HARDWARE ENHANCED 3-D RASTER DISPLAY SYSTEM

T. Whitted

Bell Laboratories
Holmdel, N.J.

ABSTRACT

There is a range of applications for shaded display of 3-dimensional scenes for which performance of 1/30th or 1/60th of real time (i.e. one frame per second) gives a sufficient level of interaction to be useful.

There have been attempts to provide this medium level of performance by adding special purpose microcode to conventional computers, or by placing bipolar microprocessors between the host computer and the display device. An extension of this approach is to directly convert the inner loops of the display code into special purpose LSI chips.

An NMOS chip has been designed (and fabricated) that executes the scan line visibility tests associated with the well known z-buffer algorithm, removing all of the pixel-by-pixel operations from the host. An identical chip is used to interpolate intensity values for smooth shading. A circuit containing both chips can be incorporated into a frame buffer memory.

Statistics gathered from a software simulation of a system using this circuit show that the chip provides a substantial gain when the average projected area of polygons in a scene is high compared to the number of polygons.

RÉSUMÉ

Il existe une gamme d'application pour affichage ombré de scènes tridimensionnelles pour lesquelles des rendements de 1/30^{e} ou 1/60^{e} du temps réel (c'est-à-dire un cadre par seconde) donnent un niveau suffisant d'interaction pour que le système soit utile.

Plusieurs tentatives ont été faites pour en arriver à ce niveau de rendement du support en ajoutant des microcodes spéciaux aux ordinateurs conventionnels ou en plaçant des microprocesseurs bipolaires entre l'ordinateur central et les dispositifs d'affichage. Une extension de cette approche consiste à convertir directement les boucles internes du code d'affichage en boîtiers LSI spéciaux.

Un boîtier NMOS a été conçu (et fabriqué) pour exécuter les essais de visibilité de lignes balayées associés à l'algorithme bien connu Z-tampon en enlevant toutes les opérations pixel par pixel de l'ordinateur central. Un boîtier identique est utilisé pour interclasser les valeurs d'intensité pour obtenir du faible ombragement. Un circuit contenant des deux types de boîtier peut être incorporé dans un cadre de mémoire tampon.

Les statistiques obtenues à partir d'une simulation d'un système utilisant ce circuit montrent que le boîtier entraîne un gain substantiel lorsque la superficie moyenne projetée des polygones dans une scène est élevée par rapport au nombre de polygones.

Hardware Enhanced 3-D Raster Display Systems

Approaches to Enhanced Performance

The real-time synthesis of shaded images is a feature usually found only in flight simulators using special purpose hardware whose expense can be justified by the application [12,13]. Recently, several proposals have been made for cheaper real-time image generation hardware composed of arrays of cheap processors [5,9,3,11,2]. A neglected research area is the range of 3-D display applications which do not require and cannot justify the expense of real-time performance. These applications have been forced to get by with frame buffer memories attached to general purpose computers. Occasionally, graphics systems designers have enhanced the display system performance by interposing a small, dedicated satellite processor between the host and the display memory [4] or by writing special purpose microcode for the host computer [7].

For these lower performance systems, the z-buffer algorithm is a popular choice for hidden surface removal because of its flexibility and simplicity. Parke [10] provides a detailed analysis of the performance of z-buffer algorithms executed either entirely within the host machine, or partitioned between the host and a satellite. He assumes, however, that both the host and satellite are machines that execute single instructions sequentially.

This paper discusses the case of the z-buffer algorithm partially implemented in custom LSI integrated circuits which are capable of parallel execution of many of the algorithm's steps. Unlike other proposals for LSI implementations of visible surface algorithms, large arrays of chips are not an essential feature. The special purpose hardware consists of only two LSI chips - one to perform the visibility calculations and another to smoothly interpolate shading along a scan line.

Elements of the Algorithm

The z-buffer algorithm [1,8] is a simple technique for determining the visibility of 3-dimensional surfaces by comparing the depth (distance from the viewer) of sample points on the surface with the previously closest depth at that sample point. Sample points are chosen so that they correspond to pixel locations in the image plane. This requires the use of a memory large enough to store both intensity and depth values at each pixel. If the depth of the current surface at a pixel is less than the stored depth for that pixel, then both the intensity and depth values for the pixel are overwritten by the current depth and intensity values. If the current depth is greater than the stored depth, then some previously processed surface is between the viewer and the current surface and neither intensity nor depth are overwritten. After processing all surfaces, only the ones visible to the viewer are represented in the z-buffer.

