1987

Implementing intersection calculations of the ray tracing algorithm with systolic arrays

Andrea Stankus

Rochester Institute of Technology
School of Computer Science and Technology

Implementing Intersection Calculations of the Ray Tracing Algorithm with Systolic Arrays

by

Andrea Stankus

May 13, 1987

A thesis submitted to
The Faculty of the School of Computer Science and Technology
in partial filfillment of the requirements for the degree of
Master of Science in Computer Science

Approved By:

May 13, 1987

Dr. Donald L. Kreher (advisor)

Guy Johnson

Dr. Andrew Kitchen

Title of Thesis: Implementing Intersection Calculations of the Ray Tracing Algorithm with Systolic Arrays

I Andrea Stankus hereby grant permission to Wallace Memorial Library, of RIT, to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

ACKNOWLEDGEMENT

I want to express my sincere thanks and appreciation to Professor Gerald Johnson for taking the time to review and critique this paper as it was being developed. I also want to thank my committee Guy Johnson, Andrew Kitchen and Don Kreher for helping to define the final form of this paper.

ABSTRACT

Ray tracing is one technique that has been used to synthesize realistic images with a computer. Unfortunately, this technqiue, when implemented in software, is slow and expensive. The trend in computer graphics has been toward the use of special purpose hardware, to speed up the calculations, and, hence, the generation of the synthesized image. This paper describes the design and the operation of a systolic based architecture, tailored to speed up the intersection calculations, that must be performed as a part of the ray tracing algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

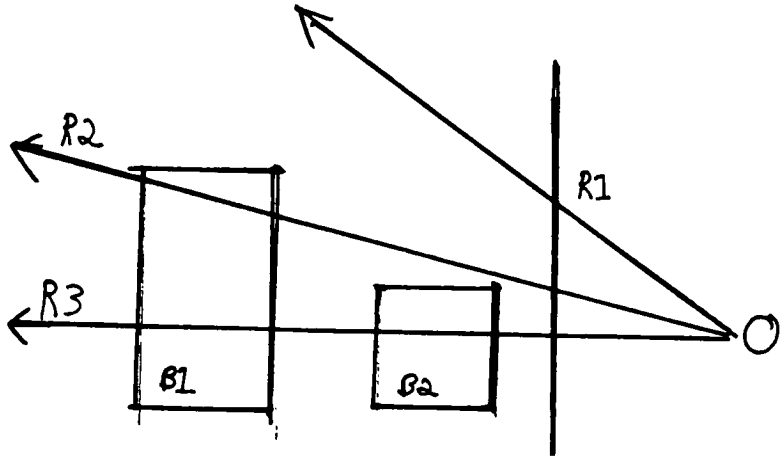
CHAPTER 1

INTRODUCTION

1.1 STATEMENT OF PROBLEM

A user of a CAD/CAM workstation implemented for the design and analysis of discrete parts, expects a realistic rendition of the part quickly. The user also expects to be able to see the results of entered changes on the updated product in the same session. A technique known as ray tracing (RAY CASTING) can be incorporated as a part of a CAD/CAM package to facilitate the realistic rendering of such a part [FLUN85]. The fundamental principle of ray tracing is to follow a ray of light reaching the observer in a direction opposite to the direction of the light ray to find any intervening objects. The intervening objects are immediate sources of light and are, therefore, visible to the viewer. This principle is used to implement ray tracing as a hidden surface removal technique. See Figure 1. If an illumination model (SOURCE OF LIGHT) is added, the amount of light received at each intersection point from each contributing surface is calculated in order to determine the intensity of the corresponding pixel [ROGE85]. Depending on the properties of the object, a ray striking the surface of the object may be splintered into subrays

that are reflected, absorbed, and/or



(Side View)

R1 - does not intersect any object in the scene; corresponding pixel is intensified to the background color.

R2 - intersects the back and front planes of block B1; the closest plane is visible and its color is used to intensify the corrsponding plane.

R3 - intersects blocks B1 and B2; since B2 is closer to the viewer, the color of its front plane is used to intensify the corrsponding pixel.

FIGURE 1: RAY TRACING USED TO IMPLEMENT HIDDEN SURFACE REMOVAL

refracted by the object. The path of each of the spawned rays is calculated and followed by the ray tracing algorithm, to determine whether these rays also intersect

a surface. Ray tracing may be further augmented by the inclusion of other visual features such as shadows, texture and transparency [ROGE85]. In addition, ray tracing may be used in conjunction with structural analysis packages as a part of a CAD/CAM system in order to determine the physical properties of a discrete part, such as, weight and center of gravity [PLUN85, WYVIL86, ROTH82].

