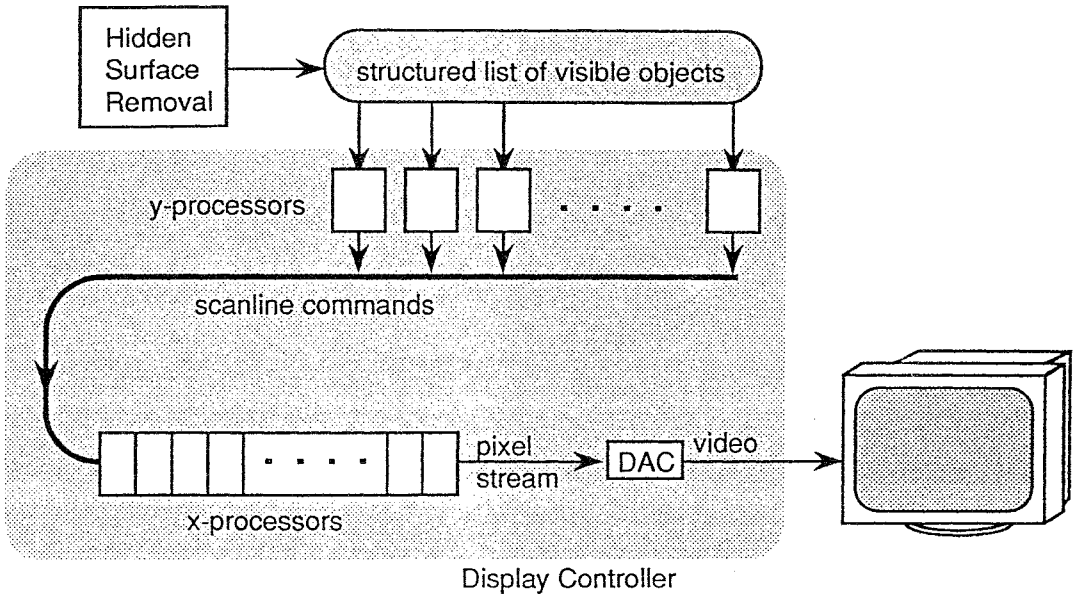

Figure 1.2: Functional Elements of Display Architecture. The Y and X processors are part of the display controller.

2. Display Controller

The graphics requirements are essentially real-time support of the Phigs+ standard, with Phong shading as the best quality shading method. This degree of realism must be achieved at 50–60 frames per second in an interactive setting where the user is free to alter models, position of the lights and viewpoint. The current state-of-the-art is real-time Gouraud shading. The next hurdle in real-time realism is therefore the familiar Phong illumination equation.

The refresh process of a graphics display device is performed by a display controller. In calligraphic systems this module interpreted line drawing instructions collected in a display list and used this to steer the electron beam (both location and intensity). In current raster graphics systems the function of a display controller is reduced to reading of pixel intensity information stored in successive memory locations of a frame buffer and transforming that information into the video signals.

In our system architecture the display controller interprets 'area drawing instructions' in the structured object list which describe non overlapping areas on the screen and their colouring information and transforms that into the video signals. The row by row nature of the video signal suggests splitting this scan-conversion process in the horizontal (X) and vertical (Y) direction.

This led to a display controller architecture with different types of processing elements (X-processors and Y-processors), and different levels of complexity for the algorithms running on them [Jayasinghe91a]. For each scanline, the intersection of objects with the scanline as well as the colour function along that scanline is calculated by an array of Y-processors (i.e., shading processors). These produce scanline commands which are send to a one-dimensional systolic array of X-processors (i.e., the pixel generator). These process and sort the commands and produce a stream of pixel values.

2.1. Pixel Generator

The pixels are generated by one one-dimensional systolic array of X-processors per colour. There is no space to introduce the full instruction repertoire of the X-processors here (see appendix A). The repertoire supports smooth shading, i.e. a continuous intensity and continuous derivative of the intensity. Thus, the X-processors generate quadratic interpolated intensities by forward differencing. Higher order interpolation (in fact any higher order) for the purpose of anti-aliasing and advanced illumination models is supported as well.

