The Geometry Engine: A VLSI Geometry System for Graphics

by

James H. Clark

Computer Systems Laboratory
Stanford University
and
Silicon Graphics, Inc.
Palo Alto, California

Abstract

The *Geometry Engine* [1] is a special-purpose VLSI processor for computer graphics. It is a four-component vector, floating-point processor for accomplishing three basic operations in computer graphics: matrix transformations, clipping and mapping to output device coordinates. This paper desribes the Geometry Engine and the Geometric Graphics System it composes. It presents the instruction set of the system, its design motivations and the Geometry System architecture.

Keywords: VLSI, Geometric processing, real-time graphics, arithmetic processing

CR Categories: 3.3, 3.4, 3.7

Geometry System Overview

The *Geometry System* is a floating-point, geometric computing system for computer graphics constructed from a basic building block, the *Geometry Engine*. Twelve copies of the Geometry Engine arranged in a pipeline compose the complete system in its most general form. In its present form, the Geometry Engine occupies a single, 40-pin IC package.

The notable characteristics of the system are:

- **General Instruction Set** - It executes a very general 2D and 3D instruction set of utility in all engineering graphics applications. This instruction set includes operations for matrix transformations, windowing (clipping), perspective and orthographic projections, stereo pair production and arbitrary output device coordinate scaling.
- **Curve Generation** - The system will generate quadratic and cubic curves and all of the conic sections, i.e. circles, parabolas, hyperbolas, etc.
- **Device Independent** - The system is independent of the output device used and works equally well in either vector-based or raster-based systems. It allows color or black and white polygons, lines and characters.
- **Flexible Input Format** - The system accepts input coordinates in either integer or floating point format.
- **High Performance Floating Point** - Its effective computation rate is equivalent to 5 million floating-point operations per second, corresponding to a fully transformed, clipped, scaled coordinate each 15 microseconds.
- **Reconfigurable** - Each Geometry Engine is "softly" configured; that is, one device with a single configuration register serves in twelve different capacities.
- **Selection/Hit-Testing Mechanism** - The Geometry Engine has a "hit-testing" mechanism to assist in "pointing" functions, such as are required for a fast, interactive graphics system with a tablet, mouse or other input devices.
- **Scales to a Single Chip** - The system can be put in a smaller number of IC packages as soon as the technology for fabrication reduces the size of the Geometry Engine design. Ultimately, the entire 12 Engine system will fit on in one IC package, be a factor of 4 faster and be correspondingly reduced in cost.

The Geometry Engine is a four-component vector function unit whose architecture is best illustrated by the chip photograph shown in Figures 1 and 2. Each of the four function units along the bottom two-thirds of the photo consists of two copies of a computing unit, a mantissa and characteristic. The chip also has an internal clock generator, at the top left corner, and a microprogram counter with push-down subroutine stack, shown at the top right. The upper third of the chip is the control store, which holds the equivalent of 40k bits of control store. This control store contains all of the microcode that implements the instructions and floating-point computations described below.

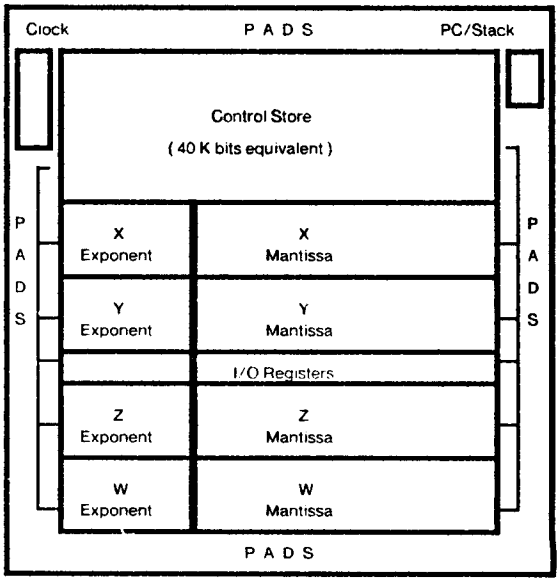

Figure 1: A Block Diagram of the Geometry Engine corresponding to the photo in Figure 2.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1982 ACM 0-89791-076-1/82/007/0127 $00.75