The problem with ray tracing is that it is computationally intensive. Traditional ray tracing is a brute force technique that must calculate the intersection of each ray, starting from the eye of the viewer through each pixel involved with image synthesis with surface of each object [ROGE85]. Up to 95% or more of the time of the algorithm may be spent calculating the intersections [WHIT80]. With the addition of a global illumination model and any other visual features, not only the computational time but also the computational complexity increases. Different techniques in terms of hardware and software have been devised to reduce the computational time and cost of the algorithm: bounding boxes and spheres, extended to hierarchially related groups of objects in a scene [WHIT80, WEGH84]; efficient means of determining the point of intersection with

specific types of surfaces [SEDE84]; use of VLSI special purpose chips to calculate the points of intersection [ROGE85]; vector and parallel computer architectures [PLUN85]; extending the ray to a beam swept over polygonal surfaces [HECK84]; expanding the ray to a cone over an area of spatial coherence [AMAN84]; implementing ray tracing in microcode [HOOK84]; using a three dimensional digital differential analyzer [FUJI86].

The large number of pixels used and the independence of each pixel intensity calculation, makes ray tracing amenable to the expertise and techniques used in image processing in order to reduce the computational time and cost.

Image processing is concerned with the manipulation and extraction of information from existing pictorial data [PAVI82]. Ray tracing, on the other hand, is an example of image synthesis or picture generation under computer control. Although the two areas may appear "opposite" each other, these two areas have much in common. Both have the problem of representing the picture internally, and storing and retrieving those representations. If raster devices are used, then both image processing (IP) and computer graphics (CG) view the scene as a two dimensional array of pixel intensities,

and both would benefit from efficient manipulation of those values to generate/extract useful information. One of the hardware architectures that image processing has adopted and used successfully are systolic arrays. Systolic arrays are an efficient means for performing repeated pixel calculations of an image. This same architecture can be incorporated in the implementation of traditional ray tracing, to perform the repeated calculations for certain representations of the scene.

1.2 LITERATURE REVIEW

The technique of ray tracing was first sugguested by Appel [APPE68] and successfully implemented in MAGI, a solid modeler [GOLD71]. Ray tracing with a global illumination model was first implemented by Whitted [WHIT80] and Kay [KAY79]. A ray of light reaching the eye of the observer, and the continuation of the ray, is followed in reverse direction to determine which objects the ray intersects. If the ray intersects the objects at more than one surface point or intersects more than one object, the intersection points are sorted by their distance from the viewer, in order to find the object that is closest to the viewer. The closest intersection point is that portion of the visible surface that is

closest to the viewer. The properties of the closest surface are used to intensify the corresponding pixel. If a global illumination model is incorporated, the corresponding pixel is intensified by the amount of reflected light reaching the user from the intersection point. The intensity of the pixel is dependent on the illumination model, the position of the viewer, the surface orientation and what other paramemters the model simulates. When a global illumination model is implemented with a ray tracing algorithm, the surface and material properties of the intersected surface are taken into consideration. For example, transparent and translucent surfaces refract the light rays and allow varying degrees of visibility of any posterior objects. A solid opaque object with a matt surface is a perfect diffuser, reflecting the same amount of light in all directions. A smooth metallic surface reflects light differently in each direction and produces specular highlights. Reflected and refracted rays spawned from the intersection point are calculated and followed by the ray tracing algorithm to determine which surfaces these rays intersect and to determine their contribution towards the intensity of the corresponding pixel. These spawned rays, calculated and followed by the ray tracing algorithm, continue until no more objects are intersected. The rays

that reach the observer and pass through a pixel of the display are the rays initially followed. Traditional ray tracing implemented the ray path and the spawned rays with a tree structure [WHIT80, KAY79]. The tree is traversed by the shader, implemented in software, to determine the contribution of light at each intersection point using the attributes of the surface as well as the lighting model.