The processors use 12 bit pixel addressing and operate on 36 bit fixed point intensity values. This precision is sufficient for second order forward differencing along the designed maximum span of 4K pixels. For higher order forward differencing, the size of the spans should be limited to assure sufficient accuracy.

Commands for a scanline issued by the Y-processors enter the X-processor array on one side and travel through the array, currently at a rate of 12 ns per processor. Commands operate on a span of pixels, and results of several commands on a pixel accumulate. At fixed time intervals, a refresh command is presented to the array. Each processor issues the resulting 12 bit pixel value upon arrival of this refresh command and is ready to operate on commands for the next scanline.

2.2. Shading Processor

The information to drive shading- or Y-processors is contained in the structured object list. Y-processors produce the instructions for the pixel generator. The Y-processors operate at the frame refresh rate and go through a complete cycle once every video frame. Their input data are produced at the interaction or animation rate (between 12 and 24 cycles per second). The output goes to the pixel processors operating at the line refresh rate.

The task of these processors can also be described as having to change 2-D display information into 1-D scanline information. The third dimension has been dealt with earlier by projection and hidden surface removal. The geometry can be specified solely in terms of (2-D) display coordinates but (3-D) world coordinates are still needed for the shading calculations. The edges of the surface primitives (triangles or trapezoids) are simple enough to find. Shading (especially Phong shading) and anti-aliasing are more of a challenge and we give an outline of our methods to achieve the high speed necessary.

The major cost in Phong shading is the renormalization (via a square root and a division) and exponentiation that has to occur after each interpolation step. Look-up tables can be used for square roots and exponentiation [Bishop86]. but this costs storage and is not feasible in highly parallel systems such as ours.

Another approach to this problem is to recast the Phong shading model, but without lapsing into expensive Gouraud shaders or being unable to deal with all practical situations. Phong shading itself is nothing more (or less) than a very good practical approximation: it is not an end in itself. In [Kuijk89] we introduced a method for quadratic Phong shading via angular interpolation. It has the lowest per pixel cost in time and storage of any method we are aware of. (See appendix B for figures on computational costs).

Our approach to quadratic Phong shading depends on two major results:

1 A parameterized piecewise quadratic expression for the cosine of an angle t, $-\frac{1}{2}\pi < t < \frac{1}{2}\pi$, raised to a power n, $1 \leq n \leq 125$.

2 A linear expression in terms of the pixel position, x, on a scanline for the angle between the interpolated normal vector of the surface and the light or highlight vector.

We interpolate vectors by rotation rather than by the standard linear interpolation and renormalization. The difference is unimportant provided we remain consistent – both approaches are equally 'wrong' since both ignore perspective projection effects.

If either the light source or the viewer are not at infinite distance then we apply an approximation. In the case of diffuse illumination this can be understood intuitively as replacing a surface with a light source near it with a more convex surface with the light source at infinity. A similar approximation exists for specular reflection.

Anti-aliasing in the form of exact area integration is applied. Object space hidden surface removal preserves the necessary information on pixel coverage. Where pixel coverage gradually increases along a scanline this is treated as a linear modulation of the quadratic shading function. This results in a cubic expression which is passed on to the pixel generator.

3. Hidden Surface Removal

In our architecture, the hidden surface removal (HSR) takes place in object space. This 2½-D space is the result of a perspective transformation on the 3-D scene so that a simple projection along the z-axes produces the 2-D image. Hidden surface removal comes down to sorting. Therefore we make use of a pre-sorted representation of the objects and store them in a data structure on a location which reflects the x- and y-position of the object [Kuijk88]. The pre-sorted representation of the objects does not only reduce the overlap calculations needed for the HSR, but also simplifies the scan conversion process. The data structure in which objects are stored is designed to reduce the search space and to be able to (evenly) distribute the data for a multi-processor implementation. This makes the HSR and other operations which involve identification based on location more efficient.