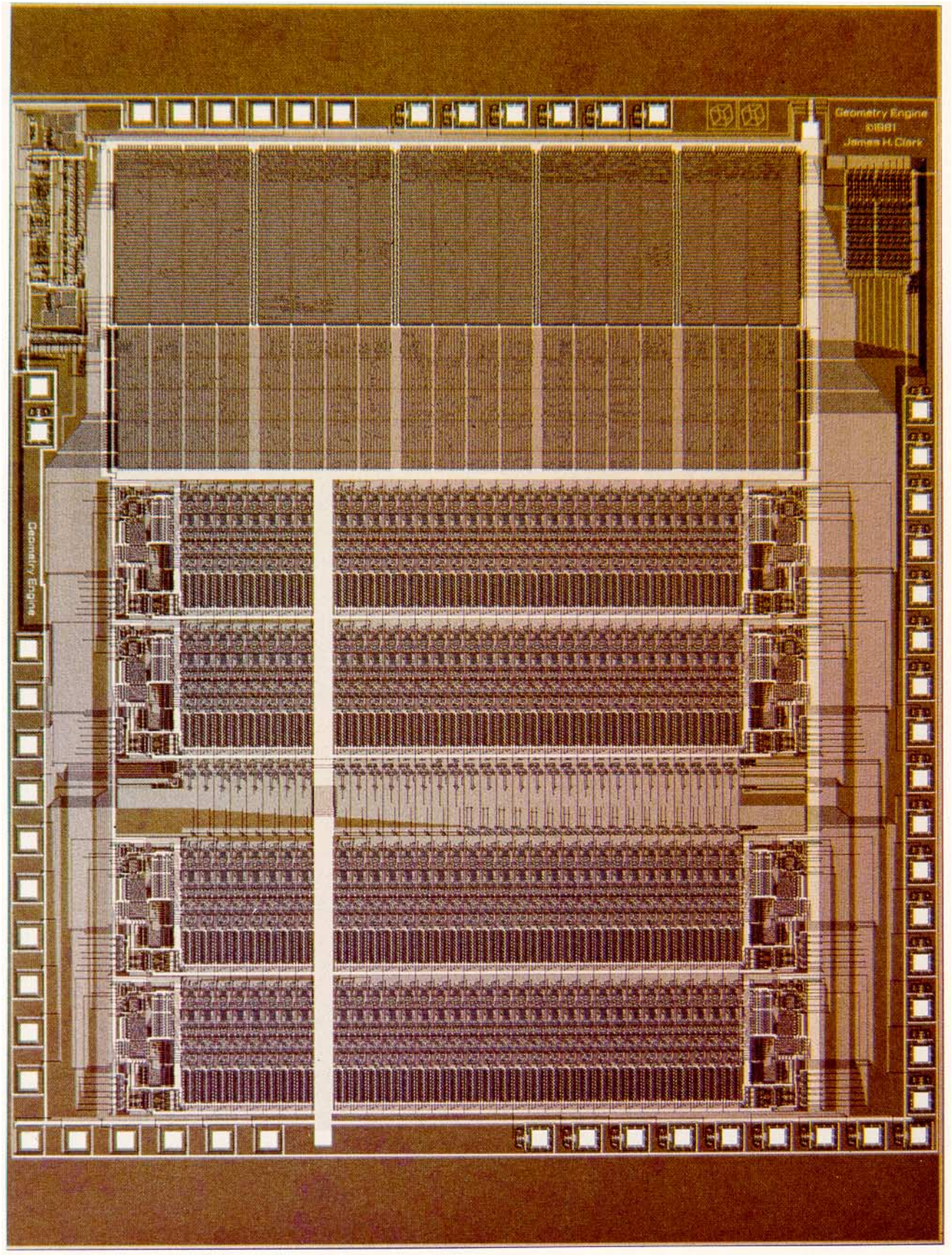

Figure 2: Photograph of the Geometry Engine.