Although the z-buffer algorithm can be applied to objects of any shape, for this discussion consider only polyhedral objects tiled by triangles. Absolutely any surface can be approximated with a mesh of triangles. Furthermore, triangles have the wonderful property of always being convex and planar.

There are two approaches to using the z-buffer algorithm. In the first, triangles are considered one at a time. As shown in figure 1, points of intersection of the polygon edges with successive scan lines are determined by $x_{n+1} = x_n + \Delta x/\Delta y$. The depth, z, of the point of intersection is determined in a like manner. (In image coordinates, the positive x axis is to the viewer's right, the positive y axis is up, and the positive z axis points away from the viewer.) For all scan lines intersecting a triangle, the region between the left and right intersection points is filled with intensity values corresponding to the relative orientation of the surface with respect to a light source. To approximate the appearance of a smoothly curved surface with the mesh of triangles, this intensity value, i, may be interpolated along the edges and between them in the same way that x and z are interpolated [6]. For a given scan line, depth and intensity are interpolated by adding $\Delta z/\Delta x$ and $\Delta i/\Delta x$ to the z and i values and accumulating the result at each pixel.

The second approach is to consider all polygons at once while proceeding in scan line order. All triangle edges are sorted by their maximum y value so that the processor can easily tell which new edges will be intersected by each scan line. When an edge is first intersected, it is added to a list of "active edges" which represent all edges intersected by the current scan line. The region between two active edges of a triangle is filled in exactly the same manner described for the first approach. The first approach is simpler, but the second is cheaper since it requires only enough z-buffer memory for a single scan line.

Performance Analysis

In his detailed analysis of the z-buffer algorithm's performance, Parke [10] splits the processing time into four components as shown below:

$$t_d = n_q{*}t_q + n_e{*}t_e + n_s{*}t_s + n_p{*}t_p$$

where

t_d is the time to display one frame,
n_q is the number of polygons,
t_q is the overhead time associated with one polygon,
n_e is the number of edges,
t_e is the overhead time associated with one edge,
n_s is the number of scan line segments,
t_s is the overhead time associated with one segment,
n_p is the number of pixels generated,
and t_p is the time to generate one pixel.

Parke decomposes each of the terms, t_q, t_e, t_s, and t_p, into low level instructions. He then makes some reasonable assumptions about the relative execution times of various instructions.

Rather than repeat Parke's analysis, the approach taken here is to actually implement the z-buffer algorithm and derive performance estimates from run-time statistics. The two-fold objective of this approach is to obtain actual numbers to check Parke's analysis, and to get an estimate of how effectively the hardware enhanced system will work in an

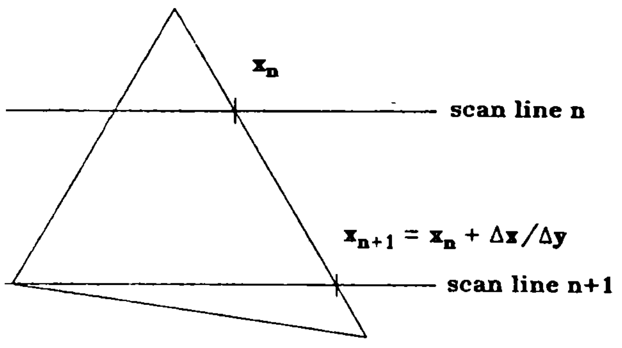


Figure 1. Incremental calculation of edge intersection point.

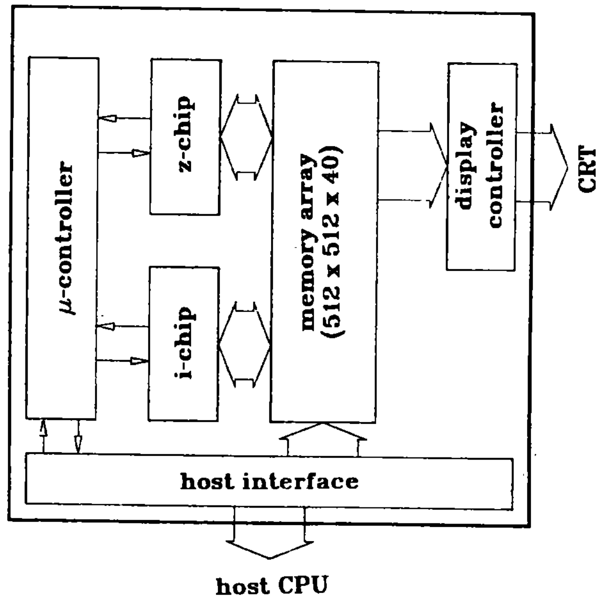


Figure 2. Enhanced frame buffer system.

interactive environment.