Ray tracing has generated some of the most realistic synthesized images to date [PLUN85]. The realism depends on the sophistication of the illumination model and on the resulting spectral distribution of light spread over the scene. Earlier implementations of ray tracing used global illumination models based on extensions of the Bui-tuong model for intensity calculations [BUI75, KAY79B]. Bui-tuong's model simulated diffuse surfaces of an ambient light source, a single light source located at infinity, specular highlights and reflection. Extensions to the global illumination model include simulating transparent and translucent surfaces and light within the scene [KAY79A, KAY79B]. Further improvements include limiting the area illuminated by a concentrated light source from a particular direction [BUI75]. Later, improvements were made to depict specular highlights of

rough surfaces, model multiple light sources and multi-colored light more accurately[WARN83, TORR67, BLIN78A, COOK82A]. Other illumination models increased realism by imitating a lens camera and scattered light [COOK82A, POTM82A, POTM82B, HALL82A]. These illumination models yielded different visual attributes such as refraction and transparency [KAY79A, KAY79B, HALL83], shadows [APPE68, KAY79A, KAY79B, COOK82, NEWE72, MEYE75, CROW77, VERB84, BOUC70, KELL70, WILL78, ALBE78] and specular highlights [ATHE78B, GOUR71, BUI75, WARN83, TORR67, BLIN77, ROTH82].

Different surfaces and classes of objects have been synthesized using ray tracing such as: planar polygonal and analytical surfaces [GOLD71, APPE68, KAY79A, KAY79B, WILL78, FUSS82]; bipolynomial parametric surfaces [WILL83, CATM74]; general algebraic surfaces [HANR83]; surfaces with superimposed density distributions [BLIN82]; fractal surfaces [KAY83, FOUR80, CARP78]; surfaces of simple swept objects [SEYB83]; swept spheres [WIJK84B]; swept planar cubic splines [WIJK84B]; free form surfaces described as Steiner patches [SEDE84]; free form b-spline surfaces [SWE86]; biquadratic surfaces [STEI84]; stochastic surfaces [FOUR82]; volume densities [KAJI84]; and superquadric surfaces [EDWE82]; generalized

cylinders [BRON85]. Other features or effects rendered with ray tracing are terrain [COQU84], texture [WILL83, BLIN76, CARP78], and rough surfaces [DUFF79, BLIN78].

Since ray tracing is a point sampling technique, a scene rendered using this technique is subject to aliasing [WHIT80]. The visual effects of aliasing are jagged lines that are especially noticeable with diagonal lines and with the boundaries between adjacent solid areas of different colors. One method of alleviating this problem is by recursively taking the average of four neighboring pixels of closely valued intensities or by subdividing until four such pixels are found or until the resolution of the display has been reached [WHIT80]. Another approach is to put the entire picture through a low-pass filter [WHIT80]. Other solutions to the aliasing problem include extending the ray to a beam (BUNDLE OF RAYS) intersecting polygonal surfaces by exploiting spatial coherence, and extending the ray to a cone [WILL78, COOK83]. Solving the aliasing problem has lead to the implementation of blurred motion with rays tracing [CASA85]. In addition, ray tracing has been used to render terrain data, and has been incorporated into solid modelers, especially for constructive solid modelers (CSG)[GOLD71, DUFF79, MAXW83, CLAR76, DAVI82, ROTH82,

WYVI86]. As part of a CAD system ray tracing in the solid modeler is used to calculate the phyiscal properties of objects, such as mass, volume and center of gravity in order to facilitate finite element analysis.

Ray tracing itself, and especially with the incorporation of shaders for realistc rendering, is computationally intensive and expensive. The cost being due to calculating the intersections between rays and surfaces. Different solutions in terms of software, hardware and firmware have been devised to reduce the computational cost of executing some form of the ray tracing algorithm. Software solutions have looked for ways of reducing the number of intersection calculations. One solution is to encompass the object by a bounding box or sphere. If the ray does not intersect the bounding volume then the object is ignored for this ray; otherwise, the intersection calculation with bounding volume is performed [WHIT80]. This idea has been extended to groups of objects and to hierarchies of bounding volumes for an entire scene [RUI80, WEGH84, GLAS84]. Other techniques include subdividing the objects space as a hierarchy of cubes or octants, called an octree, extending the ray to a beam, extending the ray to a cone, using cellular decomposition, using an adaptive tree with

depth control [JONE71, KAJI82, COOK83, HALL83, ULLN83], and using a directed acyclic graph (DAG) to represent the solid [WYVI86]. A ray tracing system that uses a three dimensional digital differentail analyzer (3DDDA) has been implemented [FUJI86]. Hardware solutions that reduce the computational time by increased processor power and speed have been proposed and/or implemented. For example, a multi-processor architecture that can be reconfigured to handle subsets of the objects, or subsets of the rays, has been sugguested [DIPP84]. Another proposed solution is to decompose the image into subregions that map onto a three dimensional array of independent processors [NISH83]. Special purpose hardware for ray traced animation has been implemented [CLEA83]. Other multiprocessor architectures have been proposed [WEIN81, FUCH81]. A number of VLSI based architectures have been discussed in order to reduce computation time [ROGE85, PLUN85, FUCH81]. The implementation of the ray tracing algorithm on a parallel array architecture required a reanalysis of the algorithm [PLUN85].

