A Contour Display Generation Algorithm for VLSI Implementation

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

Abstract

Recent articles have discussed the current trend towards designing raster graphics algorithms into VLSI chips. The purpose of these design efforts is to capture some of the real-time animation capability found in vector graphics systems. Currently, real-time vector graphics animation is limited primarily to operations involving coordinate transformations. In order to enhance this animation capability, frequently encountered vector graphics algorithms that require the high speed, parallel computation capability of VLSI must be identified. Real-time contour display generation from grid data is one such algorithm. This paper describes the specifics of a contour display generation algorithm, the architectural framework of a processor that performs this algorithm and the architectural requirements of such a processor.

The contouring algorithm is based on a data structure, the contouring tree, whose regularity and amenability for parallel computation make it an ideal candidate for VLSI. The architectural framework for a contouring processor chip that performs this algorithm for the real-time environment of interactive graphics is discussed, particularly the issues of memory size and data distribution. A model of the contouring process is created in order to determine the necessary physical parameters of the contouring processor in this architectural framework. Conclusions are drawn concerning the feasibility of producing a VLSI chip that performs this contouring algorithm.

CR Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture - vector display devices; I.3.3 [Computer Graphics]: Picture/Image Generation - display algorithms; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - Curve, surfaces, solid and object representations; Geometric algorithms, languages and systems; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Animation; B.7.1 [Integrated Circuits]: Types and Design Styles - VLSI (very large scale integration); E.1 [Data Structures]: Trees

General Terms: contouring, VLSI

This work has been supported by the following grants: NIH1 P01 GM 24483-01A1 and NIH5 T32 GM07564.

1.0 INTRODUCTION

Recent articles have discussed the current trend towards designing graphics algorithms into VLSI chips [3,6,8]. Most of these efforts have concerned limited functions, such as frame buffer control. Of these efforts, the majority are directed towards raster graphics because of the simplicity and regularity of pixel operations. These projects have aimed at capturing some of the real-time animation capability found in currently produced vector graphics systems. In vector graphics, real-time animation is seen primarily as object rotation and translation. This is because most vector graphics systems are based on minicomputers that have little hardware for special graphics other than a matrix multiplier. Sophisticated examples of vector graphics animation are usually relegated to the non real-time environment of motion picture film.

The trend towards the design of graphics algorithms into VLSI chips can change this situation for vector graphics systems. VLSI offers the possibility of multiple, computational units operating in parallel within the boundaries of a single chip. Because both processing elements and memory elements can be easily implemented in VLSI, one is encouraged to find structures formed from these elements that can use this available concurrency [5]. In order to take advantage of these VLSI capabilities with respect to vector graphics, one must identify graphics algorithms that are partitionable into relatively simple subproblems capable of the type of parallel

solution VLSI has to offer. Candidates for this treatment are the graphics algorithms frequently encountered in graphics programs and, of this group, those that require more than one-thirtieth of a second to compute. One algorithm that has both of these characteristics is contour display generation from grid data [11]. This paper discusses the specifics of this contouring algorithm, the architectural framework of a processor that performs this algorithm and the architectural requirements of such a processor.

As a graphics algorithm, contour display generation is frequently used in X-ray crystallography, computer-aided tomography, and other applications for which grid data is collected. It is generally depicted as a computationally slow operation whose output is sent to a plotter or film recorder. A number of papers have been written documenting "breakthroughs" that increase the speed of such contouring algorithms. One author has recently reported that his contouring subroutine used one second of central processor time on NCAR's Control Data 7600 [10]. Although a contour generation program of this speed is useful for static situations, it is found to be lacking when user interaction is important and the succession of images caused by contour level changes is meaningful.

One application in which real-time animation is important is the determination of molecular structures from the electron density data generated by X-ray crystallography [2]. Such an operation is executed interactively by using a computer graphics program that displays a Dreiding (stick) model of the working molecule, inside a contour display of the corresponding region of the molecule's electron density grid. In addition to the graphics function, the computer program monitors a series of signals generated by the user, while turning the various knobs on a control console [12]. The values read from these knobs are interpreted by the program as modifications to either the working molecule or the contour display. Modifications to the molecule cause flexible bonds to be rotated and bonds to be lengthened; modifications to the contour display produce an increase or decrease of the contour level. The goal of this process is to produce the stick model of the molecule that best fits inside the given electron density data set. The user can determine whether or not the model fits the density grid by modifying the contour level, shrinking the contour surface to the working molecule. Similarly, the user can expand the contour surface from the stick model for better visibility. This function requires that the hardware have the capability to rapidly change the contour display as its contour level changes.

Another application that requires the real-time animation of contour displays is connected with the systematic search procedures used in Drug Design research at Washington University, St. Louis, Missouri [4]. In this project, all possible conformers of a molecule are generated by a systematic, incremental rotation of the flexible bonds hypothesized for the molecule. Each molecular conformation generated by this process is checked against steric (Van der Waals's) constraints and various user set, geometric constraints. The molecular conformations of the molecules that pass the constraints during a run of this systematic search process are visually examined by a drug designer/bench chemist for classification and verification. One of the steps in verification is to generate an energy surface for passing molecular conformations. This surface is constructed through a contouring process. Because it outputs several thousand passing conformations, the process must be performed rapidly.

