Report Number: WUCS-79-2

1980-01-01

A VLSI Perspective of Real-Time Hidden Surface Elimination

Authors: Gruia-Catalin Roman and Takayuki D. Kimura

VLSI technology provides and demands new ways of solving large scale computational problems. In light of this fact, a piplelined version of a real-time hidden surface elimination algorithm is proposed. The approach is tuned to the requirements of the VLSI technology: it is simple and regular, employs only local communciation, and attains a high degree of parallelism. The feasibility of the technique is demonstated for a computer graphics system where objects are defined in terms of planar triangular surface elements. A case is made in terms of the early 1980's technology.

A VLSI PERSPECTIVE OF REAL-TIME HIDDEN SURFACE ELIMINATION

Gruia-Catalin Roman

Takayuki Kimura

WUCS-79-2

January 1980

Department of Computer Science
Washington University
St. Louis, Missouri 63130

As appeared in CAD 13, No. 2, March 1981, pp. 99-107.

ABSTRACT

VLSI technology provides and demands new ways of solving large scale computational problems. In light of this fact, a pipelined version of a real-time hidden surface elimination algorithm is proposed. The approach is tuned to the requirements of the VLSI technology: it is simple and regular, employs only local communication, and attains a high degree of parallelism. The feasibility of the technique is demonstrated for a computer graphics system where objects are defined in terms of planar triangular surface elements. A case is made in terms of early 1980's technology.

Keywords: computer graphics, real-time display, hidden surface elimination, VLSI, parallel processing.

CR: 3.8, 6.2, 8.2.

1. INTRODUCTION

Spontaneous transformations (translation, scaling, and rotation) of graphic representations of three-dimensional objects require the ability to perform hidden surface elimination under real-time constraints, a highly nontrivial task. The difficulty resides in the fact that, although computationally very costly, hidden surface elimination must be accomplished in less than one thirtieth of a second, the refresh rate of a typical raster scan device (TV). In other words, for a screen consisting of 512 x 512 pixels (image elements), the average computation time per pixel must be less than 128 ns.

Despite this very severe time constraint, there are many known algorithms for solving the hidden surface elimination problem. In [11] , Sutherland et al list and compare ten of them. Every algorithm described in [11] involves a sorting process in one formor another and takes advantage of coherence properties in order to keep small the number of elements to be sorted. The particular implementation strategies selected by various designers have undoubtedly been biased by the technological limitations of the time, and it is reasonable to expect that, today, one ought to re-evaluate the algorithms proposed in the sixties and seventies from the perspective of the eighties--from the viewpoint of a VLSI implementation.

VLSI, while offering the potential for high performance through distributing the computation to the largest possible degree, requires an entirely different algorithm design philosophy, one which (1) avoids the traditional memory-processor dichotomy, (2) maximizes the homogeneity and regularity among system components and their interconnections, and (3) minimizes the communication complexity of data and control flow, rather

than the arithmetic complexity as the traditional theory of algorithm design advocates. Mass production of VLSI logic/arithmetic capabilities makes the number of operations involved in a computation a less significant factor than the complexity of making the result of an operation available to another operation.

While our ultimate goal is to establish the impact of VLSI technology in the area of computer graphics algorithms and device architectures, the scope of this paper is considerably narrower and does not purport to represent anything but a small step toward accomplishing the larger task. As such, our contribution is limited to the identification of a class of hidden surface elimination algorithms that appear to be well suited to a VLSI implementation, proposal of an appropriate general architecture for a device that might use the chosen algorithm, and demonstration of the near term feasibility of building the device based on the current perception of the early 1980's solid state technology. Similar efforts have been under way for several years in the area of special purpose high power systems such as linear time matrix processing systems [3,4], and it is the authors' hope that this paper might contribute to promoting related research in computer graphics.

A formal description of the problem we have attacked is presented in Section 2 along with our solution, a pipelined version of a familiar z-buffer type hidden surface elimination algorithm. Its fundamental capabilities and limitations are also reviewed. Sections 3 and 4 are meant to convince the reader of the near future practical feasibility of the approach. Section 3 describes a particular computer graphics system designed around the proposed architecture. It assumes objects to be defined as arbitrary

collections of planar triangular surface elements each having a uniform color (no shading of individual triangles). Section 4, by considering the technology projected for the early 1980's, demonstrates the technical feasibility of implementing the proposed system.

2. A VLSI MOTIVATED APPROACH TO HIDDEN SURFACE ELIMINATION

Before formally stating the problem we have attempted to solve, it is necessary to review very briefly the underlying world model. It is conceived as a set of objects which can individually be manipulated within the confinement of a finite space, a cube of dimensions e^3. This space is called the display environment. The objects present at any one time within the display environment form a scene. While object removal is instantaneous, the addition of new objects is accomplished through a loading process which, for a few very complex objects, may require a detectable time span, i.e., an interval longer than the refresh rate of the display screen. Objects within a scene are identified by a unique name, id, and are composed of arbitrary disjoint collections of entities called rendering elements. Examples of rendering elements may be the surface patches, polygons, or convex polyhedra.

Each rendering element has a set of display attributes, such as color and depth, associated with each point on its surface. The rendering elements are assumed to be given and determined by the needs of the intended application. The total number of rendering elements involved in a particular scene is defined as the scene complexity. This measure is justified by the fact that the complexity of the hidden surface elimination algorithm proposed in this paper is measured by the number of rendering elements involved.

Given: (1) a display screen consisting of $p \times q$ pixels;

(2) n rendering elements R_i $(1 \leq i \leq n)$;

(3) d^i_{xy}, the depth of R_i at pixel P_{xy} $(1 \leq x \leq p, 1 \leq y \leq q)$ (i.e., the minimum distance between the viewpoint and R_i when measured along the line passing through the point of coordinates (x,y) on the screen);

(4) S^i_{xy}, the display attributes of R_i at pixel P_{xy} (it includes the depth d^i_{xy});

the problem of real-time hidden surface elimination may be formulated as follows:

"Find an algorithm which

(i) 30 times a second and for an indefinite period of time, computes the sequence

$$S_{11}^{k^{11}} S_{21}^{k^{21}} \ldots S_{p1}^{k^{p1}} \ldots \ldots S_{pq}^{k^{pq}}$$

where k^{xy}, for $1 \leq x \leq p$ and $1 \leq y \leq q$, is given by

$$d_{xy}^{k^{xy}} = \min_{1 \leq i \leq n} d^i_{xy}$$

(ii) and, in addition, is amenable to a VLSI implementation, a constraint which has been outlined in the preceding section."

Our search for such an algorithm brought us to consider various schemes such as

- assigning one processor per scan line,
- assigning one processor per pixel, and

- assigning one processor to each rendering element and using a comparator network.

They all proved to be technologically and economically unfeasible. The solution that was finally selected involves the assignment of rendering elements, one per processor, along a pipeline which implements a version of a z-buffer type hidden surface elmination algorithm.