The interactive applications envisioned require incremental picture changes. Known object space HSR algorithms [Franklin90] operate on a complete scene description and produce the complete set of visible objects. Therefore we defined a set of logical operations on the 2½-D objects so that individual objects can be added and deleted. These operations amongst others involve subtraction, overlap calculation and union of objects. All objects have attributes that influence the shading calculations. The objects that the logical operations produce inherit a logical combination of the attributes of the original objects. This feature allows the system to efficiently make objects invisible or transparent.

4. Discussion

The decision to start the implementation at the rasterization level was made because the technological critical elements are located there. At that level, special purpose VLSI can be relatively simple and is more likely to be competitive. This does not hold for higher levels. There the complexity is quite high which also makes the design effort high. Besides, general purpose elements are usually implemented on better (more expensive) technologies.

A VLSI based systolic array for pixel generation which was only capable of constant shading was introduced in 1985 [Gharachorloo85], this was later replaced by a version capable of Gouraud shading [Gharachorloo88]. Real-time Phong shading is also possible if enough hardware is dedicated to the problem [Deerin88, Fuchs89].

In cooperation with colleagues at the University of Twente we have produced a highly pipelined systolic array graphics engine capable of high speed quadratic interpolation using forward differencing with 36 bit arithmetic [Jayasinghe91b]. The current implementation has 9 processing elements per chip using a 1.6 μ CMOS process. Extensive simulation has indicated that the chips will run at 12–15 ns cycle time (equipment has not been available to test the chips, which have already been packaged, at this speed). It is estimated that an improved design using the same technology will be able to fit 50–60 processing elements on the same die with a slight improvement in speed.

The value of quadratic interpolation for shading was brought out in our paper on fast Phong shading [Kuijk89]. Uses of quadratic interpolation in shading are also documented in [Kirk90, Fuchs89]. In fact our architecture is optimized for quadratic interpolation only in the sense that 36 bit words allow accurate quadratic interpolation for spans up to 4096 pixels long. Higher order interpolation require shorter spans to maintain accuracy. Every extra order of interpolation requires one extra cycle of processing.

The maximum number of commands the X-processor array can accept per scanline equals the number of pixels on a full scanline. This relation between commands and number of pixels should on average hold for each individual object as well. This can be justified: all commands contribute to pixels that are visible, but it does not make sense to supply more information than can ever come out. However, in principle a system can be build which has multiple X-processor arrays and which therefore can spend multiple scanline times on a scanline.

5. Conclusion

The overall aim of our research is to develop a new architecture for an interactive workstation. A novel feature of this architecture is that there is no frame buffer – this was done mainly to improve interactive performance, but one could also argue that if 24 frames per second are needed then one might as well do 60 frames per second and get rid of the frame buffer which becomes a bottle-neck at high update rates. This feature has a number of implications, for example, that the smallest pick primitive becomes a visible surface and not a pixel, and that object space hidden surface removal is required. It also

means that fully shaded pixels that depend on multiple light sources have to be produced every 12 ns or so, depending on resolution.

The object space approach makes it possible to perform incremental updates. This reduces the amount of computations involved on the levels above the rasterization level and thus improves the interactive behaviour of the system.

The two-level implementation of the display controller (X- and Y-processors) makes it possible to scale the system in two ways. The number of Y-processors can be selected to meet the requirements on the maximum image complexity (i.e. number of objects). The number of X-processors can be adjusted to accommodate all resolutions (up to 4K pixels per scanline). The current maximum throughput rate of the X-processors is sufficient to support high resolution systems. It can handle the refresh of 2.46 M Phong shaded polygons of 5×5 pixels per second (60×41 K). Since this first version was a low cost design, it is expected that a redesign and a better technology will improve this even further. Besides, the throughput rate can be multiplied by using multiple arrays.