Geometry System Functions

The Geometry System [2] is designed for high-performance, low-cost, floating-point geometric computation in computer graphics applications. It is composed of three subsystems, each of which is composed of Geometry Engines. These subsystems are illustrated in Figure 3. The particular position of a Geometry Engine in the pipeline determines its particuar function in the whole system. Each Engine has a configuration register that is loaded when the system is powered on, after a Reset command is issued. Until the system is reset again, the Engine behaves according to the configuration code.

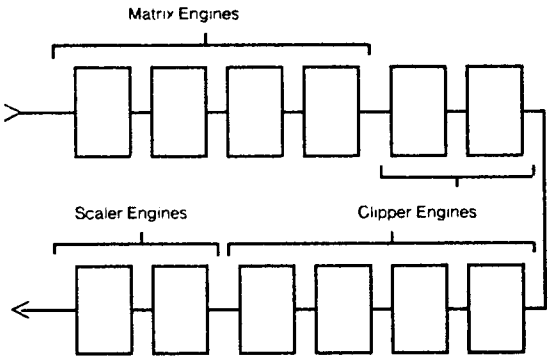

Figure 3: Geometry System; each block is a Geometry Engine.

The subsystems are:

- **Matrix Subsystem** - A stack of 4x4 floating-point matrices for completely general, 2D or 3D floating-point coordinate transformation of graphical data.
- **Clipping Subsystem** - A windowing, or clipping, capability for clipping 2D or 3D graphical data to a window into the user's virtual drawing space. In 3D, this window is a volume of the user's virtual, floating-point space, corresponding to a truncated viewing pyramid with "near" and "far" clipping.
- **Scaling Subsystem** - Scaling of 2D and 3D coordinates to the coordinate system of the particular output device of the user. In 3D, this scaling phase also includes either orthographic or perspective projection onto the viewer's virtual window. Stereo coordinates are computed and optionally supplied as the output of the system.

The characteristics of each of these subsystems follows.

Matrix Subsystem

The matrix subsystem provides arbitrary 2D and 3D transformation ability, including object rotations, translations, scaling, perspective and orthographic projection. Using this matrix, it is possible to define a completely arbitrary 2D or 3D viewing window and accomplish all affine transformations.

The matrix transformation subsystem is the first four Geometry Engines in the pipeline. Distributed over these Engines is a 4x4 matrix and an eight-deep, 4x4 matrix stack to accommodate picture subroutine structure. The top element of the stack is the *current* matrix that is used to multiply all incoming coordinates. Full, floating-point transformation of all incoming coordinates is done by this subsystem in 15 microseconds. Transformed points are supplied by this subsystem to the clipping subsystem.

The matrix stack allows the use of picture subroutines. All incoming matrix multiplication commands cause the *current* matrix (top-of-stack) to be multiplied by the incoming matrix. This allows a graphic object to be attached to a "parent" graphic object, thereby providing for a hierarchical drawing. This is done in the same way that a push-down stack is used in a general-purpose computer for storing arguments to subroutines. The **LoadMM** command causes a new matrix to be loaded onto the top of the stack, while a **MultMM** command causes the current top of the stack to be multiplied by the supplied matrix.

The matrix stack can be manipulated by user instructions. In addition to the normal **Push** and **Pop** commands, there is also a **StoreMM** command to provide for overflow handling of the stack if picture structure depth exceeds the eight levels allowed by the depth of the stack.

This subsystem also generates cubic and rational cubic curves [3]. An incremental difference matrix for forward difference curves can be loaded onto the top of the stack, and a special **Iterate** command causes the forward differences of this matrix to be computed, with the result that a new coordinate on the curve is output to the clipping subsystem. Conic curves are generated using rational cubics. New points on the curve are generated in 10 microseconds.