There are differences in Parke's statement of the z-buffer algorithm and the one presented here. For instance, since only triangles are considered, for i and z (the variables interpolated with respect to x), $\Delta z/\Delta x$ and $\Delta i/\Delta x$ need only be calculated once per polygon. As a result, the divisions needed to compute $\Delta z/\Delta y$, $\Delta i/\Delta y$, $\Delta z/\Delta x$, and $\Delta i/\Delta x$ are executed only once per polygon. Then there is a slight increase in t_q and a smaller t_s in the version of the algorithm described here.

A more significant difference is in the proposed implementation of the scan line processing. In an LSI implementation, the number of instructions executed has little meaning since the operations are performed in hardware, and in parallel. The only relevant numbers are the time required by the LSI chip to generate a pixel and the overhead time associated with initializing the chip for each scan line segment. These numbers have been obtained by exercising the chip in a test circuit.

The Chip

In its simplest configuration, a circuit containing two copies of the chip is installed in a frame buffer containing a sufficient number of bit planes to store depth and intensity for each pixel (figure 2). From its host, this circuit receives x_{left}, $x_{right} - x_{left}$, z_{left}, i_{left}, $\Delta z/\Delta x$, and $\Delta i/\Delta x$ for each scan line segment. Then for every pixel between x_{left} and x_{right} the chip interpolates z and i and compares z with the previous nearest z value for each pixel. For each pixel the chip will either overwrite the frame buffer with the updated z and i values or not depending on which z is closer.

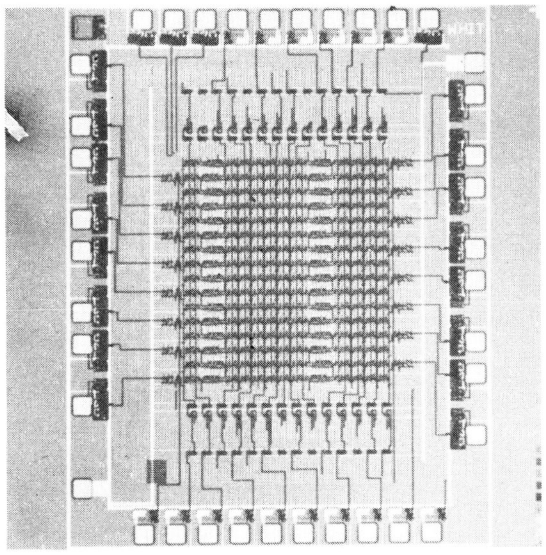
The chip basically consists of a counter, an adder, and several registers. The adder serves to increment the interpolated variable and to compare it with the z-buffer value. By the time the compare operation is completed, new z-buffer values will have been read into the chip. Therefore an additional stage of pipeline register is included for both the z-buffer data and the interpolated variable. The counter is loaded with $x_{right} - x_{left}$ and counts down for every pixel. The counter raises a completion signal when it reaches zero. The initial design, an NMOS chip of dimensions 290 by 240 mils containing 2200 transistors, is shown in figure 3.

All data paths are 16 bits wide, yet experience has shown that adequate image quality can be obtained with no fewer than 32 bits of z value. Rather than make the chip's data paths 32 bits wide, all 32-bit numbers are stored as high and low halves. An external latch saves the adder carry bit between operations to allow 32-bit adds and compares. Four clock cycles are then required to perform the 32-bit increment and compare functions. During these same four clock cycles the chip executes two z-buffer read accesses and two write accesses. If lower image quality is permitted, the chip can manipulate 16-bit numbers twice as fast.

Measured Performance

The host performance measurements presented here are run time statistics taken from a software implementation of the z-buffer algorithm which proceeds in scan line order by y-sorting all polygons in the scene (i.e. the second of the two alternatives described in the second section). The routines are coded in the C language for a VAX-11/780 [1]. The software is a subset of a "raster test-bed" package [14] developed for experiments in 3-D display. Since the routines are designed more for flexibility than performance, the host execution times listed in this report are not representative of

[1] VAX is a trademark of Digital Equipment Corporation.