1.3 RAY TRACING IMPLEMENATION TECHNIQUES

A more detailed description of the basic ray tracing algorithm is given in detail in the main body of this text. After that, a hardware implementation technique

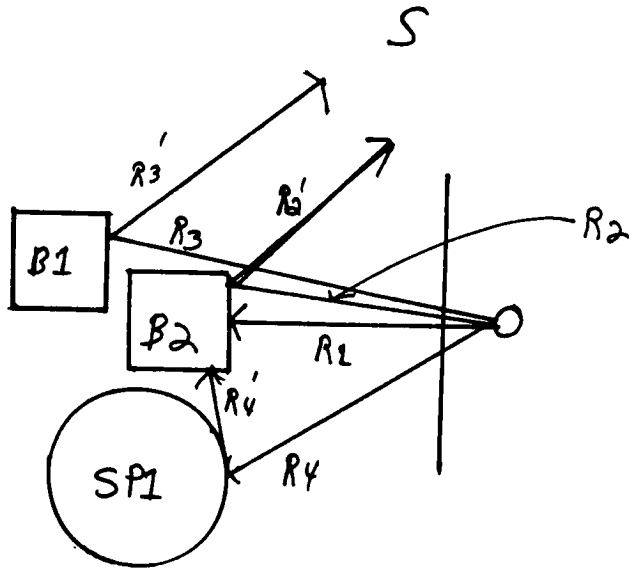
used by image processing (IP) to reduce computational cost and time, is discussed. This technique comes from VLSI technology, and is applied to the basic ray tracing algorithm. An analysis to determine the effectiveness of this technique is followed by a discussion describing the limitations of applying the IP technique to the ray tracing algorithm.

CHAPTER 2

IMPLEMENTING RAY TRACING WITH TECHNIQUES FROM IMAGE PROCESSING

2.1 DETAILED DESCRIPTION OF OPERATION OF RAY TRACING ALGORITHM

Ray tracing is the only hidden surface removal algorithm that allows the use of a global illumination model for realistic synthesis of images [WHIT80, ROGE85]. The appropriateness of ray tracing with the use of a global illumination model can be demonstrated by an example of a situation that can be encountered during image synthesis. A scene includes a solid object that partially obscures a metallic surface from the viewer. In order for this scene to be rendered realistically, a portion of the hidden surface of the solid object must be partially reflected on the metallic surface behind the the solid object. Rays deflecting from the hidden surface of the solid must be followed to see where, and if, they intersect the metallic surface. This is shown in Figure 2.



(Top View)

R1 - angle of inicidence is 90 degrees; if the surface was metallic or shiny, specular highlights could be modeled for the correspnding pixel

R2 - a visible point on the front plane of block, B2, is intersected by a R2;no other object produces a reflected component of light that reaches the intersected visible point. The reflection ray leaves the scene and need not be traced any further.

R3 - this ray intersects the visible point on the front plane of block,B1. The single source of light(S), contributes directly to the intensity of the visible surface point.

R4 - the intersected surface point receives intensity contributions from a nonvisible plane from block, B2. If the surface of the sphere(SP1) is metallic and/or shiny, the nonvisible surface of B2 will be seen reflected on a portion of the visible surface of the sphere.

FIGURE 2: RAY TRACING WITH A GLOBAL ILLUMINATION MODEL

The global illumination model, using the intersection information from the ray tracing algorithm, determines the intensity of the corresponding pixels. Ray tracing can support a variety of shading models and a variety of geometrical descriptions of the objects in the scene.

In traditional ray tracing, for each ray reaching the viewer and intersecting a pixel on the screen (SCANNING LEFT TO RIGHT AND TOP TO BOTTOM), a tree is built. The nodes of the tree represent the ray-surface intersections,and the branches represent any spawned, reflected and/or refracted rays that might be modeled in the shader. The intensity of a pixel is dependent on the intensity of light transmitted from the corresponding point on the visible surface of an object to the viewer. Among the contributing components that are modeled by the illumination model, are the reflected and refracted light rays that reach that point. Contributions of these light rays are based on the laws of optics [WHIT80, ROGE85]. The refraction and reflection rays are traced to their respective sources. These sources can be other surfaces, or the light source (S), and are assumed to be point sources. If these sources are other surfaces then the contributions from other reflection and refraction rays