Clipping Subsystem

The four to six Geometry Engines following the Matrix Subsystem comprise the Clipping Subsystem. Each Geometry Engine in the clipping subsystem clips the objects to a single boundary (plane) of the viewing window. Thus, if there is no need or desire to clip objects to the *near* or *far* clipping boundaries, either or both of the corresponding Engines may be eliminated, with no undesired side effects. This might be done to decrease the cost of the system.

The clipping subsystem gets all input data after it has passed through the matrix subsystem, so that only transformed coordinates are supplied to it. It has no explicit registers that the user may manipulate. It always clips transformed coordinates to specific boundaries. The boundaries are made to correspond to particular boundaries of the user's drawing space by altering the transformation matrix so that the desired portion of the environment to be within the window is scaled to be within the standard clipping boundaries.

As an assistance in testing objects for intersection with the viewer's window, a special *hit-testing* mode is included in the clipping subsystem. This mode disables output of certain data from the Geometry System. For example, to *select* an object on the screen that is being pointed to by the input device *cursor*, *hit-testing* is enabled and a special hit-testing matrix is loaded into the *current* matrix. This matrix is computed from the screen coordinates of the cursor; it might correspond to a tiny window centered at the pointing device's screen coordinates. If anything comes out of the geometry system in this mode, it signifies that an object has passed within the tiny window near the cursor position. Of course, the hit-testing window may be of any size, so that this feature can be useful in area-select functions, as well.

To provide further information useful in identifying *how* objects pass through the *hit-window*, each drawing instruction gets from one to six bits set in it to signify which of the one to six clipping boundaries were intersected by the line-segment drawn. To assist in identifying the object, a special object naming convention is used, thereby providing a completely general selection and hit-testing mechanism.

Scaling Subsystem

The last two Geometry Engines in the pipeline are the Scaling Subsystem. This subystem converts output from the Clipping Subsystem to the coordinate system of the output device. This process causes the *window* on the user's drawing space, which is specified by loading the appropriate matrix into the Matrix Subsystem, to be mapped onto a *viewport* of the output device, which is specified by loading the scaling subsystem's *viewport* registers. The viewport registers allow up to 24-bit integer values, depending upon the coordinate system of the output device; they are the only device-dependent part of the system. In 3D, the mapping process includes an orthographic or perspective projection and stereo pair production.

Because the Geometry System is a homogeneous system that treats all three coordinates (x,y,z) the same, the Scaling Subsystem also maps the z coordinate. Thus, by loading the z viewport registers with appropriate values, either perspective depth values or intensity depth-cue values will be supplied by the Geometry System, according to the manner in which the output device interprets the z values. Of course, if no depth values are needed in the particular application, they may be discarded.

Either two or four integer values are output by the Scaling Subsystem for each coordinate point that comes out of the system. When two values come out, they are X and Y, in screen coordinates. If the Scaler Engines are configured properly, these four values are:

- X right - the x screen coordinate for the right eye.
- X left - the x screen coordinate for the left eye.
- Y - the y screen coordinate for both eyes.
- Z - the perspective depth value for both eyes.

Geometry System Computations

The matrix system does the computation:

$$[x' y' z' w'] = [x y z w] M,$$

where M is the top of the matrix stack and [x y z w] is the input vector to be transformed. The coordinates [x' y' z' w'] are supplied to the clipping subsystem, which clips them so that they satisfy

$$-w' < x' < w',$$
$$-w' < y' < w',$$
$$\text{and } -w' < z' < w'.$$

Note that these clipping boundaries are somewhat different from those used in most homogeneous clipping systems [4], in that the z coordinate is treated identically to the x and y coordinates. This simplifies the system, and is equivalent to all other homogeneous clipping systems if the correct matrix is used and the proper viewport scale factors are used.