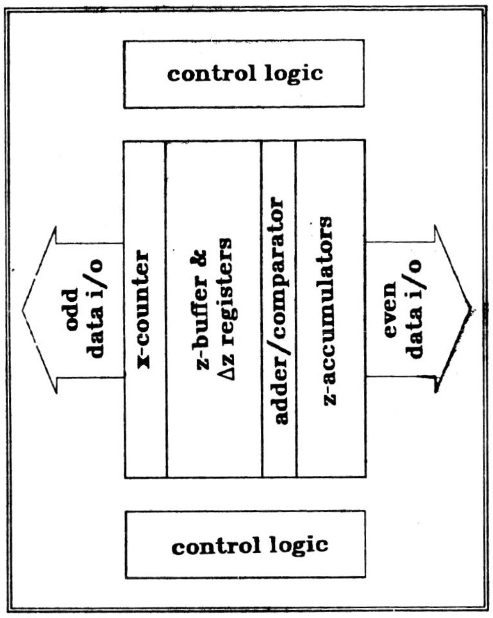


Figure 3. Chip layout.

an efficiently coded implementation. For the portion of the processing performed by the host computer,

$$t_{host} = n_q{*}t_q + n_e{*}t_e + n_s{*}t_s.$$

Note that t_s includes only the time required to compute $x_{right} - x_{left}$ plus the overhead of transferring the appropriate terms to the scan line processor.

If 32 bits of z are used in the visibility calculations and two LSI chips are included in the satellite processor, four clock cycles are required to generate a pixel. With rather conservative margins, the chips have been run at a rate of one cycle every 500 ηsecs. If $\Delta z/\Delta x$ and $\Delta i/\Delta x$ are loaded into their respective chips each time they are initialized for a scan line segment, then five clock cycles are required for the initialization. The expression for the chip's performance is then

$$t_{LSI} = n_s{*}2.5\ \mu sec + (n_p + 1){*}2.0\ \mu sec.$$

The one pixel overhead in the last term of the expression is for flushing the internal pipeline on the chip.

Although the host and satellite operate in parallel, the satellite cannot proceed without data from the host. Similarly, the host must wait for the satellite to finish a scan line segment before sending more data. To eliminate the waits, a FIFO must be included between the host and satellite. Then the expression for display system performance is approximated by

$$t_d = \max(t_{host}, t_{LSI})$$

as long as the FIFO doesn't overflow. In Parke's analysis, the point for which $t_{host} = t_{LSI}$ is called a balance point, i.e. the point of optimum match between host and satellite performance.

Execution times have been obtained for several objects characterized in table 1. Two different times are shown for t_{LSI} depending on whether the chip is used to initialize the intensity and z values for each frame. (Some commercial frame buffers can initialize all pixels in a single frame time.) The images shown in figures 4 and 5 each contain 8 triangles, but n_p is much higher for the 4 overlapping planes. In figures 7 through 10 a torus is approximated by increasing numbers of polygons. As more polygons are used, the number of pixels covered approaches an upper limit so that t_p is nearly constant. Figure 11 shows the effect on t_{host} and t_{LSI} of n_q (the number of triangles) for the torus.

Consider the case of processing one polygon at a time. There is a penalty for each edge shared between two polygons since the interpolation of x, z, and i along the edge must be performed separately for each of the two polygons. However, by expanding the chip with additional registers for x_{left}, x_{right}, z_{left}, $\Delta x_{left}/\Delta y$, $\Delta x_{right}/\Delta y$, $\Delta z_{left}/\Delta y$, Δy_{left}, and Δy_{right} interpolation of variables along the edges can be performed by the chip. Then communication between the host and satellite occurs only when the chip is initialized for a polygon and whenever the scan line passes the bottom of the left or right edge (whichever comes first) as shown in figure 12. For a configuration using this expanded chip, the per-pixel timing remains unchanged. There is, however, additional on-chip processing for each edge at each scan line, and the per-scan line initialization overhead is replaced by the per-edge overhead.