A VLSI chip that generates contour displays for such demanding applications must be able to produce and distribute a new picture in the amount of time it takes the graphics hardware to change display frames. This is less than one-thirtieth of a second. Any greater amount of time is discernable by the viewer, either as a flicker or a hesitation in the picture update. In fact, one-thirtieth of a second is discernable to many people, making one-sixtieth of a second a more desirable time for the change of display frames. [7].

2.0 CONTOURING ALGORITHM

2.1 Contour Display

The contouring algorithm is best described within the context of a contour display. A contour display is a graphic representation of all the isovalued points in a given region of space. The region of space for this algorithm is a two-dimensional grid of data values. Its graphic representation is a set of line segments that run through interpolated equivalued points on this two-dimensional sheet. A three-dimensional grid can be contoured by this algorithm by graphically combining the line segments generated from all possible orthogonal, two-dimensional sheets of the three-dimensional grid. This produces a chicken-wire-like view of the three-dimensional surface at a particular contour level.

The contour level is the value at which a particular sheet is contoured. A given, two-dimensional sheet has a continuous series of contour displays between its minimum and maximum grid values. The difference between contour displays from one level to the next is not large if the difference in levels is not large. This is the basis for the formulation of a data structure that represents the continuum of contour displays-- the contouring tree [11]. The contouring algorithm proposed in this paper is based upon a reduction in the scope of the contouring tree.

2.2 Contouring Data Structure

For this discussion a contouring tree is a data structure that represents a 2 x 2 grid region in a form that permits the user to easily retrieve the contour display for any given contour level. A contouring tree is generated for every 2 x 2 subgrid of a larger, two-dimensional sheet. The creation and use of the contouring tree is best described with an example of a small grid.

Figure 1 depicts the line segments for contour levels 50 and 100. The contour at level 100 is a closed contour that forms a single, connected loop. The contour at level 50 is an open contour. Figure 2 presents a contouring tree for the lower, lefthand 2 x 2 subgrid of Figure 1. The edges of the contouring tree correspond to the directed, downhill edges inscribed on the 2 x 2 subgrid. The edges of the tree are ordered, maintaining the same counterclockwise ordering as in the original grid. The dashed lines in Figure 2 indicate the order in which the coordinates are generated from the contouring tree for the display at levels 100 and 50. The boxed "[1]" under nodes 2 and 5 indicates that a setpoint display command should be generated for any coordinate that is created along the edges 1-2 and 2-5, respectively. We can best describe the features of the contouring tree in the course of the following description of the processes of tree creation and display generation.

2.3 Creation Of Contouring Trees

The first step in the process of creating contouring trees is to choose which of the four points that border the 2 x 2 region is to be designated the maxima. The purpose of a maxima point is to have a point to serve as the root of the contouring tree. In this context, the maxima can be the point that has: (1) the maximum value for the entire two-dimensional grid, or (2) the maximum value only for the concerned 2 x 2 subgrid, or (3) the first encountered point of multiple equal maximum values for the 2 x 2. This latter condition can be basis for the maxima because, in an ordered consideration of grid points, with the maxima set to the first encountered maximum valued point, all equal and maximum valued points appear as roots of contouring trees in other 2 x 2 regions. For the selection of maxima in this illustration, we have considered the four grid points in counterclockwise order.

The selection of a local maxima for the 2 x 2 region determines a large part of the configuration of the contouring trees. The root of the tree is the maxima and the remaining three points are the immediate descendent nodes of that root. The selection of the root also determines the order in which the three descendent nodes are added onto the root. In fact, once the root is chosen the descendent pointers are easily indexed from a small table.

At this point in the procedure, only two edges of a total of five remain unattached to the descendent nodes of the root. The first of the two remaining edges can be attached as the descendent of either the first node attached to the root or the second node attached to the root in counterclockwise order. To select the attachment for the first edge, one compares the grid values on both ends of the free edge. The free edge is attached to the node that has the highest value. This attaches the fourth edge of the 2 x 2. The fifth and remaining free edge is attached like that the fourth. It can be added onto either the second node attached to the root or the third node attached to the root in counterclockwise order. The conditions for this final step differ from the latter only in the possibility that this fifth edge would be the second descendent edge added to the second descendent node of the root. The addition of this edge completes the formation of the contouring tree for the 2 x 2 grid section.

Display pen command information must be placed in the contouring tree when the edges are attached during tree construction. Two display commands are required for drawing the contour display: setpoint and drawto. The setpoint command causes the display pen to be moved in a non-drawing mode and set on a specified coordinate. The drawto command causes the display pen to draw from the current position of the pen to a specified coordinate. One must place a setpoint command in the contouring tree on the lower valued node of each perimeter edge whose downhill direction is counterclockwise. This command appears in Figure 2 as a boxed "[1]" on nodes 2 and 5, with respect to edges 1-2 and 2-5. All other nodes have drawto commands. The display pen command information indicates when a line enters the 2 x 2 from a neighboring 2 x 2. This portion of the algorithm is described in greater detail in [11]. After this information has been entered, the contouring tree can be used for the generation of coordinates and display pen commands at the selected contour level.