After clipping, since all points coming out of the clipper satisfy these inequalities, the scaler does the final mapping to output device coordinates with the following computations:

$$D = (z'/w')*Ss + Cs,$$
$$Z = (z'/w')*Sz + Cz,$$
$$X = (x'/w')*Sx + Cx,$$
$$\text{and } Y = (y'/w')*Sy + Cy.$$

The coefficients Sx and Cx are the X half-size and X center of the viewport in the coordinate system of the output device. Similarly for the Y and Z values. The Ss and Cs values are explained in the next section.

Stereo Computation

The Geometry Engine can be used to obtain stereo pair pictures at no extra computational cost. Consider first the ordinary monographic case.

Monographic Case

In a system where the origin is the perspective projection point, the ordinary projection for 3 dimensional scenes [4] is to divide both x and y by z. That is, the screen coordinates of the point are given by

$$X = (x/z)*Sx + Cx$$
$$\text{and } Y = (y/z)*Sy + Cy,$$

where (Cx,Cy) is the center of the "viewport" and (Sx,Sy) is its half-size.

If homogeneous coordinates are used, these equations are modified to compute perspective depth. The transformation on [x,y,z] is modified to compute homogeneous coordinates as follows:

$$[x' y' z' w'] = [x y z 1] M.$$

M is chosen to yield

$$[x', y', z', w'] = [x, y, az+b, z],$$
$$\text{where } a = (1+N/F)/(1-N/F)$$
$$\text{and } b = -2N/(1-N/F).$$

N and F are the respective distances of the Near and Far clipping planes from the projection point. With these definitions, the projected coordinates are computed from

$$X = (x'/w')*Sx + Cx, \quad (1)$$
$$Y = (x'/w')*Sy + Cy,$$
$$\text{and } Z = (z'/w')*Sz + Cz = (a+b/z)*Sz + Cz,$$

where we have substituted the values of $z' = az+b$ and $w' = z$, from above.

This yields the same values for X and Y as before. In addition, however, it computes perspective depth, which can be useful in hidden-surface computations. With this computation, points at the Near clipping plane will be mapped into Cz-Sz and points at the Far clipping plane will be mapped into Cz+Sz.

Stereographic Case

For proper stereo, we wish to compute two different views, one for the left eye and one for the right eye. In other words, there are two different projection points that differ in a displacement in the x direction only:

$$Xright = ((x'+dx)/w')*Sx + Cx.right,$$
$$\text{and } Xleft = ((y'-dx)/w')*Sx + Cx.left,$$

where dx is half the distance between the two projection points (distance from the center of the head to each of the eyes). Cx.left is the center of the left projection viewport and Cx.right is the center of the right projection viewport. The Y and Z coordinates are unaffected.

Defining Cx.offset to be the offset of the right and left viewports from a "center" viewport, Cx, we have

$$Cx.left = Cx - Cx.offset$$

and Cx.right = Cx + Cx.offset.

The foregoing equations then become

Xright = (x'/w')*Sx + Cx + { (dx/w')*Sx + Cx.offset }
and Xleft = (x'/w')*Sx + Cx - { (dx/w')*Sx + Cx.offset },

or

Xleft = X + D,
and Xright = X - D,

where X is the "normal" X computation in Equation 1 and D is the quantity in brackets.

Note that D is a computation like that of X,Y and Z in Equation 1. In other words, it involves a division, a multiplication and an add. Inspection of the third of Equation 1 suggests that we define "stereo viewport" parameters as follows:

Ss = dx*Sx/b,
and Cs = Cx.offset - a*(dx*Sx/b).

Then the quantity D is computed to be

D = (z'/w')*Ss + Cs,

giving the required result for D when these substitutions are made.

The Geometry Engine has four floating-point function units; two are required to accomplish one computation of the sort

A = (B/C) * E + F.