The z-buffer was first used by Newell et al [6], who noticed that if one writes in a picture buffer the images of various polygons, in the proper order, the final result is a correct hidden surface view of some given frame. The scheme was later implemented by Schumacker at G.E. by using an entire frame buffer memory. Besides its simplicity, the approach allows for a clean separation between the method used for rendering of three-dimensional objects and the hidden surface elimination process. For this particular reason, versions of the z-buffer approach have been employed in quite distinct contexts by Catmull [1], Myers [5], and by Fuchs and Johnson [2]. Our own usage of the same technique is detailed below.

The algorithm being proposed here assumes the existence of n processes, Q_i $(1 \leq i \leq n)$, one for each rendering element, forming a unidirectional pipeline. Messages flowing along the pipe (from Q_i to Q_{i+1}) contain five-tuples of the type $(x,y, D^i_{xy}, C^i_{xy}, ID^i_{xy})$ where x and y identify a pixel P_{xy} on a screen of dimensions $p \times q$, while D^i_{xy}, C^i_{xy}, and ID^i_{xy} denote the depth, color, and object name (a typical set of display attributes) associated with that pixel, so far. For each pixel P_{xy}, Q_i checks the line segment starting from the viewpoint and passing through the point (x,y) on the screen for intersection against the rendering element

controlled by Q_i . Whenever one or more intersections are detected, the depth and color (d^i_{xy} and c^i_{xy}) of the intersection closest to the viewer are computed. In all other cases, the constant ∞ is assigned as the depth of the point of intersection. While Q_i computes the depth and color of the i'th rendering element relative to pixel P_{xy} , its left neighbor, Q_{i-1}, is already considering the (i-1)'th element against the pixel following P_{xy}. After computing d^i_{xy} and c^i_{xy}, Q_i compares d^i_{xy} with $D^{i-1}_{x,y}$, which represents $\min_{k<i}\{d^k_{xy}\}$, and generates $D^i_{xy} = \min_{k\leq i}\{d^k_{xy}\}$. C^i_{xy} and ID^i_{xy} are adjusted accordingly. During the next message exchange Q_i passes $(x,y,D^i_{xy},C^i_{xy},ID^i_{xy})$ to Q_{i+1}. Thus, at the end of the pipe, each pixel's minimum depth, corresponding color, and object affiliation arrive at regular intervals and are placed in the buffer of some display processor. Consequently, the algorithm described above executes n minimization operations on n rendering elements in n time intervals, or equivalently, one minimization operation on n elements in one time interval. A formal description of the algorithm appears in Figure 1.

As stated earlier, the algorithm has been selected with the intent of being implemented on a VLSI based architecture and not on conventional machines. As such, the emphasis is on regularity, uniformity, extensive distribution of concurrent computational activities, and local communication. The algorithm is also very promising in terms of its growth characteristics regarding both changes in size and technology. Starting from the basic premise that each process Q_i is implementable on a single chip, the capacity of the system could be increased by simply adding new chips at the end of the pipe. Therefore, after the initial investment, the cost becomes

a linear function of the system complexity. Such a property would make the system very attractive to a large class of users who cannot afford this type of graphics capability today.

The proposed approach also adapts well to technological changes. First of all, the number of pins per chip stays constant with respect to the number of processes Q_i implemented on a single chip. Furthermore, since the rendering algorithm is irrelevant for the hidden surface elimination, growth may also occur through increases in the complexity of the rendering algorithm. If in the early 1980's, as shown later, it will be possible to assign a single triangle per chip, in the 1990's, complex surface generators may replace the triangle as the basic rendering element. Moreover, the modularity of the system allows future enhancement to take place also via specialization of the various processes Q_i. Features that might be considered vector display capabilities, character manipulation, etc.

Postponing the feasibility question for Sections 3 and 4, the remaining part of this section identifies several important issues and fundamental limitations regarding the implementation of the algorithm. (A process per chip distribution is assumed from now on.) The discussion is separated into three parts: issues regarding time constraints, features which ought to be incorporated into the system, and fault tolerance.

Assuming a 512 x 512 screen size and a refresh rate of 30 frames per second, the average pixel generation time is 128 ns. This represents the maximum time interval allowable for each process Qi to conclude the five-tuple generation and the message exchange. However, a 128 ns constraint

over the process Qi implies the existence of a frame buffer in the display processor. Reductions in the buffer size further limit this time interval. When using a single line buffer, for instance, the time constraint becomes 111 ns.

The time intervals given above apply only to the hidden surface elimination. However, the processes Qi ought also to allow for the manipulation (rotation, translation, scaling, change in viewpoint, etc.) of the rendering elements they control. Single object transformation commands could be issued by some host computer, and all affected processes would recognize the object id included in each command. Thus, all the elements associated with the given object would be subjected to the same transformation. The transformation would become effective two frames later, thus always allowing at least 30 ms for the processing of each command. Because the different Qi's affected may be working on different frames, the commands cannot be broadcasted; they need to be passed along the pipe in sync with the pixel processing.

While deleting an object from a scene may be done in real time, loading a new object poses special problems; depending on the object complexity, the loading may require a noticeable time delay and may result in frames which contain only partially loaded objects. To avoid such situations, one may inhibit the display of loaded elements until the receipt of an 'on' command which the host could issue when loading is completed. Another fundamental limitation of the system is related to the delay along the pipe (e.g., n = 781250 generates a 0.1s delay). Most applications envisioned today, however, could be accommodated by using less than 64k processes i.e., less than 8 ms delay.

Several issues that have not yet been resolved are shadow generation, effective anti-aliasing, and the representation of transparent surfaces. Regarding transparency, thought has been given to simulating it by forcing processes controlling transparent elements to inhibit their output periodically. Transparent surfaces would thus be rendered as random discrete dots whose density is inversely proportional to the level of transparency of the surface. (The impressionists already used this technique very successfully in their paintings.) In conjunction with anti-aliasing features built into the display processor, the technique may prove to be adequate.

Finally, there is the issue of fault tolerance. Given the anticipated large number of chips, reliability becomes an important factor. Redundant circuitry could be used to detect chip malfunctions. If enough redundancy is included, a voting scheme could be employed thus decreasing the probability of chip failure. The simplest approach to dealing with chip failures is to stop the display, set in the malfunctioning chip a failure bit designed to change the functionality of the chip to that of a simple delay along the pipe, and reload the entire scene. The faster reconfiguration of the system through reloading only the affected object is possible if the current state of the object is known. Depending upon the nature of the failure, it is also conceivable for the failing chip to issue a load command, thus transferring the element description to a chip which happens to be unassigned. In general, the strategy will depend upon technological and application constraints.

```
PROCESS Q_o./* originator of control signals */

   begin initially  x=0, y=0.
     loop
        x←(x mod p)+1.
        if x=1 then y←(y mod q)+1.
        D←depth-background(x,y).
        C←color-background(x,y).
        ID←nil.
        send(x,y,D.C,ID) to Q_1.
     end-loop.
   end Q_o.


PROCESS Q_i. /* for 1≤i≤n */

   begin
     loop
        get(x,y,D,C,ID) from Q_i-1.
        d←min.depth-element.i(x,y).
        c←color-element.i(x,y,d).
        if d<D then D←d,
                    C←c,
                    ID←i.
        if d=D then C←combine(C,c),
                    ID←nil.
        send(x,y,D,C,ID) to Q_i.
     end-loop
   end Q_i.


PROCESS Q_n+1. /* display process */

   begin
     loop
        get(x,y,D,C,ID) from Q_n.
        display(x,y,c).
        if identification-request then send(x,y,D,C,ID) to USER.
     end-loop.
   end-Q_n+1.
```

Figure 1: Formal Description of the Algorithm.