2.4 Display Generation From A Contouring Tree

Only four configurations of the ordered tree can be created for any 2 x 2 subgrid (Figure 3). Because of this limited number, one can select the tree configuration during the tree construction process and use that configuration information to create a list containing the order in which the tree's nodes should be considered for display generation. This list is termed the "enumeration list." The order of the nodes on this list is top-down from the root and counterclockwise. Generated with the enumeration list is a second list that specifies the "next node" to consider if one places a coordinate on an edge of the tree. With this second list one can skip over nodes lower on the path formed by the connected set of nodes from the root to an external node. The edge under consideration list of Figure 2 serves to remind the user that when one is considering node 2 one is determining if a coordinate is to be

generated along the edge 1-2.

The nodes on the enumeration list are sequentially examined. If the grid value contained at a node is less than or equal to the currently selected contour level, a coordinate is generated, via linear interpolation, for the edge under consideration. At the same time a display pen command is issued. This display pen command is for the operation contained at the lower valued node of the edge from which the coordinate was generated. After the generation of the coordinate and display pen commands, the position in the enumeration list is advanced according to the value of the next node list element for that lower valued node. If the grid value contained at a node is greater than the currently selected contour level, consideration is given to the next node on the enumeration list. This process continues until either the enumeration list is exhausted or the next node list causes it to be exhausted.

2.5 Algorithm Parallelism

To this point we have described the method for the generation and use of the contouring tree for a single 2 x 2 grid region. In order to generate the contour display for a larger plane, this algorithm must be executed for every 2 x 2 subgrid of this plane. The processing involved in the display generation for each of these 2 x 2 subgrids is independent of that performed for any of the neighboring 2 x 2 grid regions. One may think of the computation for each 2 x 2 as occurring in a "cellular" processor only concerned with that 2 x 2 subgrid. Synchronization during the contour generation process is not required, nor is complex data communication. The only communication necessary is that of transmitting grid endpoints, contour levels, and control signals to each 2 x 2 cellular processor and that of retrieving the display coordinates and commands from each cellular processor. Because data communication is minimal and there is no requirement for synchronization of the contouring procedures for each 2 x 2 subgrid, the potential for concurrency is quite large.

3.0 ARCHITECTURAL FRAMEWORK

From the above discussion of algorithm parallelism, we can determine the type of VLSI layout necessary for the 2 x 2 cellular processor and its interconnections (see Figures 4 and 5). Before describing this VLSI architecture, we must first understand how the capability for real-time contouring is to be used. For this discussion, we describe the necessary characteristics of a "contouring processor." For initial consideration, we should think of this processor as a single VLSI chip, although examination may show that the processor might require multiple chips because of VLSI density limitations.

The architectural framework of this contouring processor is that of a device used in conjunction with the typical display processor/minicomputer system. In this system, the host minicomputer initiates the contouring processor whenever a new contour level is detected or a new grid is delivered. The contouring processor computes the new display according to the algorithm discussed and deposits the resulting coordinates and display pen commands into the picture memory of the display processor. A display system can contain several contouring processors. The number of contouring processors required depends on two factors: the maximum size of the grids that one chooses to contour in real-time and the total number of the maximum size grids that must be contoured by each processor.

In order to specify these factors for a particular display system, we must choose a problem requiring contouring. Using the molecular modeling program presented in the introduction as the typical application, we find that the largest three-dimensional grid of concern is a cube of 30 units on each side [2]. As discussed, a three-dimensional grid is contoured by generating the display for all of the possible orthogonal planes that compose the grid. One must therefore contour 90, 30 x 30 planes in order to complete the picture representing the 30 x 30 x 30 cube.

3.1 Architectural Modeling

After the selection of a target application and a figure for its maximum grid size, one must compute a value for the maximum contouring capability of a single contouring processor, i.e., the number of 30 x 30 sheets it can contour in one-thirtieth of a second. To obtain this value for the contour generation process, one constructs a model of this process and monitors the behavior of this model while it simulates the performance of the contouring processor.

Aho's [1] discussion of algorithm analysis methods is pertinent to this formulation. In Aho's computational models the key to analyzing the time complexity of an algorithm is the ability to assign a time cost on an instruction by instruction basis for the process under consideration. Central to his modeling methodology are the assumptions of a uniform time cost for each instruction and a varying time cost, depending on the operands required at each instruction. Although the computational models of [1] are generally used for determining the time-order of magnitude of simple algorithms, they can be extended to produce a modeling methodology for more complex processes. The design under consideration in this paper takes this approach, rejecting, however, Aho's analytical method of determining time costs in favor of an explicit totaling of the memory reference costs for a series of randomly generated grids. This choice is based on the assumption that the execution of a single memory reference takes the same time or longer than the execution of a single instruction.

For this model, the memory reference totals are recorded in three parts. These are the grid to contouring processor transfer time, the actual display generation time, and the coordinate/display pen command delivery time. Upon the assumption that the transfer is accomplished serially, the grid transfer time is computed by totaling the number of grid elements to be transferred to the contouring processor from the host minicomputer. The display generation time is computed by totaling the memory references required to compute the display for a single 2 x 2 subgrid. In this case, all the 2 x 2 subgrid displays are understood to be computed in parallel. The time for the delivery of the coordinate/display pen commands is computed by totaling the maximum number of coordinates possible from a single 30 x 30 sheet and multiplying this value by four. The value of four was determined upon the assumption that each coordinate/display pen command set is representable by a quadruple of memory locations.