Therefore, one Engine will perform two of these computations, for example for the X and Y coordinates. Since another Engine is required to compute Z, it has two free units that can be computing D as well, using the Ss and Cs values defined above. If the Engine computing D and Z is put in the pipeline before the X and Y Engine, the X-Y Engine's microcode can compute X+D and X-D, outputing the four values [X+D,X-D,Y,Z]. Of course, if no stereo is desired, but Z is still needed, the coefficients Ss and Cs can be zero. The Geometry Engine implements this stereo computation, and when properly configured, will output these four quantities.

Programming the Geometry System

The Geometry System is a slave processor. It has no instruction fetch unit; it must be given every instruction and data value by a controlling processor. Likewise, the display controller must take each value that comes out of the Geometry System.

The instruction/data stream supplied to the system is a high-level graphics instruction set mixed with coordinate data. Instructions and data are supplied to the system via its input port, which is the set of input pins of the first Matrix Subsystem Engine, and output data and instructions are taken from its output port, which is the set of output pins of the last Scaling Engine. A convenient view of the system is as a hardware subroutine; in fact, this is precisely the first way it will be used, as a hardware subroutine to the IRIS processor/memory system, which is based on the Motorola 68000 and IEEE Multi-bus.

Input data must always be in user's virtual-drawing (integer or floating-point) coordinate system, and except in special non-display circumstances such as hit-testing, output data is always in the coordinate system of the user's output device.

Instruction Set Summary

The instruction set for the geometry engine partitions into three types:

- **Register Manipulation** - These instructions alter the matrix, matrix stack, or viewport registers. They are used to set the *window* for a particular view of the virtual drawing, load the *viewport* registers, change the matrix or matrix stack to draw a different object, orient a particular object (rotate, translate, etc.) or save the state of the matrix stack for later restoration. Instructions in this category are:
  - **LoadMM** - Load the following 16 floating-point data values onto the top of the stack, destroying the current matrix. The 16 floating-point numbers are the 4x4 matrix.
  - **MultMM** - Multiply the *current* matrix on the top of the stack with the following 4x4 matrix.
  - **PushMM** - Push all matrices on the stack down one position, leaving the current top of stack unaltered. (After this operation the second stack position is a copy of the top of the stack.)
  - **PopMM** - Pop all matrices in the stack up one position.
  - **StoreMM** - Store the top of the matrix stack. This instruction input to the geometry system causes the StoreMM instruction, followed by the 4x4 matrix (16 floating-point numbers) to come out of the Geometry System at its output port. It can be used to save the complete state of the matrix stack.
  - **LoadVP** - Load the viewport registers. Following this instruction, eight 32-bit numbers describing the viewport parameters must be supplied.
- **Drawing Instructions** - These instructions actually cause graphic objects to be drawn. All drawing instructions are followed by four 32-bit floating-point numbers, representing the (x,y,z,w) coordinates of the point being supplied to the Matrix Subsystem for transformation. Each drawing command assumes that there is a current point in the drawing, for example the current pen position in a virtual-space plotter. Certain instructions update that position, while others cause things to be drawn from that point. We refer to this position as the Current Point. Assuming clipping does not eliminate them, each of the following instructions except **Curve** comes out of the Geometry System at its output port, followed by the device coordinates.
  - **Move** - Move the Current Point to the position specified by the floating-point vector that follows.
  - **MoveI** - Same as **Move**, but integer data is supplied.
  - **Draw** - Draw from the Current Point to the position specified by the following data. Update the Current Point with this value after drawing the line segment.
  - **DrawI** - Same as **Draw**, except that integer data is supplied.
  - **Point** and **PointI** - Cause a dot to appear at the point specified in the following data. Update the Current Point with this value after drawing the point.
  - **Curve** - Iterate the forward differences of the matrix on the top of the matrix stack; issue from the Matrix Subsystem to the Clipping Subsystem a **Draw** command followed by the computed coordinates of the point on the curve. The Current Point is updated just as with the **Draw** command. This command should *not* be followed by data as with the other drawing commands.
  - **MovePoly** and **MovePolyI** - In Polygon mode, move

the Current Point to the position supplied by the following data. This command must be used rather than **Move** if a closed polygon is to be drawn.