Object and Image Statistics									
name	polys	edges	vertices	pixels	segments	t_{host}	t^1_{LSI}	t^2_{LSI}	t^3_{soft}
sheet	8	16	9	82227	1001	0.26	0.167	0.692	4.5
planes	8	20	16	786486	3560	0.55	1.58	2.11	15.7
pair	520	816	298	211375	15581	2.5	0.462	0.987	10.9
torus6	72	120	49	105853	5052	0.68	0.224	0.750	5.7
torus10	200	320	121	123724	8880	1.3	0.269	0.795	7.3
torus16	512	800	289	129148	14580	2.4	0.298	0.820	9.1
torus30	1800	2760	961	131941	27370	6.1	0.332	0.857	14.7

[1] Excludes time to initialize depth and intensity for every pixel.
[2] Includes time to initialize depth and intensity for every pixel.
[3] Refers to time required by software without hardware enhancement.
All times are given in seconds.

Table 1. Characteristics of test objects.

As indicated above, the z and i values along the right edge of a planar polygon are redundant if $\Delta z/\Delta x$ and $\Delta i/\Delta x$ for the polygon are known. The edge initialization cost for a right edge is then only that of loading x_{right}, $\Delta x_{right}/\Delta y$, and Δy_{right} into the chip. For a left edge, five registers must be initialized. There is also the cost of loading $\Delta z/\Delta x$ and $\Delta i/\Delta x$ once per polygon. In the worst case of two left edges and one right edge for a triangle, there is a per-polygon cost of 7.5 μsec. For each scan line segment the chip must increment x and z for each edge and compute $x_{right} - x_{left}$. The time, t_s, required for these operations is 3.0 μsec. (Since the intensity calculations take place in parallel with z calculations, only one or the other need be considered for timing purposes.) Therefore, the time per frame for an expanded chip is

$$t_{LSI} = n_q*7.5\ \mu sec + n_s*3.0\ \mu sec + (n_p+1)*2.0\ \mu sec.$$

while

$$t_{host} = n_q*t_q.$$

Figure 13 plots the modified t_{host} and t_{LSI} for the same shapes used to create figure 11. The effect of this modification of the chip is to push the balance point towards the range of more complex objects. The absence of any dramatic gains for the more complex objects can be partially explained by the fact that the host must process edges shared between polygons twice instead of only once. For all of the objects shown, a large percentage of the edges are shared. As figure 13 shows, the additional edge processing transferred to the chip does not slow the chip appreciably. The real culprit is the code in the host that computes $\Delta x/\Delta y$, $\Delta z/\Delta y$, $\Delta z/\Delta x$, etc.

Summary

The class of display system described here utilizes a straightforward translation of software into hardware. From the measurements presented , one can conclude that this approach to the use of custom LSI for 3-D raster display will yield a useful level of performance. Despite the previously mentioned differences in implementation, figures 11 and 13 are similar to estimates provided in Parke's simulation. Although the performance statistics show that t_{host} is the dominant term for all but the simplest scenes, the chip provides at least a two to one increase in performance for each case tested. Because the custom circuits are installed in an ordinary frame buffer based display system, and because large numbers of the custom chips are not required, the approach promises to be cost effective as well.

References

[1] Catmull, Edwin, A Subdivision Algorithm for Computer Display of Curved Surfaces, PhD thesis, University of Utah, 1974.

[2] Clark, James H., and Hannah, Marc R., "Distributed Processing in a High-Performance Smart Image Memory", *Lambda*, vol. 1, no. 3, 1980.

[3] Cohen, D., and Demetrescu, S., A VLSI Approach to Computer Generated Imagery, Technical Report, Cal. Tech, 1979, oral presentation by Dan Cohen made at SIGGRAPH '80, Seattle, July 1980.

[4] Eastman, Jeffrey, "An Efficient Scan Conversion and Hidden Surface Removal Algorithm", *Computers and Graphics*, Vol. 1, No. 2, Sept. 1975, p. 215.

[5] Fuchs, Henry, "Distributing a Visible Surface Algorithm Over Multiple Processors", Proc. ACM Annual Conf., Seattle, Oct. 1977.

[6] Gouraud, Henri, "Continuous Shading of Curved Surfaces", *IEEE Trans. Cmptrs.*, C-20 (June 1971), 623.

[7] Jackson, J.H., "Dynamic Scan-Converted Images with a Frame Buffer Display.Device", *Computer Graphics*, 14,3, (July 1980), 163.