One can easily compute the first memory reference total-- the transfer of a single 30 x 30 grid from the host minicomputer to the contouring processor. Assuming that each element of this grid is representable by one memory location and allowing additional references for overhead, one finds 1000 memory references to be a reasonable approximation of the reference total for this transfer operation.

It is more difficult to compute the second memory reference total-- the actual time for display generation. In order to accomplish this task within the modeling framework discussed, one must program the contouring algorithm as it would be written for the contouring processor. One then evaluates the probable memory reference contribution at each branch point of that program. Having performed these two steps, one executes the contouring program with imbedded memory reference counters. The number of references, approximately 2500 per 2 x 2 subgrid, is generally constant for all 2 x 2 subgrid configurations. This number is constant because the algorithm consists primarily of table lookups derived from the initial maxima choice.

The third memory reference total-- the coordinate/display pen command transfer from the contouring processor to the picture memory of the display processor, is the largest part of the 30 x 30 memory reference count. This reference total depends on the maximum number of coordinates and display pen commands that can be generated for a single 30 x 30 grid. The maximum number of coordinates and display pen commands for a single 2 x 2 subgrid is four. Given 841 2 x 2 subgrids in a single 30 x 30 sheet, there are a maximum of 3364 coordinate/display pen command quadruples. Tests with randomly generated grids, however, have shown 3364 to be too large a number. The largest number encountered is 2600 coordinate/display pen commands. Using 2600 coordinate/display pen commands as the limit and multiplying by the size of the quadruple, we find that approximately 11,000 references are needed to transfer the largest display for a single 30 x 30 grid.

Summing the three reference counts, the memory reference total for contouring a single 30 x 30 grid is approximately 15,000. Assuming a rather fast memory and comparable processor and using 100,000 references as the maximum number of references to be allowed in one-thirtieth of a second, one finds that six, 30 x 30 grids can be contoured by each contouring processor. This means that fifteen contouring processors are needed to contour the 90, 30 x 30 grids that make up the display for the 30 x 30 x 30 cube.

3.2 Architectural Problems

This model poses two serious architectural problems: (1) There are difficulties of memory contention in the delivery of the contour display to the picture memory; and (2) assuming that the data can be delivered, there are possibly more vectors than currently available vector graphics systems can draw. Because the second problem is more serious, we consider it first. The maximum number of vectors that we expect to generate for the 90, 30 x 30 grids is 234,000. The computer graphics manufacturer Evans and Sutherland claims for its latest product, the PS-300, the capability to display 95,000 vectors. The difference between the two numbers, although not an order of magnitude, is sufficiently large for concern. The total number of vectors produced by the contouring processor must be reduced to a quantity that the display device can handle. The ideal location for this reduction is in the contouring processor, before the delivery of the coordinate and display pen commands.

One method by which to reduce the total number of vectors is based on the actual visibility of the object after it has been transformed by a viewing matrix. In this method, coordinates that are output from the contouring process would be transformed by that matrix and examined to determine whether they were visible within the boundaries of the display screen. Vectors entirely out of range of the screen's boundaries would not be passed on, and vectors inside the boundaries would be passed on. Vectors that cross the screen's boundaries would be clipped inside the contouring processor and the clipped version of the coordinates would be sent to the display device.

This method, however, provides only a partial solution. A problem occurs when the viewing matrix projects the entire picture within the screen's boundaries. Under these conditions, it would be necessary to make a second visibility check in order to determine if the vectors generated from the transformed coordinates could be plotted as single points, i.e. degenerate vectors. The procedure for making this check would depend upon the limitations of the display device. If the display device could not handle the required number of vectors, it might be necessary to provide a degeneracy window value as a tuning device. This window value would allow one to map small lines into single points. The test for degenerate vectors would be performed at

the same time as the check for visibility within the screen's boundaries. The most difficult part of this solution would be the actual coordinate transformation operation. Assuming that floating point operations were performed as part the contouring process, this reduction would not be difficult. The additional capabilities of coordinate transformation and the check for vector visibility and degeneracy could be accommodated in a total of approximately 4000 references.

The other architectural problem is the memory contention during the delivery of the contour display to the picture memory of the display processor. This problem arises from the original architectural framework. The framework is that of a typical display processor/minicomputer system. Current display processor systems have only one picture memory. This picture memory is accessed by the display processor, in order to refresh the screen, and by the host minicomputer, in order to deposit the latest display update. A problem occurs when fifteen contouring processors access a single picture memory. Under these conditions the contention for memory is significant, preventing the delivery of the latest picture and the display of the current picture.

One solution to the problem of memory contention would be to partition the picture memory. A memory partition would be provided for each contouring processor, insuring that each contouring processor only shares its portion of display memory with the display processor. The memory could be organized so that it logically appeared as one memory to the display processor. To resolve any remaining contention between the single contouring processor and the display processor one would provide a precedence mechanism that favors the display processor, because display refresh must occur on time.