3. CASE STUDY DESCRIPTION

This section presents an instantiation of the architecture proposed above. The assumed application is computer aided design, and the system will be referred to by the name ARTEMIS. The application itself is really only a pretext. The actual intent is to describe a particular implementation of the algorithm and to later demonstrate its feasibility in Section 4.

ARTEMIS is designed to support real-time manipulation of color graphic representations of three-dimensional objects. Objects are represented as arbitrary collections of planar triangles. We chose, as a rendering element, a planar triangle because (1) the planarity reduces the incremental computation of d^i_{xy} to $d^i_{x-1,y} + K^i$, i.e., a simple addition, and (2) the storage requirement for each rendering element is uniformly identical. Each triangle is defined by three points in the object coordinate system and by its color, which is uniform over the entire surface of the triangle, on both faces. (Note: shading of individual triangles is not available.) Within a scene, individual objects can be rotated, translated, scaled, made completely transparent, identified through the use of a lightpen, or destroyed; all these operations as well as changes in the viewpoint-screen distance are performed in real time. All changes in the size, position, or orientation of a particular object must be accomplished strictly within the boundaries of the display environment. However, they may occur without regard to the presence of other objects within the display environment since object penetration is permitted. (Note: The hidden surface algorithm implemented by ARTEMIS is not affected by the presence or absence of object penetration.)

Because transformations act solely upon objects and result in changes

applied uniformly to all the triangles making up the particular object, individual triangles cannot be identified or modified by the system. However, this is only an apparent limitation because in the absence of any restrictions regarding the allowable number of triangles per object, it is conceivable for the user to specify single triangles as separate objects. Since the complexity of an object is measured by the number of rendering elements that make it up and the complexity of a scene is defined as the sum of the complexities of the component objects, the complexity of a scene is not affected by the way the triangles are distributed among objects.

The video image generated on the screen of the raster scan device corresponds to the view seen by an observer at distance 1/v from the display environment and looking through a viewing window or screen of size $p \times q$ located at the center of some face of the display environment cube. (See Figure 2a.) The viewpoint is restricted to the normal to the center of the viewing window, but the distance 1/v may change in real time. The orthonormal view is obtainable by specifying $v = 0$.

An additional real-time feature supported by ARTEMIS is the so-called "transparency pyramid." This pyramid is defined by the viewpoint, a window (called "cutting window") of size $w_x \leq p$ by $w_y \leq q$, and a depth w_z . The transparency pyramid, whose function is illustrated in Figure 2b, provides a mechanism for examining surfaces normally hidden from the observer by making invisible any surface which intersects the transparency pyramid at a depth less than the value w_z .

The design of ARTEMIS assumes the following principal system parameters:

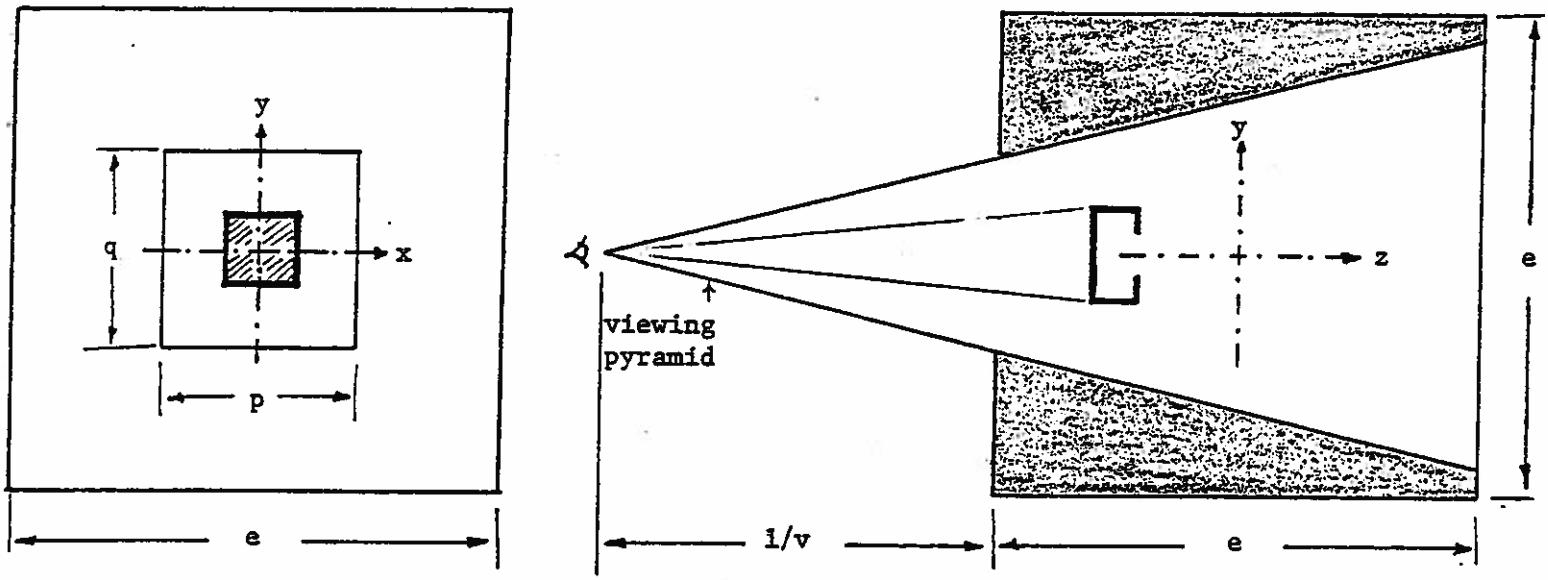
maximum scene complexity (generates 8 ms delay)	$n = 2^{16}-1$
maximum number of objects within a scene (each triangle may be a separate object)	$2^{16}-1$ (i.e., $0<id<2^{16}$)
size of the display environment cube (however, all computations are done using 32 bits)	$e = 2^{16}$
size of the viewing window (same as the screen size)	$p \times q = 512^2$
number of bits used to specify the color (red, blue, green)	$3 \times 8 = 24$
range of allowable depth values (note: $(2^{15}-1)$ is used to represent ∞).	$[-2^{15}, 2^{15}-1]$

Figure 3 depicts schematically the overall architecture of the proposed system: a host computer with assorted input/output and storage devices, a digitized image processor (DIP) storing a background scenery, the real-time unit (RT) which performs the hidden surface elimination, a display processor (DP), and a controller which acts as an intermediary between the host and the DP and also as a diagnostic analyzer for the RT. All system activities are controlled by the host which issues specific commands to all other components. A sample repertoire is given in Table 1.