[8] Myers, Allan J., An Efficient Visible Surface Algorithm, Report to NSF, Grant No. DCR 74-00768 A01, Computer Graphics Research Group, Ohio State University, July 1975.

[9] Parke, Frederic I., "Simulation and Expected Performance of Multiple Processor Z-Buffer Systems", *Computer Graphics*, Volume 14, No. 3 (July 1980), pp. 48-56.

[10] Parke, Frederic I., Performance Analysis of Z-buffer Convex Tiler Based Shaded Image Generation, Tech. Rep. CES 79-15, Case Western Reserve University, October 1979.

[11] Roman, Grui-Catalin, and Kimura, Takayuki, "A VLSI Architecture for Real-Time Color Display of Three-Dimensional Objects", Proceedings of MICRO-DELCON, April 1979, pp. 113-118.

[12] Schachter, Bruce, and Ahuja, Narendra, "A History of Visual Flight Simulation", *Computer Graphics World*, Vol. 3, no. 3 (May 1980) pp. 16-31.

[13] Watkins, G.S., A Real-Time Hidden Surface Algorithm, PhD. thesis, Univ. of Utah, 1970.

[14] Whitted, Turner, and Weimer David M., "A Software Test-Bed for the Development of 3-D Raster Graphics Systems", to appear in Proceedings of SIGGRAPH '81, August 1981.

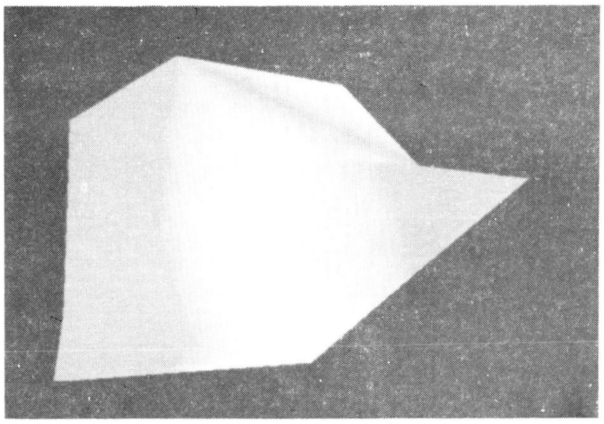


Figure 4. Warped sheet.

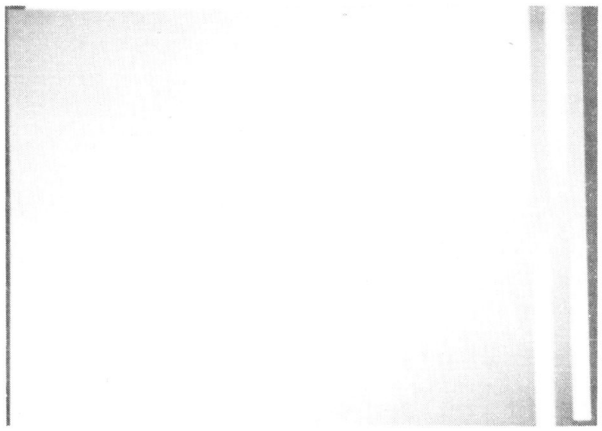


Figure 5. Four parallel planes.

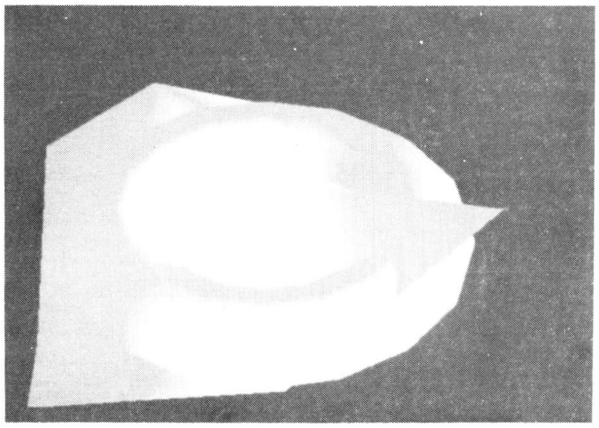


Figure 6. Pair of objects.