to those point surfaces are also traced. This tracing of the two light ray types continues until the light source is reached, or until the rays follow paths out of the scene. These rays are traced backwards to the light source (S) from the viewer. The shader then traverses the tree from the leaves to the root using the contributions from the reflected and refracted surfaces, to determine the intensity of the pixel. The calculations performed, and the information maintained in the data structures depend on the illumination model being implemented. Regardless of the illumination model used, the ray tracing algorithm must be able to extract the geometrical information from the internal representation of the objects, and determine the points of intersection. Different internal representations include approximation by polygons. The polygon is represented by an implicit form of a planar equation. In many implementations the ray is represented as a parametric equation. The two points on the ray are the position of the viewer (P1) and the pixel of current interest (P2). The ray can be defined as thus:

$$R = P1 + (P2 - P1)T$$

$$RX = P1X + AT \qquad A = P2X - P1X$$
$$RY = P1Y + BT \qquad B = P2Y - P1Y$$
$$RZ = P1Z + CT \qquad C = P2Z - P1Z$$

The values for t are non-negative. This anchors the ray at the viewer. Values less than zero are behind the viewer and indicate intersection with objects not in the scene. The parametric equations for x,y,z (RX,RY,RZ) are substituted into the implicit form of the planar equation, and then solved for t. The value for t is substituted into the parametric equations for RX,RY,RZ to determine the actual intersection point. To make sure that the intersection is within the bounded plane, a bounds test is performed. This same technique can be applied to surfaces defined by quadric equations. Here, the quadratic formula is used to solve for t. A complex root indicates no intersection. Reduction in time to do intersection calculations with traditional ray tracing may be achieved by: 1- reducing the number of intersection calculations-- a software solution; or 2- using special-purpose hardware to shorten the time in performing the intersection calculations [GLAS84].

2.2 INTRODUCTION TO SYSTOLIC ARRAYS

Ray tracing, like many image processing operations performed on pictures, is computationally intensive. Calculations must be performed for each pixel. Special purpose architectures have been built to expedite the IP calculations, such as convolution, filtering, and image

enhancement techniques [FU--82]. Some of these special purpose architectures are described briefly here. The CLIP4 is 96 x 96 array of parallel processors that perform local neighborhood calculations, such as shrink and expand. The processors can be reconfigured by different types of neighbor connections (I.E. 6-CONNECTED OR 8-CONNECTED). The mpp (MASSIVELY PARALLEL PROCESSOR), built by Goodyear Aerospace Co. for NASA ,consists of 256 processing elements, which may contain up to 32 processing units. The MPP was designed to perform IP functions, such as feature extraction, pattern recognition, and relaxation. The ILLIAC IV is a SIMD architecture with one central control unit, and 64 processing units, and was designed to handle a range of problems. The IP applications include clustering, pattern classification and texture analysis. Picap II is a MIMD architecture that allows up to 64 x 64 neighborhoods in performing pattern recognition calculations on multiple images. Measurements are automatically performed on a field of data that is 512 x 640 to reduce the input. The measurements include histogram collection, gray scale range determination, and x-y extent determination. In addition, there are dedicated processors for performing filtering, generating graphical overlays, and for labeling and template matching [FU--82]. Also, extensions

to existing programming languages have been implemented to reflect these new architectures, and to facilitate the development of algorithms to take advantage of the new architectures. For the CLIP4 array processor, an extension of Pascal, i. e. Pascalpl, has been implemented [UHR82]. Applications that have been implemented using this language include performing calculations in IP, pattern recognition and scene description. Constructs of the language include parallel assignment, condition, and i/o statements as well as parallel proceedures, allowing for the simultaneous processing of array elements, as well as sets of arrays. The data types of these arrays are binary, gray scale, or nonnumeric. Also, the capability exists to access relative locations in a two dimensional array in order to perform operations, such as, windowing and masking [DUFF8X1, UHR84]. PPL (PICTURE PROCESSING LANGAUGE) developed for the Picap I and Picap II is also an extension of Pascal. Features of the language to facilitate the image processing computations include binary picture, picture, and template element as data types. Picture operators include parallel arithmetic operations on 3x3 neighborhoods [DUFF81]. One of the languages includes Pixal (PIXEL MANIPULATION ALGORITHMIC LANGUAGE), also targeted for the CLIP4, is an extension of Algol60. Data types specific to IP include grey,

binary, mask and frame. Control structures tailored for IP include parallel assignment statements, parallel comparisons between masks, and submatricies of a two dimensional array, and parallel arithmetic operations. Pixal has been used to implement parallel thinning and object extraction algorithms [DUFF81, FU82].