The architectural problems encountered in the design of the contouring processor appear resolvable. The addition of coordinate transformation as an integral part of the contouring processor is an easily compartmentalized and isolated system change. It can be imbedded in the contouring processor with little impact on the framework of the display system. The only effect foreseen occurs in passing the viewing matrix to the contouring processor. By contrast, the architectural problem of display memory partitioning significantly affects the design of the display system. This was expected, because experience with current vector graphics systems has shown that the most frequent bottleneck in updating a display is the transfer of data from the host computer to the display memory via a single, serial pathway. This is the most serious impediment to the animation capability of current vector graphics systems. The advent of special graphics processors, such as the contouring processor, will require the reconsideration and redesign of the commonly used display processor system.

3.3 VLSI Perspectives

In the preceding sections we have defined the architectural framework and required capabilities of the contouring processor. With this overview of the architectural requirements, we can begin to describe how such an architecture can be implemented in VLSI. Figure 4 is a schematic view of the 2 x 2 cellular processor. The interconnection scheme for a set of these 2 x 2 cellular processors is shown in Figure 5, which is derived from the discussion of algorithm parallelism in Section 2.5. In reference to these two figures, the VLSI decision that must be made is how much can be placed on a single VLSI chip. It would be ideal to be able to place all the cellular processors for all six planes of the 30 x 30 grids described at the end of the modeling discussion in Section 3.1 onto a single chip. This would require a single chip total of 5,046 cellular processors, each of the complexity shown in Figure 4. This is clearly impossible for reasons of VLSI density limitations. The practical question, then, is how many 2 x 2 cellular processors can be put on a single chip. In order to answer this, we must first consider the hardware requirements, as sketched in Figure 4.

3.4 Hardware Requirements

There are 5 functions that the 2 x 2 cellular processor has to be able to perform on external command: (1) collect grid data from the system bus, (2) collect the view matrix from the system bus, (3) receive the contour level from the system bus, and execute the contouring/ coordinate transformation procedure, (4) output the generated coordinates to the system bus for transfer to the picture memory of the display processor, and (5) reset and count. Three of the five functions require very little in the way of special hardware because they are only operations that transfer data to/from the cellular processor's memory from/to the system bus. The only special requirement for this data transfer operation is that it be capable of addressing each cellular processor. When collecting grid data from the system bus, the cellular processor needs to be able to ignore all grid data not specifically addressed for it. The cellular processor also needs to be able to ignore output commands on the system bus. This implies that each cellular processor has knowledge of which 2 x 2 subgrid it represents and, further, that the external system bus control line indicates, at some time, which cellular processor is being addressed. The external address presentation to the cellular processor is containable within 15 bits, given the 30 x 30 x 30 limitation discussed earlier. This external address is placed into the External Data Register of each cellular processor upon the initiation of each grid data delivery cycle and each coordinate retrieval cycle. The actual data delivery/retrieval then follows the address indication. The next question is how does the cellular processor receive and maintain its copy of this address. There are two possibilities.

The first possibility is to have the address of each cellular processor permanently contained within the cellular processor in a small ROM. This would require a slightly different VLSI layout for each cellular processor. This is clearly unsuitable because we can benefit the most from VLSI if we strive towards "regularity." Regularity, at least for VLSI, means many copies of the same thing. The best mechanism, then, for assigning addresses to the cellular processors is one in which the layout for each cellular processor is identical.

The second possibility for cellular processor address assignment is that of a chain of "count enabling" wires, one input to and one output from each cellular processor. With this wire, each processor is sequentially addressed in the following fashion: simultaneous to the initiation of the external reset function, the first cellular processor in the chain has its Enable In line set high. The reset/enable combination causes the cellular processor to increment the address held in the External Data register by one. This address is then returned to the system bus at the same time that the Count Enable Out line is set high. This enables the neighboring cellular processor, which follows the same procedure. The end result is that each cellular processor has a unique "address" which can be used, through judicious upper level control, to deliver/retrieve data on an individual cellular processor basis. In addition to meeting the regularity criterion mentioned above, this mechanism is one that can easily travel across the multiple chip boundaries necessary in the composition of the contouring processor.

Having shown that four of the five functions the cellular processor performs on external command can be accomplished with a minimum of special hardware, we now examine the requirements of the final function. The hardware requirements for this function, that of executing the contouring and coordinate transformation procedures, are not substantial, although the execution sequence for this hardware is not trivial. The cellular processor needs an ALU capable of integer multiplication, division, addition, and subtraction. From the description of the algorithm it appears as if floating point arithmetic were required. The contouring procedure, however, was originally implemented using entirely integer arithmetic and should therefore present no problem for such a limited function ALU. Coordinate transformations are likewise capable of simulation via integer arithmetic.

Another piece of hardware important when considering the contouring/transformation function is that of memory. Figure 4 shows a 128 word (16 bit) RAM that should suffice for all contouring tree construction, grid collection, coordinate generation, and coordinate transformation operations expected in the cellular processor. There are other pieces of hardware that take up space on the VLSI chip, such as the ALU input and output registers, the ALU and Cell flags, and the previously mentioned External Data register. But the hardware that takes up the most space is that concerned with control. The control section is made up of the External Instruction register, the Microprogram counter logic, the Decoder, and the Microcode Memory. The amount of chip space taken up by the External Instruction register, the Microprogram counter logic, and the Decoder is not large in comparison to that taken up by the Microcode memory. The Microcode memory holds the sequence of hardware execution instructions for the 2 x 2 contouring tree creation, coordinate generation, and coordinate transformation procedures. The Microcode memory also holds the instruction sequences for the other externally initiated functions previously mentioned. Using the figures from the architectural modeling section, it is estimated that approximately 4096 16 bit words are needed to accommodate the operations necessary for all of the functions of the cellular processor. Making comparisons on a byte-by-byte basis with devices such as the MC68000 [9] or the available 64K RAMs, we estimate from two to four cellular processors on a single chip.