- **DrawPoly** and **DrawPolyI** - In polygon mode, same as **Draw** command.
- **ClosePoly** - Close the currently open polygon, flushing the polygon from the clipping subsystem.

- **Miscellaneous Commands** -
  - **SetHit** - Set Hit Mode. This causes the state of the Clipping Subsystem to change so that only commands, and not data, are output. Refer to the "**Selection and Hit-testing**" section for a complete description.
  - **ClearHit** - Clear Hit Mode. This restores the state of the Clipping Subsystem to normal. Refer to the "**Selection and Hit-testing**" section for a complete description.
  - **PassThru** - This instruction allows the passing of a variable number of 16-bit words through the geometry system unaltered and uninterpreted. It is useful for passing instructions and data that are unique to the display controller and that have no meaninng to the Geometry System.The number of words to be passed through is specified by a 7-bit field in the instruction.

Selecting and Hit-testing

In an interactive computer graphics environment it is frequently necessary to select certain objects that appear in the display for special attention. This is usually done with the aid of some type of input device, such as a light-pen, mouse, tablet or joy-stick.

If the input device being used is a light-pen, the common selection mechanism varies, but involves detecting in hardware when the "beam" of the CRT is under the field of view of the light-pen. This approach is good for pointing at objects on the screen but poor for entering new objects into the drawing, because a tracking mechanism must be drawing some type of tracking object that the light-pen must be sensing. Because of the extra expense of the light-pen tracking mechanism and because many people no longer believe it necessary to actually point to objects directly on the screen, the light-pen is not feasible in low-cost systems.

The alternatives to the light-pen, the tablet and mouse (we chose to ignore the joy-stick), are useful for entering new data into drawings, but without an extra mechanism, they are poor for pointing at existing objects in a drawing. The hit-testing mechanism in the Geometry System solves this problem.

The common software mechanism for doing this selection task is to check each object to see if it is in the selection area. This selection area might be an area specified by identifying some portion of the drawing space to check objects against or it might be a small neighborhood around the cursor, which is tracking the position of the mouse or tablet. Intelligent operations can be done to reduce the amount of time spent in checking. For example, the bounding box around an object can be tested to see if any portion of the object is in the selection area; if it is not, then none of the object is in the selection area and therefore need not be further tested. This selection task is basically a clipping task, and the Geometry System has a special mode for handling it.

The Hit-testing mode disables all data from coming out of the Geometry System. However, specific drawing instructions still come out of the system, missing their corresponding data. Thus, in hit-testing mode, if anything comes out of the output port of the system, this means that there was a "hit." In other words, something was in the selection area established by loading the selection matrix into the Matrix Subsystem.

For a completely general selection mechanism, one might not only like to know whether an object passes through the selection window, but also which boundaries it intersects, or whether it is completely contained within the selection area, or perhaps completely surrounds the area. To accommodate these needs, the Geometry System provides information in the form of "hit-bits" that tell which of the six clipping boundaries are intersected by each drawing command. In this way, the device that is receiving Geometry System output may assemble the necessary information by "integrating" the various "hit-bits" from successive drawing commands used in drawing the object.

Hit-testing is useful only when combined with a naming mechanism for identifying the objects being drawn. This can be done by loading a *name* register in the display controller before drawing each object that is to be identified with a hit. This can be done using the **PassThru** instruction.

Character Handling

Characters provide a special problem for any geometric transformation subsystem. Of course, characters may be defined as strokes, or vectors, and supplied just as all other data to the Geometry System, but since the number of strokes to make up a character might be quite large, we ordinarily do not wish to draw characters in this way. On the other hand, any other approach will not provide for complete, general rotations, etc. of 3D characters. As a result, most systems must make a compromise and provide characters as a special case.

The usual problem with characters is that if they are a special case, then clipping them is a special case. The Geometry System clips characters only if they are defined as strokes, just like all other data. However, since it must make possible the clipping of special-case characters and character generation in the display controller, the **LoadVP** instruction and corresponding data is always passed on to the output port of the system. The reason for this is that this data defines the boundaries of the *character* clipping window in the display controller.