IP has recently found help for its calculations from the VLSI (VERY LARGE SCALE INTEGRATION) technology in the form of systolic based architectures [KUNG82]. Currently, VLSI on a single chip is a two dimensional technology, that is, architectures are designed and laid out in a two dimensional grid. Problems arise when there is too much crossover of communications paths between the cells [JALA85]. To reduce crosstalk it is neccessary to have short and regular paths on the chip, in order to reduce the initial design cost and delays during execution. These two dimensional architectures are easier and cheaper to manufacture when the cells/components of the chip are of regular shapes with regular interconnections. The arrays may be one dimensional or two dimensional, depending on the algorithm to be implemented.

Systolic architectures exploit these constraints to make algorithms that can also exploit these constraints to execute more efficiently. The systolic architecture is

a specialized architecture that embeds much of the algorithm in the hardware. Applications for systolic architectures have included IP operations, matrix arithmetic operations, data base manipulations, speech recognition systems, vision, and stero vision systems [KUNG82, MEAD80, FRIS85, GUER85, SYMA85]. The design principles of systolic architectures are described below. A single cell is replaced by an array of cells [KUNG82]. This allows the throughput to be increased without increasing the memory bandwidth. One of the goals of systolic architectures is to decompose complex problems into simple and regular pieces that can map onto regular, simple, and relatively fast cells. These cells perform the specialized calculations, and have the same design. The throughput of the system is then improved by also implementing concurrency to the extent permitted by the algorithm. In a systolic architecture, this is accomplished by "pumping" the data through the cells. Once an input value has been used in a calculation, the data is passed on to the next cell. The next cell is algorithm dependent. This circulating of data imposes a regular flow of data throughout the system, and enables the simultaneous execution of many cells. The regular communication paths for the data reduce the overhead incurred in interprocessor communications. Another goal

of the systolic architecture is to balance the computational bandwidth with the I/O bandwidth. Once a data item has entered the array, the data is reused. The data are input to the boundary cells of the array [KUNG82]. Therefore, the criteria used to determine whether an algorithm is amenable to execution on a systolic based architecture appear to be the following:

1- the algorithm is compute bound rather than I/O bound (.I.E,THE NUMBER OF OPERATIONS EXCEED THE NUMBER OF I/O DATA ELEMENTS);

2- the algorithm must execute in real-time (I.E.,WHILE THE USER VIEWS THE IMAGE ON THE SCREEN);

3- the calculation are regular and can be defined with simple recurrence relations;

4- the design of the components on the chip to implement the algorithm are regular (I.E.,MINIMIZE CROSSTALK AND DESIGN COSTS) [JALA85].

2.3 APPLICATION OF SYSTOLIC ARCHITECTURES FOR RAY TRACING

2.3.1 RAY TRACING AS A HIDDEN SURFACE REMOVAL TECHNIQUE

2.3.1.1 POLYGON APPROXIMATION

The ray tracing algorithm can be adapted to take advantage of the VLSI systolic technology. Starting with ray tracing as a visible surface algorithm, and with the solid objects in a scene approximated with polygons, the following analysis will show that the execution time of

the traditional ray tracing algorithm can be reduced by adapting the algorithm to, and by, running the algorithm, on, a systolic based architecture. The ray tracing algorithm meets the criteria stated in the previous section (SECTION 1.2). The ray tracing algorithm is certainly compute bound with each pixel ray being compared to each polygon in the scene. Back plane culling could be used to remove those polygons that are obviously hidden from the viewer. The picture must be generated in real-time, especially when ray tracing is a part of a solids modeler of a CAD/CAM workstation. Although there are no recurrence relations, the same calculations are done repeatedly for each pixel in the screen, for each plane in the scene. For ray-plane intersections, the calculations are straightforward. Since the cells must perform the same calculations, and the pixels are scanned in an incremental order, the design of the cells should be the same. The interconnections between the cells should be linear rather than diagonal, vertical, or horizontal as in IP. This type of connection is certainly regular. The number of pixels that need to be intensified or shaded is proportional to an area on the surface of the graphical display device. The area is normally rectangular. The complexity of the scene and of the objects would determine the number of polygons needed to

approximate the objects. The systolic architecture to implement the ray tracing algorithm is based on the calculations that must be performed. The calculations that must be performed are the intersections of a directed line segment with planes. The general form of a planar equation is:

$$AX + BY + CZ + D = 0$$

The parametric form of the x,y,z components of the ray are: (VIEWER POSITION DENOTED BY XO,YO,ZO)