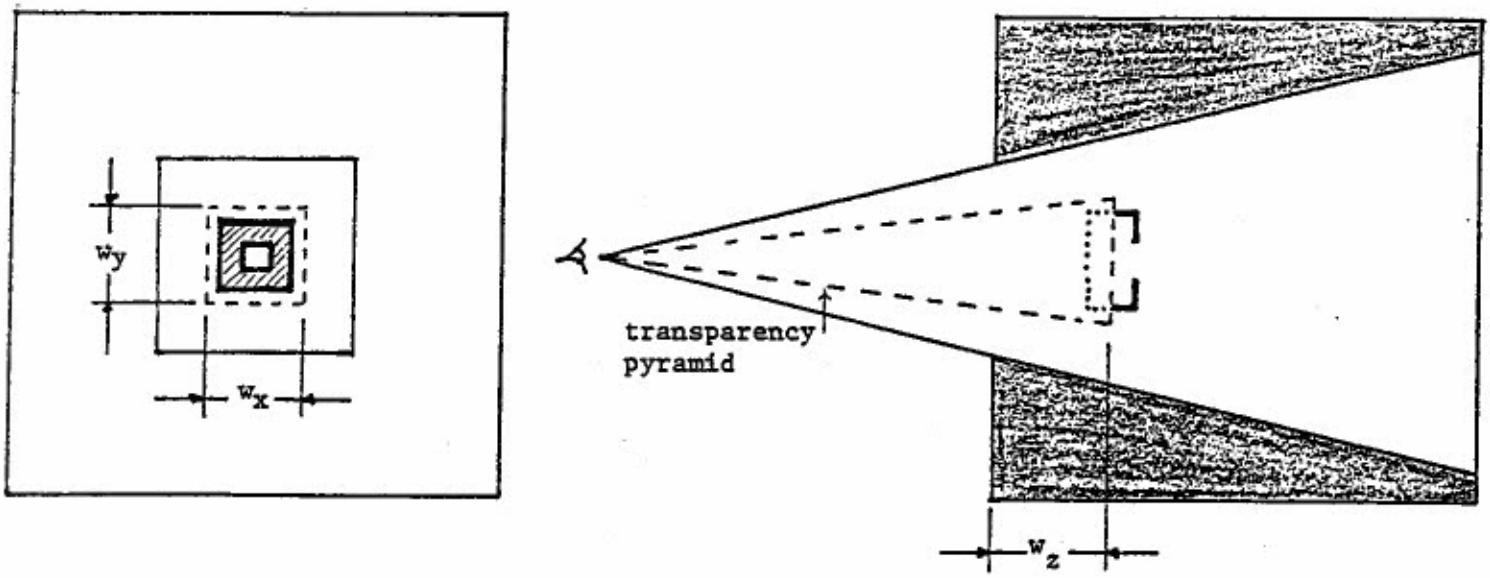
A typical ARTEMIS work session is envisioned as starting by clearing the system. Next, the application program issues a series of load commands defining a scene to be manipulated. Once an object is loaded, it continues to be part of the scene unless it is explicitly deleted. A background

scenery is also built in the form of a digitized image stored in the DIP. For each pixel, the color and depth of that background point are specified. The display process is activated by the START command, but changes in the background and loading of new objects may resume at any point. During the display process, the application program may request object transformations, redefinition of the transparency pyramid, and perspective alterations by directing the proper commands to the RT unit. All changes become effective two frames after the one during which they are received by RT. Furthermore, the application program has access to the coordinates of any pixel to which the lightpen points, and they can be used to request from DP the object id, depth value, and color pertinent to that particular pixel. The current state of a given object is made available to the application program when a STORE command is issued. It results in passing the current object description to the host via the controller.

The ability to implement the RT unit of this particular system is argued in the following section in terms of gate requirements and achievable execution speed relative to current projections for early 1980's technology.



(a)



(b)

Figure 2: World Model.