Appendix A : Instruction repertoire of the X-processor

SetI (x,I)	Set the intensity at pixel location x to I
SetdI (x,dI)	Set the first forward difference of the intensity at pixel location x at dI
SetddI (x,ddI)	Set the second forward difference of the intensity at pixel location x at dI
SetPI (x,dx,I)	Set the intensity at pixel locations x, x+dx, x+2dx.... to I
SetPdI (x,dx,dI)	Set the first forward difference of the intensity at pixel locations x, x+dx, x+2dx.... at ddI
SetPddI (x,dx,ddI)	Set the second forward difference of the intensity at pixel locations x, x+dx, x+2dx.... at ddI
Eval0 (x,dx,I)	Set the intensities between the pixel locations x and x+dx at I and disable the accumulation of intensities until the next Refresh command
Eval1 (x,dx,I)	Accumulate the intensities between the pixel locations x and x+dx. If I has been set use the set value.
Eval2 (x,dx,I,dI)	Interpolate and accumulate the intensities between the pixel locations x and x+dx by first order forward differencing. If I or dI has been set use the set values.
Eval3 (x,dx,I,dI,ddI)	Interpolate and accumulate the intensities between the pixel locations x and x+dx by second order forward differencing. If I, dI or ddI has been set use the set values.
Dis (x, dx)	Disable the accumulation of the intensities between the pixel locations x and x+dx until the next Eval command.
Acc_mode ()	Enable/disable accumulation of negative intensities.
Refresh ()	Output the accumulated intensity and reset the processor.
Nop ()	No operation

Table 1: Instruction set of the X-processors.

Appendix B : Time complexity of the shading algorithm

It is difficult to compare the practical complexity of various algorithms. We have chosen to express complexity in terms of the operations typically available on modern RISC architectures that have divisions, seeds for reciprocal square roots, etc. implemented. The salient feature is that there is a single standard cycle time for the basic operations – all are equal to the time of a multiplication, say. Specifically we have chosen the Intel® i860™ microprocessor as an example. Various typical operations were given the costs indicated in Table 2.

Operation		Elementary Operations	Flops	Total Operations
acos	arc cosine	9+ 6× 1sqrt	13+ 20×	33
ang	angle	11+ 9× 1sqrt	15+ 23×	38
cos	cosine	8+ 8×	8+ 8×	16
crp	cross product	3+ 6×	3+ 6×	9
div	division	1div	2+ 5×	7
dot	dot product	2+ 3×	2+ 3×	5
exp	exponent int	7×	7×	7
expf	exponent float	10+ 8× 1rsr	14+ 22×	36
len	length	2+ 3× 1sqrt	4+ 14×	18
ncp	normalized crp	5+ 12× 1rsr	9+ 25×	34
rsr	reciprocal sqrt	1rsr	4+ 13×	17
sin	sine	7+ 8×	7+ 8×	15
sqrt	square root	1sqrt	4+ 14×	18

Table 2: Standard Operations and their Complexity.

In Table 3 the angular method we found is analysed in terms of incremental costs incurred for preprocessing every facet (triangle or trapezium), and within a facet for preprocessing the scanlines and finally the incremental cost of generating pixels. The method of Bishop and Weimer [Bishop86] is given for comparison only. It should be emphasized that the large lookup tables required for that method in order to do exponentiation after quadratic interpolation make it unsuitable for a massively parallel implementation.

The "facet" costs are incurred whenever models change. The "line" costs arise for every scanline at the scanline refresh rate in the shading processors. The pixel rate is actually the computational work performed by the systolic array graphics engine.

Method		Directions constant			Directions varying		
		Facet	Line	Pixel	Facet	Line	Pixel
1	Bishop & Weimer	273	8	12	732	8	12
2	Angular, Trapezia	434	282	5	1080	353	5
3	Angular, Triangle	198	282	5	578	353	5

Table 3: Costs in Elementary Operations for the Shading Methods.

References

[Akman88] V. Akman, P.J.W. Ten Hagen, and A.A.M. Kuijk, "A Vector-like Architecture for Raster Graphics," in *Advances in Graphics Hardware II*, ed. A.A.M. Kuijk, W. Straβer, EurographicSeminars, pp. 137-154, Springer-Verlag, 1988.