$$X(T) = XO + ET$$
$$Y(T) = YO + FT$$
$$Z(T) = ZO + GT$$

The right side of the equations are substituted into the plane equation:

$$A(XO + ET) + B(YO + FT) + C(ZO + GT) + D = 0$$

Then the equation is solved for t:

$$t = -(AXO + BYO + CZO + D) / AE + BF + CG$$

WHERE: AE + BF + CG is NOT EQUAL TO ZERO

The value of t can be checked to make sure that it is positive, and that the point of intersection is in front of the viewer. If positive, the value of t is substituted back into the parametric equations for the x,y,z components of the ray. A rectangular bounds test is

performed for each bounding plane to make sure that the intersection point is within the bounded planes that are part of the object. The following set of equations must be simultaneously satisfied.

```
XMIN <= X <= XMAX
YMIN <= Y <= YMAX
ZMIN <= Z <= ZMAX
```

For visibility calculations, the pixel can be intensified to the color of the polygon, or to the background, if there are no valid intersections. The colors available are dependent on the capabilities of the graphics processor in the display terminal. Normally, some method is provided to specify the varying intensities of red, green and blue to generate the desired color for the pixel. The data for calculating t, the coefficients of the plane, the position of the viewer, and the current pixel to be intensified, are readily available. A systolic array of linearly connected cells that can support the calculations is shown in Figure 3. The number of cells (M) is equal to the number of polygons in the scene. The inputs and the outputs of the cells are the pixel coordinates, the line parameters of the ray, a t parameter value, and a polygon identifier. At the start of the intersection calculation, the first pixel coordinate and its associated input are

used to calculate the t for the cell. If the calculated t value is negative, the value is ignored, and the input t value, the input polygon identifier, the pixel coordinate, and the ray line parameters are sent to the next cell. If the calculated t value is the same as the input t value, the calculated t value is ignored, and the input t value and polygon identifier is sent to the next cell. If the calculated t value is positive and less than the input t value, the calculated t value and the polygon identifier associated with the cell is sent to the next cell. The passing and the comparing of the t value at each cell has the effect of sorting the t values, and of selecting the smallest positive parameter value, if one exists. When the first pixel value is output from the last cell in the array, the output from that cell will be the t value, the associated polygon identifier, the pixel coordinate, and the ray line parameters -- all the information needed to calculate the actual point on the polygon surface, except for the perspective point. The polygon identifier can be used to index a lookup table to retrieve the corresponding x,y,z extents. The coordinates of the calculated point can be compared with the appropriate extent to determine whether the point is within the polygon in the scene. All these calculations, the point determination, and the comparison of the

coordinates with the extents, can be performed by a separate, single cell. Depicted as S in Figure 3. The output of this cell is the intersection point and the polygon identifier, or a not valid intersection indicator. This information is circulated through the systolic array, or sent directly to the interface buffer. Figure 3 depicts a high level representation of the supporting architecture. The operations performed indicate a need for local memory to store intermediate results.

The architecture of a cell is based on the assembly language level-like calculations found in Appendix D. Figure 4 shows the architecture designed to support the above calculations.