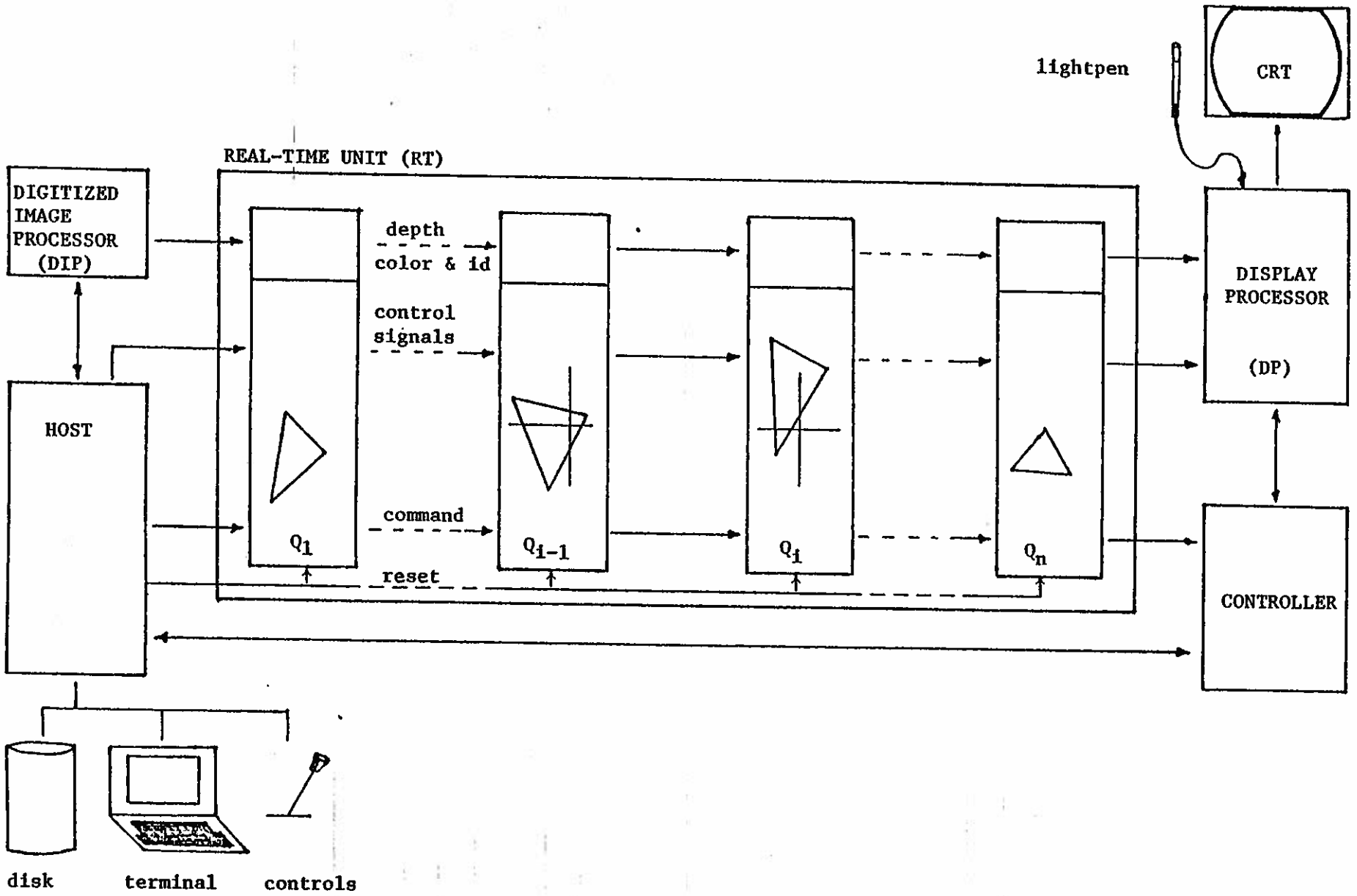


Figure 3: System Architecture.

TABLE 1

INTERFACE LANGUAGE DESCRIPTION

A. Object oriented commands

CLEAR		- clear the entire system
LOAD	id, triangle	- load one triangle as part of the object 'id'
OFF	id	- stop displaying the object
ON	id	- start displaying the object again
KILL	id	- destroy the object
STORE	id	- pass to the host computer the current description of the specified object
TRANS	id, ΔT	- translate by ΔT (3 item vector)
ROT	id, ΔM	- rotate/scale by ΔM (3 by 3 matrix)
XROT	id, ΔM, ΔT	- rotate with respect to translated axes
VIEW	v	- set the perspective parameter
CUT	w_x, w_y, w_z	- define the transparency pyramid

B. Display processor commands

CLEAR		- same as above
START		- start the display process
LIGHT		- get lightpen coordinates
GET	x, y	- get all information pertinent to pixel (x,y)
ZMOD	k	- set depth modulation index
HTP	k	- set degree of horizontal interpolation
MHP		- define uniform color transformation rules

C. Digitized image processor commands

SET	x, y, z, c	- set the depth and the color of pixel (x,y)
READ	x, y	- read the depth and the color of pixel (x,y)
CLR	c	- clear the processor and attach to each pixel the color c and maximum depth

4. FEASIBILITY

The RT unit introduced in Section 3 represents an instantiation of the advocated hidden surface elmininationalgorithm. Within RT, the processes Q_i materialize as a pipeline of 64k identical Q units, each representing a VLSI chip. The economical and technical feasibility of the RT unit depends on the ability of each Q unit to compute d^i_{xy} in less then 128 ns. This section evaluates the computational complexity of the Q unit employed by ARTEMIS and demonstrates that a 20,000 gate chip using less that 64 pins suffices under the following technological assumptions about solid state circuitry in the early 1980's [8]:

- Intra-chip gate delay time can be made less than 3 ns.
- Inter-chip transmission delay can be made less than 20 ns.

In addition, it is also assumed that:

- There exist fast algorithms for addition and comparison such that for a 32-bit word size, an adder requires 400 gates and 15 gate delays and a comparator 200 gates and 5 gate delays [9].
- A frame buffer is present in the display processor, thus imposing an average response time of 128 ns per pixel.
- Fault detection occurs visually. (No redundancy is included in the chip. The reader may easily estimate the impact of potential redundancy on the number of gates required.)
- Fault location is determined by running appropriate diagnostic software on the host.
- Fault correction requires chip substitution.

In order to process pixels at the rate of 128 ns each, the Q unit itself is pipelined as indicated in Figure 4. The command subunit C

handles the command traffic, the loading and unloading of triangles, the recognition of pertinent commands, and the passage of appropriate information to all other subunits. When a series of load commands containing triangle descriptions are sent to the Q units, the C component of the first empty Q unit loads the triangle and removes the command from the pipe by replacing it with an acknowledgment message which is returned to the host via the controller. All other commands are simply passed from one C subunit to the next along the pipeline. Within each Q unit, the triangle is actually maintained by the transformation subunit T. Rotations, translation, scaling, and perspective transformation are carried out in T. The triangle, after transformation, is made available to the constants generator K. The subunits C, T, and K represent the slow half of the Q unit since the output from K is needed only at the beginning of each new frame, i.e., every 30 ms.

The fast half of the Q unit operates at the 128 ns rate, and the functionality of its components is given below:

(Note: Assume that Q_i controls some triangle t in the plane s and whose sides are ℓ_1, ℓ_2, and ℓ_3. Since t is considered after the perspective transformation, the line segment L_{xy} corresponding to pixel P_{xy} is parallel to the Z-axis).

- D computes z^s_{xy} as the depth of the intersection between the plane s and L_{xy}; if the plane happens to be parallel to the Z-axis, z^s_{xy} is set to ∞.
- Lk(k = 1,2,3) checks on which side of ℓ_k L_{xy} intersects the triangle's plane. This information, α^k_{xy} taking the values

-1, 0, or +1 , is passed to B for use in determining if the intersection with the triangle's plane is inside or outside the triangle. L_{xy} falls inside the triangle if $\alpha^1_{xy} = \alpha^2_{xy} = \alpha^3_{xy}$. Lk also computes z^k_{xy} as the depth of the intersection of ℓ_k and L_{xy} whenever L_{xy} moves from one side of ℓ_k to the other. In all other cases, $z^k_{xy} = \infty$. The value z^k_{xy} is needed in order to handle the special case when the triangle's plane is parallel to the Z-axis.

- B determines if L_{xy} intersects s inside the triangle t by comparing the signs of α^k_{xy}(k=1,2,3); if this is not the case,the output z^t_{xy} is set to ∞ , otherwise z^s_{xy} is passed on to M .
- M selects the closest distance, $z_{xy} = \min(z^s_{xy}, z^1_{xy}, z^2_{xy}, z^3_{xy})$, between the triangle and the screen (along the direction L_{xy}) .
- I inhibits (sets to ∞) the depth value if the particular point of intersection (x,y,z_{xy}) falls within the transparency pyramid, i.e., $|x| \leq w_x$, $|y| \leq w_y$, and $z_{xy} \leq w_z$.
- S performs the comparison with the depth furnished by the unit Q_{i-1} and passes on to Q_{i+1} the smallest depth detected so far along with the id and color associated with the corresponding triangle.