[Bishop86] G. Bishop and D.M. Weimer, "Fast Phong Shading," *ACM Computer Graphics (SIGGRAPH '86 Proceedings)*, vol. 20, no. 4, pp. 103-106, August 1986. Corrections appeared in Computer Graphics vol.21, no. 4, pp. 53. Also in the expression for T4 on pp. 105 -fjlqr should be replaced with +fjlqr and -2ffnr with -2fllnr.

[Deerin88] M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt, "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics," *ACM Computer Graphics (SIGGRAPH '88 Proceedings)*, vol. 22, no. 4, pp. 21-30, 1988.

[Franklin90] W.R. Franklin and M.S. Kankanhali, "Parallel Object-Space Hidden Surface Removal," *ACM Computer Graphics (SIGGRAPH '90 Proceedings)*, vol. 24, no. 4, pp. 87-94, 1990.

[Fuchs89] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Isreal, "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System using Processor-enhanced Memories," *ACM Computer Graphics (SIGGRAPH '89 Proceedings)*, vol. 23, no. 3, pp. 79-88, 1989.

[Gharachorloo85] N. Gharachorloo and C. Pottle, "Super Buffer: A Systolic VLSI Graphics Engine for Real Time Raster Image Generation," in *1985 Chapel Hill Conference on VLSI*, ed. H. Fuchs, pp. 285-305, Computer Science Press, Rockville, Maryland, 1985.

[Gharachorloo88] N. Gharachorloo, S. Gupta, E. Hokenek, P. Balasubramanian, B. Bogholtz, C. Mathieu, and C. Zoulas, "Subnanosecond Pixel Rendering with Million Transistor Chips," *ACM Computer Graphics (SIGGRAPH '88 Proceedings)*, vol. 22, no. 4, pp. 41-49, 1988.

[Guravage92] M.A. Guravage, E.H. Blake, and A.A.M. Kuijk, "XInPosse: Structural Simulation for Graphics Hardware," in *Advances in Graphics Hardware VI*, EurographicSeminars, Springer-Verlag, 1992. to appear.

[Hagen87] P.J.W. Ten Hagen, A.A.M. Kuijk, and C.G. Trienekens, "Display architecture for VLSI-based graphics workstations," in *Advances in Graphics Hardware I*, ed. W. Straβer, EurographicSeminars, pp. 3-16, Springer-Verlag, 1987.

[Jayasinghe91a] J.A.K.S. Jayasinghe, A.A.M. Kuijk, and L. Spaanenburg, "A Display Controller for an Object-level Frame Store System," in *Advances in Graphics Hardware III*, ed. A.A.M. Kuijk, EurographicSeminars, pp. 141-170, Springer-Verlag, 1991.

[Jayasinghe91b] J.A.K.S. Jayasinghe, G. Karagiannis, F. Moelaert El-Hadidy, O.E. Herrmann, and J. Smit, "Two-level Pipelined Systolic Array Graphics Engine," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 3, pp. 229-236, March 1991. revised version of the paper in Proceedings IEEE 1990 Custom Integrated Circuits Conference, Boston, Massachusetts pp. 17.2.1-17.2.4.

[Kirk90] D. Kirk and D. Voorhies, "The rendering Architecture of the DN10000VS," *ACM Computer Graphics (SIGGRAPH '90 Proceedings)*, vol. 24, no. 4, pp. 299-307, 1990.

[Kuijk88] A.A.M. Kuijk, P.J.W. Ten Hagen, and V. Akman, "An Exact Incremental Hidden Surface Algorithm," in *Advances in Graphics Hardware II*, ed. A.A.M. Kuijk, W. Straβer, EurographicSeminars, pp. 21-38, Springer-Verlag, 1988.

[Kuijk89] A.A.M. Kuijk and E.H. Blake, "Faster Phong Shading via Angular Interpolation," *Computer Graphics Forum*, vol. 8, no. 4, pp. 315-324, 1989. Errata: on pp. 321 the definitions of a and b should be swapped.