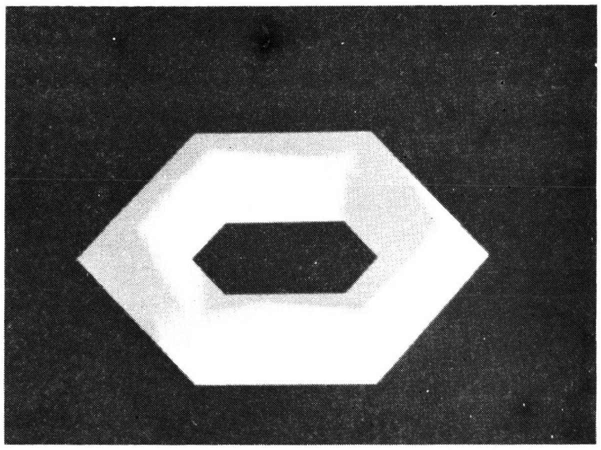


Figure 7. Torus6.

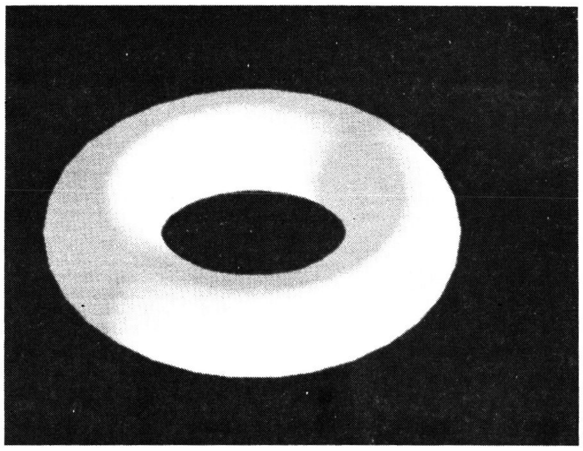


Figure 10. Torus30.

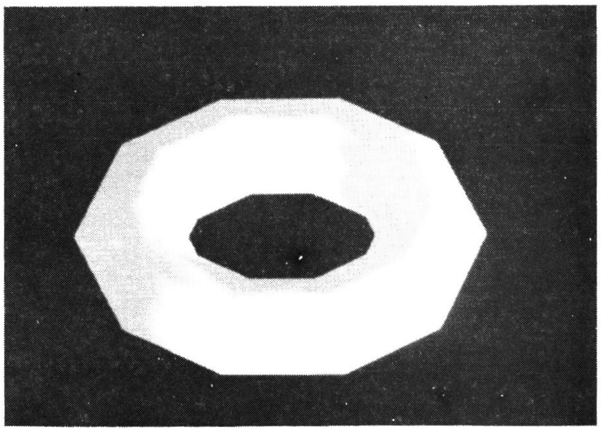


Figure 8. Torus10.

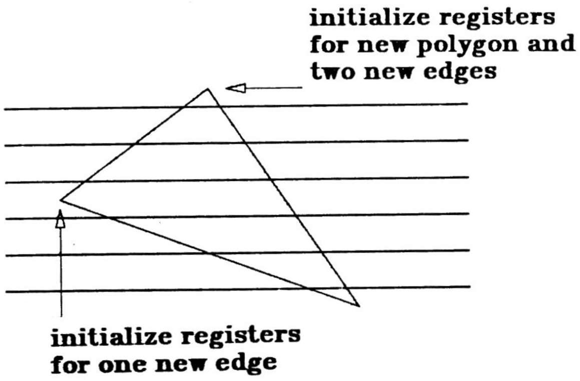


Figure 12. Initialization overhead for expanded chip.

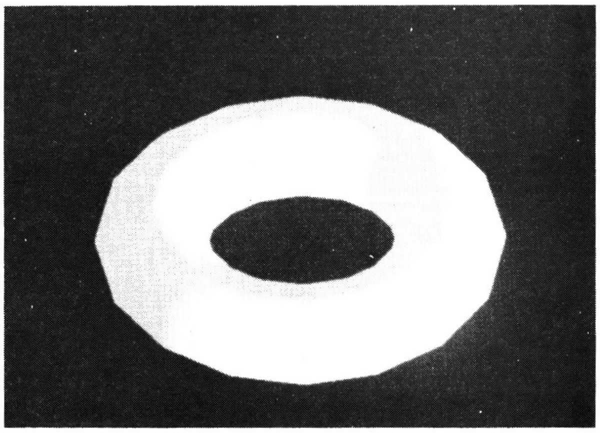


Figure 9. Torus16.