The remainder of this section outlines the evaluation of the Q-unit complexity. The analysis is presented in two parts: a short discussion of the slow half (subunits C, T,and K) and a detailed look at the fast half (subunits S, L1, L2, L3, B, M, I,and S). The slow half of the Q unit can easily be implemented on a typical microprocessor (e.g., Z8000 or M68000) augmented by a fast 32 bit multiplier, occupying half of the chip

(about 10,000 gates) and requiring only 16 pins (8 inputs, 8 outputs) for the command traffic. Its response time constraint is 30 ms, and its memory is limited to storing the endpoints of the triangle before the perspective transformation, id, color, transformation matrix to be multiplied with the current coordinates, several parameters, flags, and constants to be passed to the fast half of the Q unit. Furthermore, the functions performed by the slow half include only command decoding, parameter and flag setting, one generalized 3D transformation, one perspective transformation, and constants generation. Actually, the generation of the constants by the K subunit is the most involved computation carried by the slow half. The mathematical details are given below before discussing the fast half whose performance is determined by the availability of the right constants.

Mathematical Preliminaries

Let $\begin{vmatrix} P_1 \\ P_2 \\ P_3 \end{vmatrix} = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$ be the coordinates of the triangle t after perspective transformation, and let (x_o, y_o) be the coordinates of the left upper corner of the screen, the coordinates of the first pixel of the first scan line on the screen. Let

$$A = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

$$A^z = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad A^x = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad A^y = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

and

A^d_{ij} = the cofactor of the ij-th element of A^d, where $d \in \{x,y,z\}$, and $1 \leq i, j \leq 3$.

For example:

$$A^x_{11} = \begin{vmatrix} y_2 & z_2 \\ y_3 & z_3 \end{vmatrix} \qquad A^x_{12} = -\begin{vmatrix} 1 & z_2 \\ 1 & z_3 \end{vmatrix} \qquad A^x_{13} = \begin{vmatrix} 1 & y_2 \\ 1 & y_3 \end{vmatrix}$$

$$A^y_{11} = \begin{vmatrix} 1 & z_2 \\ 1 & z_3 \end{vmatrix} \qquad A^y_{12} = -\begin{vmatrix} x_2 & z_2 \\ x_3 & z_3 \end{vmatrix} \qquad A^y_{13} = \begin{vmatrix} x_2 & 1 \\ x_3 & 1 \end{vmatrix}$$

$$A^z_{11} = \begin{vmatrix} y_2 & 1 \\ y_3 & 1 \end{vmatrix} \qquad A^z_{12} = -\begin{vmatrix} x_2 & 1 \\ x_3 & 1 \end{vmatrix} \qquad A^z_{13} = \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix}$$

Let:

$$f^x_k(y,z) = A^x_{k1} + A^x_{k2}y + A^x_{k3}\, z,$$

$$f^y_k(x,z) = A^y_{k1}\, x + A^y_{k2} + A^y_{k3}\, z,$$

$$f^z_k(x,y) = A^z_{k1}\, x + A^z_{k2}\, y + A^z_{k3} .$$

where $1 \leq k \leq 3$.

Then, the following facts are relevant for the K subunit:

(1) $A^x x + A\, y + A^z z = A$ is the equation of the plane s on which the triangle t exists. (Figure 4.)

(2) $f^x_k(y,z) = f^y_k(x,z) = f^z_k(x,y) = 0$, $1 \leq k \leq 3$, is the equation of the line on which the line segment ℓ_k exists.

(3) $f^d_k = 0$, $1 \leq k \leq 3$, $d \in \{x,y,z\}$, is the equation of the projection on the d-plane of the line on which ℓ_k exists.

(4) $f^d_k(u,v)$ have the same sign for all $1 \leq k \leq 3$ if the point (u,v) is <u>inside</u> the projection of the triangle t on the d-plane, where $d \in \{x,y,z\}$.

(5) (u,v,w) is on the triangle surface t if $A^x u + A^y v + A^z w = A$ and for all $d \in \{x,y,z\}$ f_1^d, f_2^d, and f_3^d have the same sign.

The K subunit.

Using the above determinants, A, A^d, A_{ij}^d, $d \in \{x,y,z\}$, $1 \leq i, j \leq 3$, the K subunit generates constants for D, L1, L2, and L3 subunits.

(1) Constants for D: $(z_o, \Delta_x, \Delta_y)$

If $A^z = 0$ then $z_o = \Delta_x = \Delta_y = \infty$

else $z_o = (A - A^x x_o - A^y y_o)/A^z$ (the depth of plane s at pixel $P_{x_o y_o}$)

$\Delta_x = -A^x/A^z$ (depth change for unit increment in x)

$\Delta_y = A^y/A^z$ (depth change for unit increment in y)

Rationale: For a plane s which is not parallel to the Z-axis (i.e., $A^z \neq 0$), the D-unit will be able to compute the depth corresponding to any pixel on the screen given the depth at pixel (x_0, y_0) and the partial derivatives of the depth with respect to x and y, Δx and Δy:

$$z(x,y) = z_o + \Delta x(x - x_o) - \Delta y(y - y_o).$$

Furthermore, since during the generation of any image on the screen x and y are subject to unit changes, the D subunit will need to perform only additions, an important factor in achieving the required performance, i.e., real-time constraint.

(2) Constants for Lk where $1 \leq k \leq 3$: $(\delta_x, \delta'_x, \delta_y, \delta'_y, \alpha_o, \alpha_x, \alpha_y, \beta_x, \beta_y, z_o, \Delta)$

$$\begin{cases} \delta_x = x_o - x_{k+1} & \delta'_x = x_o - x_{k+2} \quad \text{where} \quad x_4 \equiv x_1 \\ \delta_y = y_{k+1} - y_o & \delta'_y = y_{k+2} - y_o \qquad\quad y_4 \equiv y_1 \\ & \qquad\qquad\quad z_4 \equiv z_1 \end{cases}$$

$$\begin{cases} \alpha_o + A^z_{k1}x_o + A^z_{k2}y_o + A^z_{k3} & \text{(line equation evaluated for } (x_o, y_o)) \\ \alpha_x = A^z_{k1} & \text{(change in } \alpha_o \text{ for unit increment in x)} \\ \alpha_y = -A^z_{k2} & \text{(change in } \alpha_o \text{ for unit decrement in y)} \end{cases}$$

$$\begin{cases} \beta_x = \text{if } A^x_{k3} = 0 \text{ then 1 else 0} & (\beta_x = 1 \text{ indicates that the line } \ell_k \text{ is parallel to } y = 0 \text{ plane}) \\ \beta_y = \text{if } A^y_{k3} = 0 \text{ then 1 else 0} & (\beta_y = 1 \text{ indicates that the line } \ell_k \text{ is parallel to } x = 0 \text{ plane}) \end{cases}$$

$$\begin{cases} \text{If } A^x_{k3} = A^y_{k3} = 0 \text{ then } z_o = \min\{z_{k+1}, z_{k+2}\}. & (\text{line } \ell_k \text{ is parallel to Z-axis}) \\ \text{If } A^x_{k3} \neq 0 \text{ then } (z_o = -(A^x_{k1} + A^x_{k2}y_o)/A^x_{k3},\ \Delta = A^x_{k2}/A^x_{k3}) & (\text{line } \ell_k \text{ is not parallel to X-axis}) \\ \text{If } A^y_{k3} \neq 0 \text{ then } (z_o = -(A^y_{k1}x_o + A^y_{k2})/A^y_{k3},\ \Delta = -A^y_{k1}/A^y_{k3}) & (\text{line } \ell_k \text{ is not parallel to Y-axis}) \end{cases}$$

Rationale: The constants above help the Lk subunits determine the traversal of the line ℓ_k and the depth of the crossover by employing only additions and comparisons. The sign of the line equation for a given pixel is given by

$$\alpha(x,y) = \alpha_o + (x - x_o)\alpha_x - (y - y_o)\alpha_y$$

which may be evaluated incrementally by Lk through the use of addition only. Similarly, the depths of the consecutive crossover points will also

be determined by single additions per unit increment in x or unit decrement in y, depending upon the orientation of the line ℓ_k.