Without actually completing the VLSI design, there is no adequate way to estimate the amount of space it requires on a VLSI chip other than to roughly compare the hardware requirements with already existing VLSI devices. The presentation of a complete VLSI design, however, is not within the scope of this paper. We have been concerned here to examine the extent to which the presented contour display generation algorithm can benefit from the highly concurrent capabilities suggested by the VLSI technology.

4.0 CONCLUSION

This paper has presented an algorithm whose VLSI implementation can greatly increase the animation capability of current vector display systems. The algorithm for contour display generation was selected for examination because of its frequent use in computer graphics programs. Two applications have been introduced in which the need for a real-time contour display generation capability is evident. For such applications we have discussed the development of a contouring algorithm that can use the highly parallel computation capability available in the VLSI technology. In connection with this algorithm, we have examined the architectural framework for a contouring processor that performs the contouring algorithm. This discussion serves to highlight features of the contouring algorithm that are not easily recognized from its serial description. The architectural requirements of the contouring processor have been formulated with respect to the hardware necessary for its VLSI implementation. A question of the feasibility of using multiple cellular processors on a single VLSI chip has also been addressed. We have concluded from our research that the number of cellular processors per chip, though low, is sufficiently large to warrant further development.

5.0 REFERENCES

1. Aho, Alfred V., Hopcroft, John E., and Ullman, Jeffrey D. The Design and Analysis of Computer Algorithms. Reading, Massachusetts: Addison-Wesley Publishing Company, 1974, Chapters 1-5.

2. Barry, C.D. and Sucher, J. A. "Interactive Real-Time Contouring of Density Maps," ACA Winter Meeting, Honolulu, March 1979, Poster Session.

3. Clark, James H. "A System Design Revolution," Computer Graphics: A Quarterly Report of SIGGRAPH-ACM, Volume 15, Number 3 (August 1981), pp. 79-80.

4. Dammkoehler, R.A. Personal communication. October 1981.

5. Mead, Carver and Conway, Lynn, Introduction to VLSI Systems, Reading, Massachusetts: Addison-Wesley Publishing Company, 1980, Chapter 8.

6. Myers, Ware "Algorithms, Ergonomics, and Solid Modeling Highlight SIGGRAPH '81," Computer, Volume 14, Number 10 (October 1981), pp. 126-127.

7. Newman, William H. and Sproull, Robert F. Principles of Interactive Computer Graphics. Second Edition. New York: McGraw-Hill, 1979, Chapter 1.

8. Sproull, Robert "Custom VLSI Chips for Graphics," Computer Graphics: A Quarterly Report of SIGGRAPH-ACM, Volume 15, Number 3 (August 1981), p. 79.

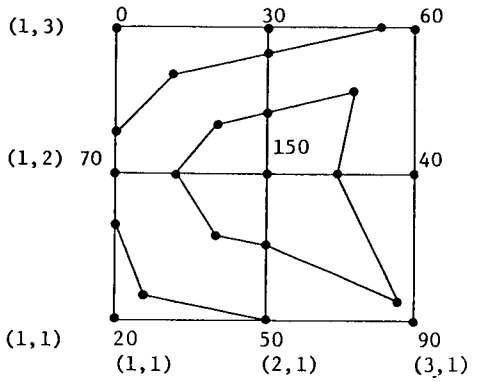
9. Stritter, Edward and Gunter, Tom "A Microprocessor Architecture for a Changing World: The Motorola 68000," Computer, Vol. 12, No. 2 (February 1979).

10. Wright, Thomas and Humbrecht, John "ISOSRF -- An Algorithm for Plotting Iso-Valued Surfaces of a Function of Three Variables," Computer Graphics: A Quarterly Report of SIGGRAPH-ACM, Volume 13, Number 2 (August 1979), pp. 182-189.

11. Zyda, Michael J. "Multiprocessor Considerations in the Design of a Real-Time Contour Display Generator," Technical Memorandum 42, St. Louis: Department of Computer Science, Washington University, October 1981.

12. Zyda, Michael J. "Joystick Driven Display Rotation and Control Console Management," Technical Memorandum 24, St. Louis: Department of Computer Science, Washington University, November 1980.