Mixing special-case characters and graphics presents another problem. There are two cases:

- Putting characters in a drawing - this is handled by combining special sentinels to the display controller via the **PassThru** command with the **Point** command. The **Point** command is used to position the beginning of the character string. The Raster Subsystem, which is designed as a companion to the Geometry Subsystem, does the actual character clipping. Completely general character clipping is accomplished by proper use of these subsystems together.
- Putting a drawing with characters - This case is straightforwardly handled by properly modifying the tranformation matrix to reflect the character clipping window position. Then drawing can proceed as usual. The particular modifications for each case are handled by the software package mentioned above.

The IRIS Graphics System

The Geometry System is being implemented on the a system called the Integrated Raster Imaging System, IRIS, which consists of the following components:

- A processor/memory board with the Motorola 68000 and 256k bytes of RAM; the memory can be expanded to 2M bytes. The 68000 microprocessor executes instructions in the on-board memory at 8 MHz. This memory is fully mapped and segmented for 16 processes. Additional memory is accessed over the Multibus at normal Multibus rates.

- A Geometry Subsystem, with a multibus interface, FIFO's at the input and output of the Geometry System and from ten to twelve copies of the Geometry Engine.
- A custom 1024x1024 Color Raster Subsystem, with high-performance hardware for polygon fill, vector drawing and arbitrary, variable-pitch characters. The hardware and firmware provide for color and textured lines and polygons, character clipping, color mapping of up to 256 colors and selectable double or single-buffered image planes.
- A 10 Megabit EtherNet interface board.

Summary

The Geometry System is a powerful computing system for graphics applications. It combines a number of useful geometric computing primitives in a custom VLSI system that has a future because of its scalable nature. It is quite likely that within 5 years the system will be implemented on one, 1/2-million transistor, integrated-circuit chip, with a correspondingly reduced cost and increased speed.

Acknowledgements

Many people provided advice and suggestions during the two years over which this project has been done. Marc Hannah's masterful ability with VLSI Design Tools and UNIX and his graphics understanding were indispensible. Professor John Hennessy provided an indispensible microcode development tool in SLIM, and his willingness to help us when in need is appreciated. Lynn Conway of Xerox PARC made resources available during the formative stages of the project, and without them, it probably would not have been carried out; we are indebted to her for this. Forest Baskett of Stanford made it possible by supporting us in the early stages. Dick Lyon was an important first advisor on IC design. Martin Haeberli was very helpful in the testing phase. Valuable conversations were had with Chuck Thacker, Bob Sproull, Alan Bell, Martin Newell, Ed Chang, Danny Cohen, Doug Fairbairn, John Warnock, Chuck Seitz, Carver Mead, and Lance Williams. Hewlett-Packard Corporation fabricated the first copy of the first part of the data-path, and Bob Spencer and Bill Meuli of Xerox PARC's Integrated Circuits Laboratory fabricated the first fully functional copy of the entire chip.

We are especially grateful for the enthusiasm and support of Xerox Corporation's Palo Alto Research Center; this project could not have been done without the support of the insightful people there.

The research was supported by the Advanced Research Projects Agency of the Department of Defense, DARPA, under contract number MDA 903-79-C-0680.

References

1. Clark, J.H. "A VLSI Geometry Processor for Graphics." *Computer 13*, 7 (July 1980), 59-68.

2. Clark, J. H. Graphic Display Processing System and Processor. Patent Pending.

3. Clark, J. H. Parametric Curves, Surfaces and Volumes in Computer Graphics and Computer-Aided Geometric Design. Tech. Rept. 221, Computer Systems Laboratory, Stanford University, November, 1981.

4. Newman, W. and Sproull, R. F.. *Principles of Interactive Computer Graphics*. Addison-Weseley, Reading, Mass., 1980.