Next, let us consider the complexity of the fast half in terms of the number of registers (most of them 32 bits long) and the number of gate delays involved in one pixel computation. Denote by U and V the pixel synchronization signals moving the pixel from (x,y) to $(x+1,y)$ and to $(x_o,y-1)$, respectively.

D subunit

The D subunit computes the depth of the surface s corresponding to the pixel P_{xy}. D consists of one adder and four registers: r_1, r_2, r_3, r_4. The initial contents of the registers are given by the K subunit:

$$\left.\begin{aligned} r_1 &= z_o = (A - A^x x_o - A^y y_o)/A^z \\ r_2 &= z_o \\ r_3 &= \Delta_x = -A^x/A^z \\ r_4 &= \Delta_y = -A^y/A^z \end{aligned}\right\} \quad \text{if } A^z \neq 0$$

For each U signal, $r_1 = r_1 + r_3$

For each V signal, $r_2 = r_2 + r_4$; $r_1 = r_2$.

Note that the contents of r_3 and r_4 remain unchanged.

After $(y_o - y)$ V signals followed by $(x - x_o)$ U signals, the contents of registers r_1 and r_2 will be

$$r_1 = r_2 - (A^x/A^z)(x - x_o) = (A - A^x x - A^y y)/A^z$$

$$r_2 = (A - A^x x_o - A^y y_o)/A^z + (A^x/A^z)(y_o - y) = (A - A^x x_o - A^y y)/A^z$$

Because $A^x x + A^y y + A^z r_1 = A$, r_1 must be the depth of the plane s defined by t at the pixel P_{xy}; i.e., $r_1 = z^s_{xy}$.

Lk subunit.

The role of the Lk subunit is to determine when the scanning crosses the line ℓ_k and at what depth. Lk consists of 7 registers r_1 to r_7, 4 counters r_8 to r_{11}, 2 adders, and 5 zero-testers. The registers contain initially:

$$r_1 = r_2 = \alpha_o = A^z_{k1}x_o + A^z_{k2}y_o + A^z_{k3}$$

$$r_3 = \alpha_x = A^z_{k1}$$

$$r_4 = \alpha_y = -A^z_{k2}$$

$$r_5 = r_6 = z_o = \begin{cases} -(A^x_{k1} + A^x_{k2}y_o)/A^x_{k3} & \text{if } A^x_{k3} \neq 0 \\ -(A^y_{k1}x_o + A^y_{k2})/A^y_{k3} & \text{if } A^y_{k3} \neq 0 \end{cases}$$

$$r_7 = \Delta = \begin{cases} A^x_{k2}/A^x_{k3} & \text{if } A^x_{k3} \neq 0 \\ A^y_{k2}/A^y_{k3} & \text{if } A^y_{k3} \neq 0 \end{cases}$$

$$r_8 = \delta_x = x_o - x_{k+1} \qquad r_9 = \delta'_x = x_o - x_{k+2}$$

$$r_{10} = \delta_y = y_{k+1} - y_o \qquad r_{11} = \delta'_y = y_{k+2} - y_o$$

The contents of these registers will be modified upon receiving either a U or V signal, as follows:

(1) For each U signal:

$$r_1 := r_1 + r_3$$

$$r_5 := r_7 + r_5 \qquad \text{if } \beta_y = 1$$

$r_8 := r_8 + 1$

$r_9 := r_9 + 1$

(2) For each V signal:

$r_1 := r_2 := r_2 + r_4$

$r_5 := r_5 + r_7$ if $\beta_x = 1$

$r_5 := r_6$ if $\beta_y = 1$

$r_{10} := r_{10} + 1$

$r_{11} := r_{11} + 1$

Note that the contents of the other registers remain unchanged.

By a similar argument to the one used above for D, one can show that after $(y_o - y)$ V signals followed by $(x - x_o)$ U signals, the contents of the registers become:

$$r_1 = A^z_{k1}x + A^z_{k2}y + A^z_{k3}$$

$$r_5 = \begin{cases} -(A^x_{k1} - A^x_{k2}y)/A^x_{k3} & \text{if } A^x_{k3} \neq 0, \\ -(A^y_{k1}x - A^y_{k2})/A^y_{k3} & \text{if } A^y_{k3} \neq 0, \end{cases}$$

$$r_8 = x - x_{k+1}$$

$$r_9 = x - x_{k+2}$$

$$r_{10} = y_{k+1} - y$$

$$r_{11} = y_{k+2} - y$$

Therefore, r_5 gives the depth z^k_{xy} of the line ℓ_k at the pixel (x,y),

when $r_1 = 0$ or changed sign while r_8 and r_9, and also r_{10} and r_{11} have opposite signs, respectively. α^k_{xy} equals sign(r_1), and it is sent to the B subunit.

Subunits B,M,I, and S.

The B subunit involves only gating and no computation, for it compares three signs: α^1_{xy}, α^2_{xy} and α^3_{xy}. The minimization subunit M requires 3 comparators. The I subunit consists of 4 registers, 4 zero-testers, and 1 comparator. The S subunit has one comparator.

A summary of register requirement in the fast half is given in Table 2 below.

	Registers	Adders	Comparators (Zero-testers)
D	4	1	0
B	0	0	0
L1	11	2	5
L2	11	2	5
L3	11	2	5
M	0	0	3
I	4	0	4
S	2	0	1
	43	7	23

TABLE 2

Register Requirements

Since the computations in D, L1, L2, and L3 are carried out in parallel, their delay is one addition plus one comparison time, assuming the same delay time for the zero-tester as for the comparator. The minimization of 4 elements incurs a two-comparison delay time. S and I require a one-comparison delay time each. Altogether, the response time for the fast half equals, at most, one addition plus five comparisons, or in terms of the number of gate delays, 40 gate delays (counting 15 for the addition and 5 for each comparison).

This corresponds to 120 ns, assuming 3 ns for one gate delay.

Finally, note that the output of S consists of the depth (16 bits), color (24 bits), and id (16 bits), i.e., a total of 56 bits, and is generated every 128 ns. For a 20 ns inter-chip delay time, transmission of 64 bits takes 4 x 20 = 80 ns with 4 transmissions of 16 bits each overlapping the 120 ns computation time. Consequently, the number of pins for the entire Q unit, excluding power and synchronization signals, is 48.