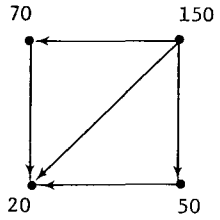
Figure 1

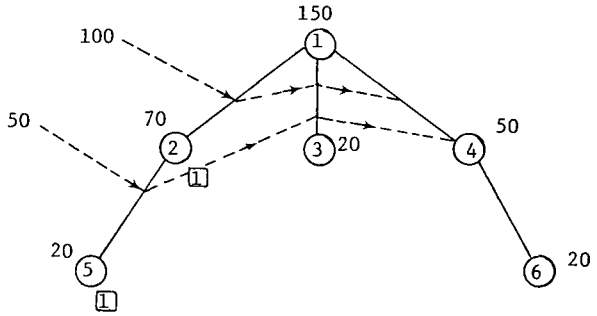


Contour Levels 50 + 100

Figure 2

Sample Contouring Tree for 2 x 2 Subgrid





Order #	Enumeration List	Next Node List	Edge Under Consideration
1	1	Done	root alone
2	2	3	1 - 2
3	5	3	2 - 5
4	3	4	1 - 3
5	4	Done	1 - 4
6	6	Done	4 - 6

Figure 3

All Possible Configurations of the Contouring Tree
Generated for a 2 x 2 Subgrid

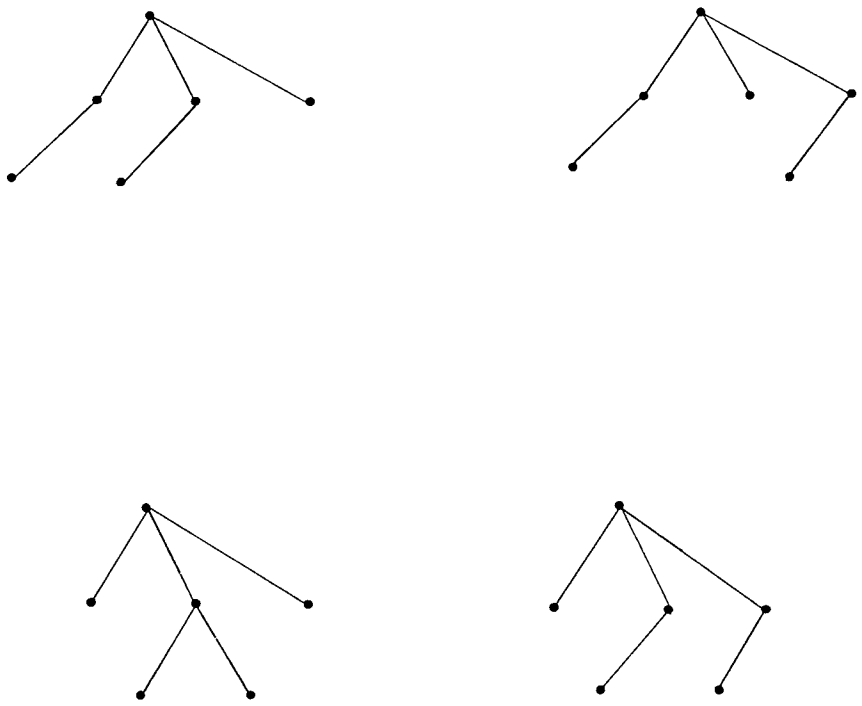