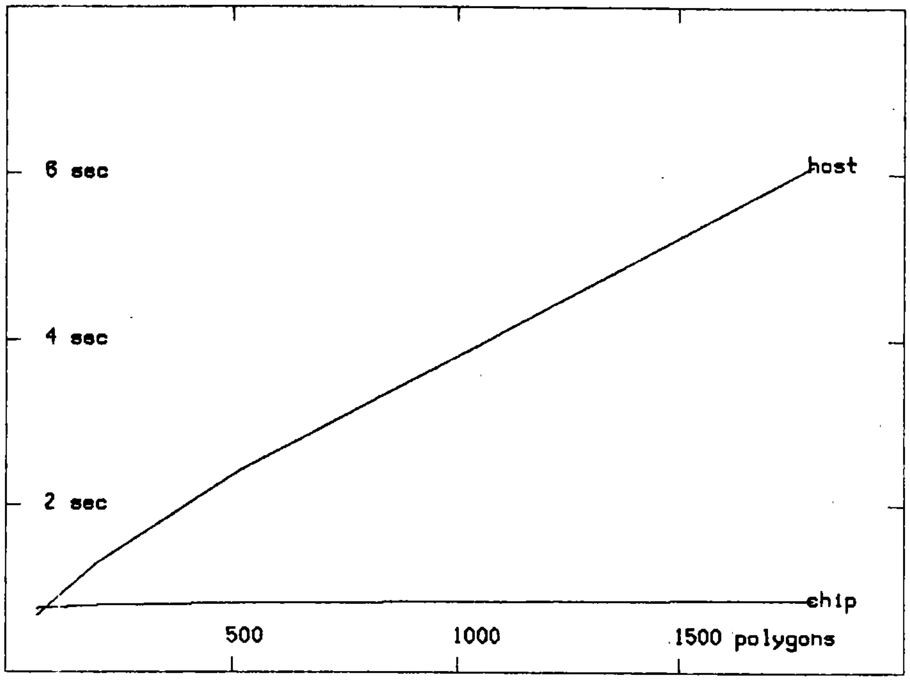


Figure 11. Relative execution times of host and current chip.

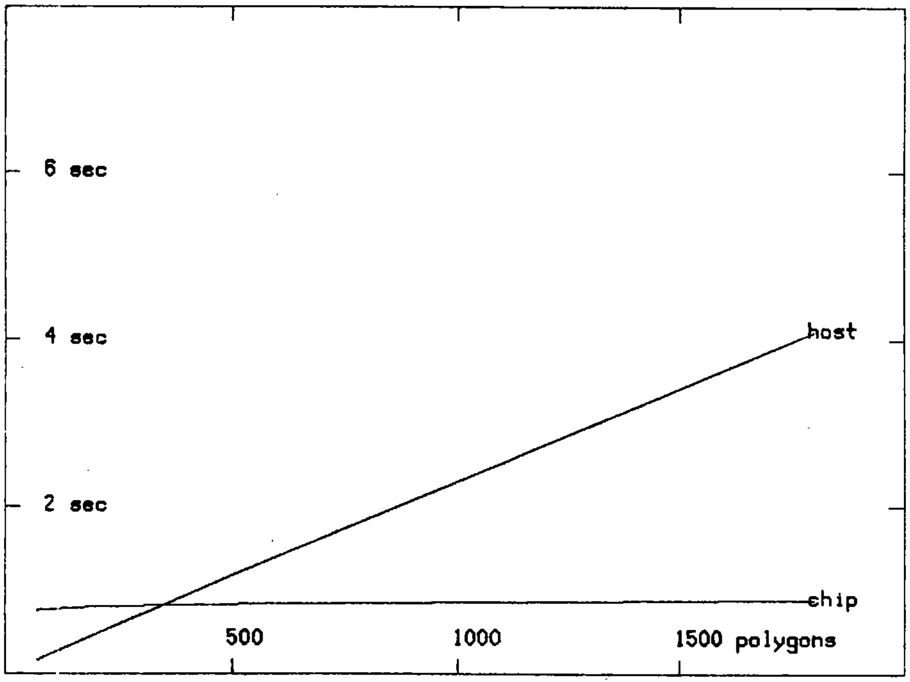


Figure 13. Relative execution times for host and expanded chip.