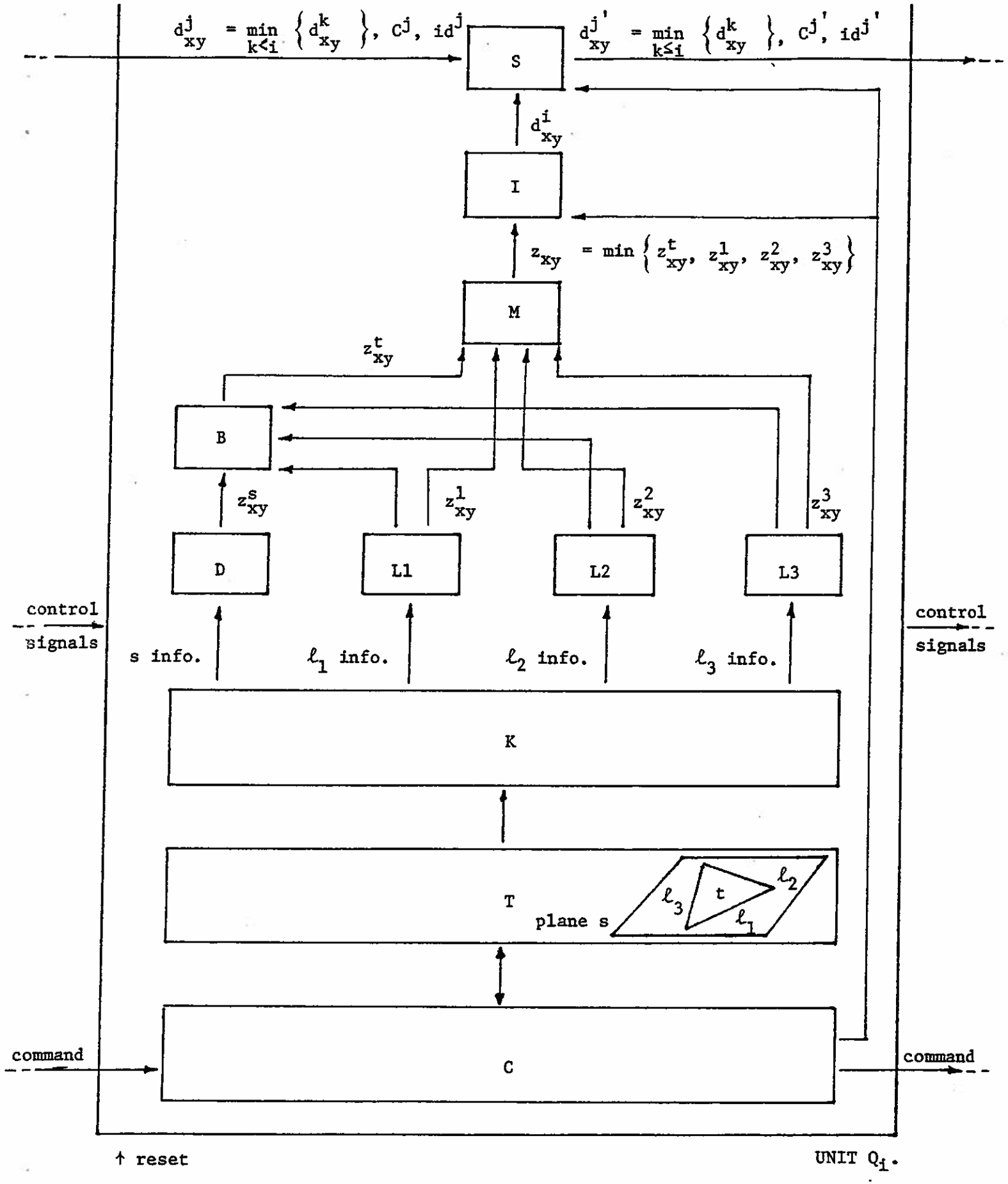


Figure 4: Q Unit Structure.

5. CONCLUSION

The new VLSI technology provides and demands a new way of solving large scale computational problems. A system design based on the VLSI technology requires a quite different set of design criteria and a different perspective on the problem at hand. For example, design and analysis of algorithms for VLSI architecture must evaluate more of the communication complexity than the arithmetic complexity, and must exploit as much parallelism as possible.

The problem of real-time hidden surface elimination has traditionally been a very difficult problem because of its large computational complexity and severe real-time constraint. How hard is it with the VLSI technology? We have tried to answer this question by studying various possibilities of VLSI architectures for the problem, and in this paper we described the most reasonable and the simplest one. Some of the other schemes we considered are as follows:

(1) assigning one processor to each scan line,

(2) assigning one processor to each pixel point,

(3) assigning one processor to each rendering element, and using a comparator network for computing the minimum distance.

In all these cases, the switching complexity, i.e., data-flow control, is too much to be overcome with the current VLSI technology, economically or technologically.

As a by-product of our study, the algorithm and the architecture described in this paper may serve as a prototype VLSI architecture for the hidden surface elimination problem, to be implemented and tested. One very

important issue in actual implementation is that of fault tolerance. However, this issue is not unique to the hidden surface elimination problem, but universal to any VLSI architecture design. Therefore, we propose to study the issue in a wider context separately.

Bibliographical List

[1] Catmull, E., "Computer Display of Curved Surfaces." Ph.D. thesis, University of Utah, 1974.

[2] Fuchs, H. and Johnson, B. W., "A Multiprocessor Architecture for Video Graphics," Proceedings of the International Symposium on Computer Architecture, Philadelphia, PA, 1979.

[3] Kant, M. and Kimura, T., "Decentralized Parallel Algorithms for Matrix Computation." Proceedings of the Fifth Annual Symposium on Computer Architecture, Palo Alto, California, April 1978, pp. 96-100.

[4] Kung, H. T. and Leiserson, C. E., "Algorithms for VLSI Processor Arrays." section 8.3 of Introduction to VLSI Systems, by Mead, C. and Conway, L., Addison-Wesley Pub. Co., 1980.

[5] Myers, A. J., "An Efficient Visible Surface Algorithm." Report to N.S.F., Computer Graphics Research Group, Ohio State University, 1975.

[6] Newell, M. E., Newell, R. G. and Sancha, T. L., "A New Approach to the Shaded Picture Problem." Proceedings of the A.C.M. National Conference. 1972.

[7] Newman, W. M. and Sproull, R. F. Principles of Interactive Computer Graphics, McGraw-Hill, 1973.

[8] Noyce, R. N., "Microelectronics." Scientific American, No. 3, September 1971.

[9] Savage, J. E., The Complexity of Computing, John Wiley, 1976.

[10] Sutherland, I. E. and Hodgman, G. W., "Reentrant Polygon Clipping." CACM 17, No. 1, January 1974.

[11] Sutherland, I. E., Sproull, R. F. and Schumacker, R. F., "A Characterization of Ten Hidden-Surface Algorithms." Computing Surveys 6, No. 1, January 1974.