Figure 4
Schematic View of 2 x 2 Cellular Processor

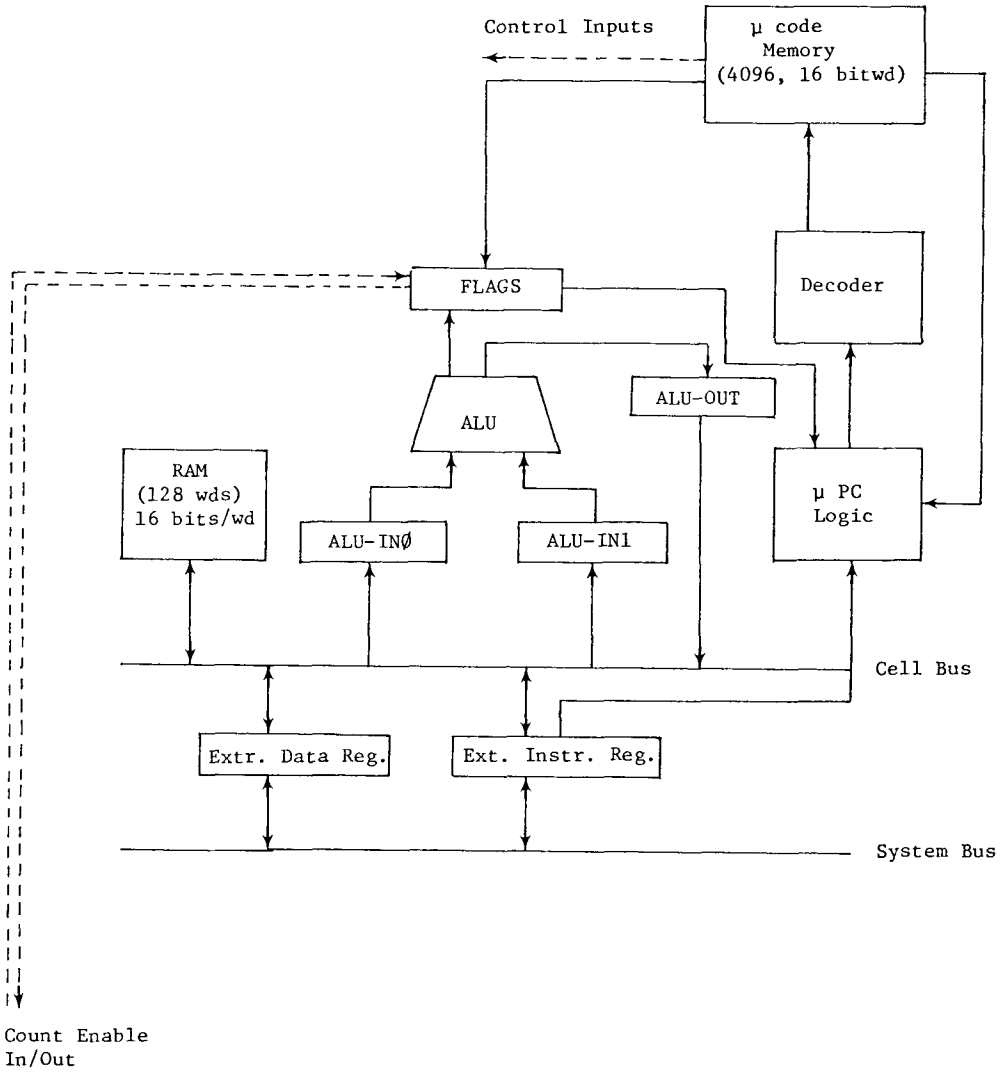
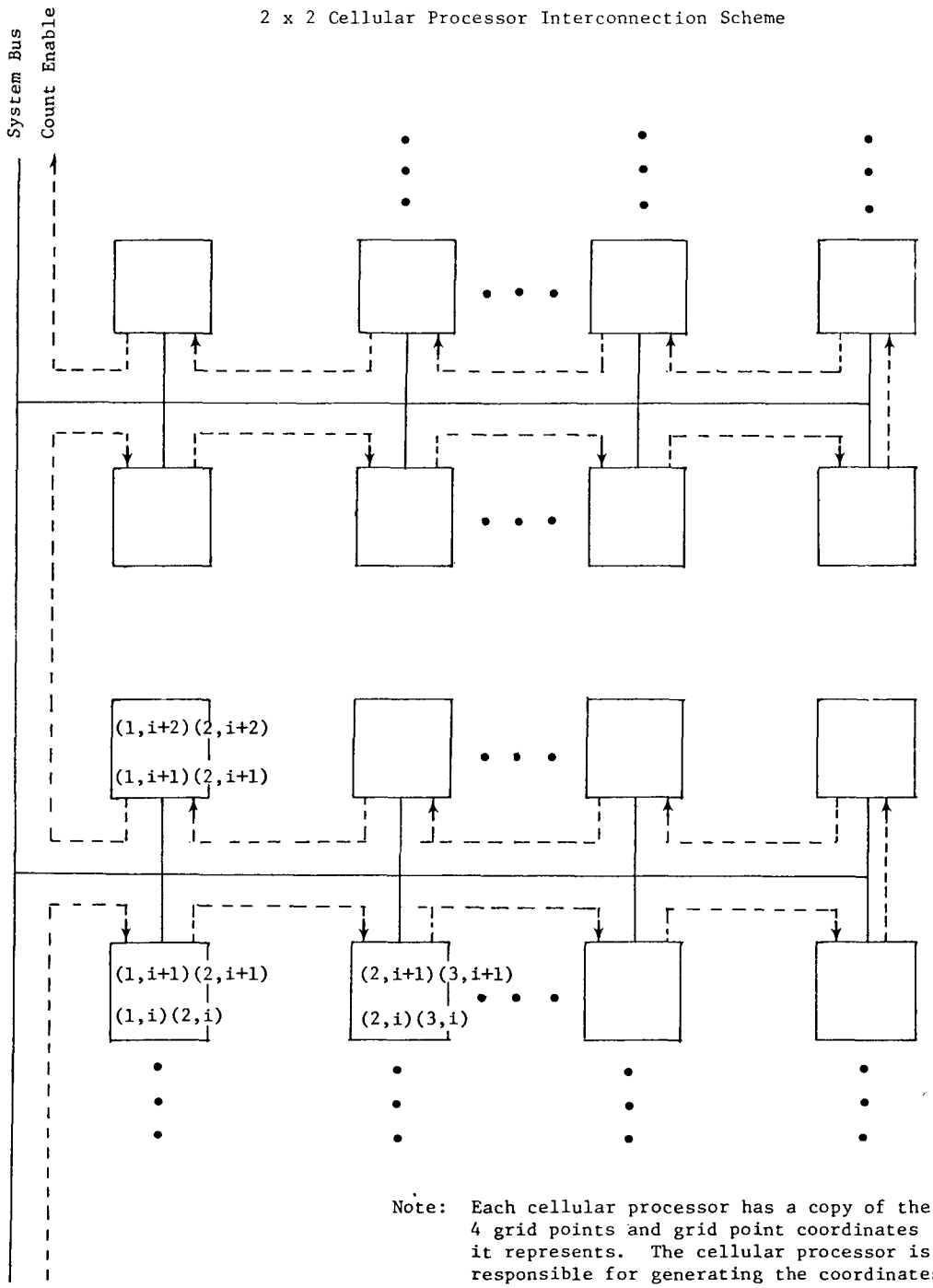


Figure 5

2 x 2 Cellular Processor Interconnection Scheme



Note: Each cellular processor has a copy of the 4 grid points and grid point coordinates it represents. The cellular processor is responsible for generating the coordinates and drawing instructions for its assigned 2 x 2 subgrid on each receipt of a new contour level.