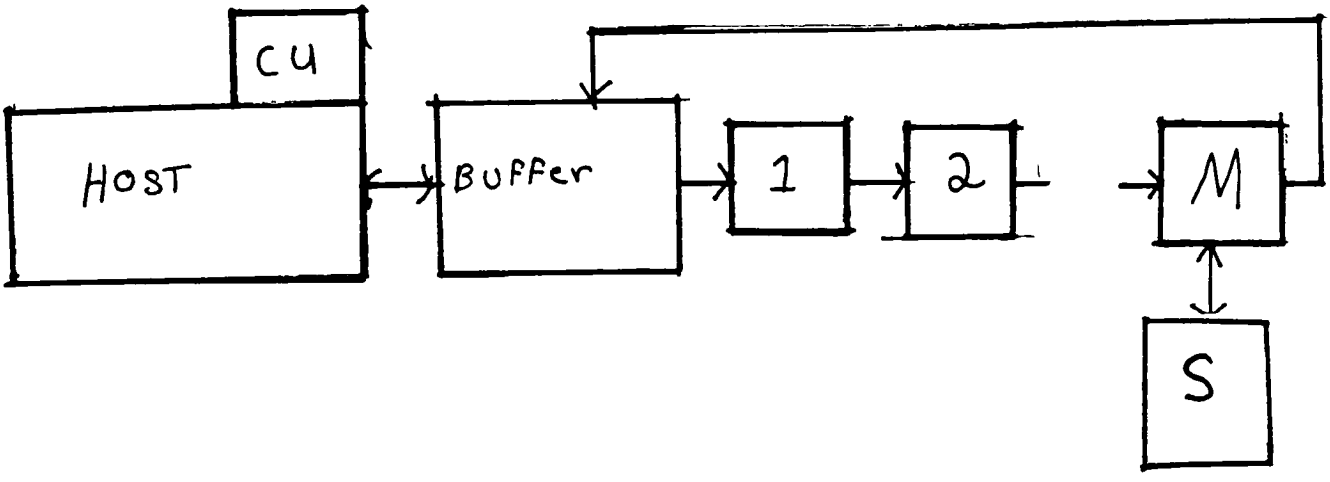


FIGURE 3: SYSTOLIC BASED ARCHITECTURE FOR IMPLEMENTING RAY TRACING AS A HIDDEN SURFACE REMOVAL ALGORITHM

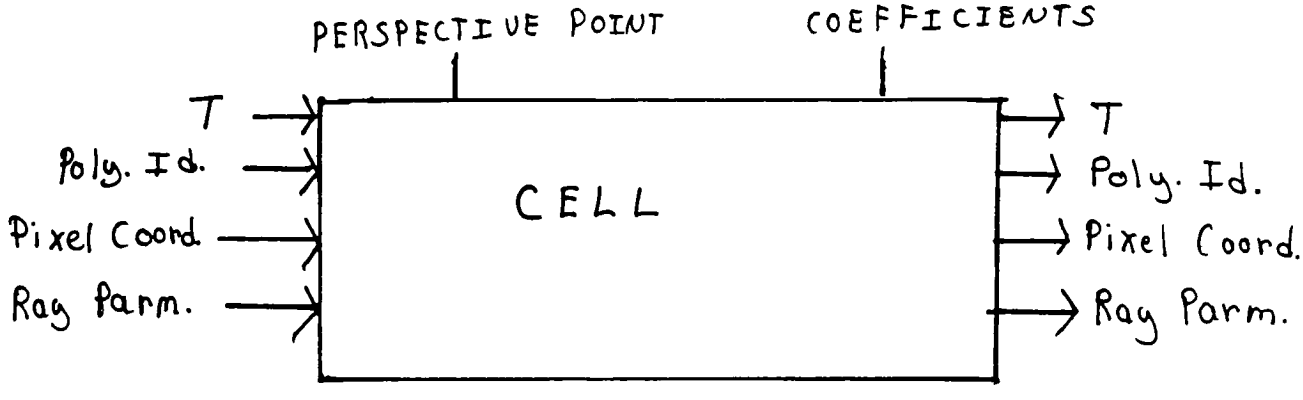


FIGURE 4: HIGH LEVEL DESCRIPTION OF SINGLE SYSTOLIC CELL

The architecture of the separate single cell(S) is used to support the assembly language level-like calculations

found in Appendix D. Figure 5 shows the architecture of the single separate cell.

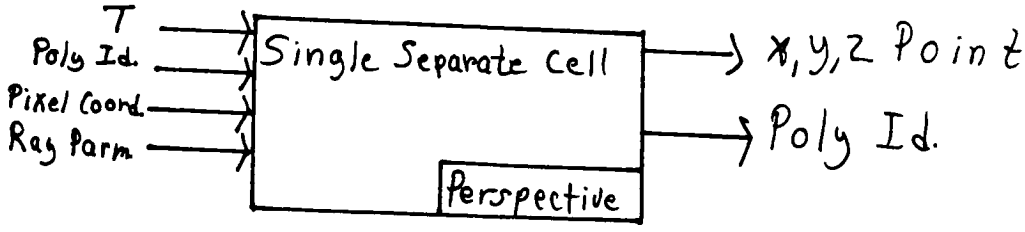


FIGURE 5: HIGH LEVEL DESCRIPTION OF A SEPARATE CELL PROCESSOR

One consequence of using the systolic based architecture is the necessary mapping of the algorithm onto this hardware. A high level description of the algorithm is given here:

```
PICK A PERSPECTIVE POINT
FOR EACH PIXEL IN THE VERTCAL DIRECTION
  FOR EACH PIXEL IN THE HORIZONTAL DIRECTION
    FOR EACH POLYGON IN THE SCENE
      DETERMINE RAY PARAMETERS
      CALCULATE INTERSECTION POINT
    END FOR
  END FOR
END FOR
```

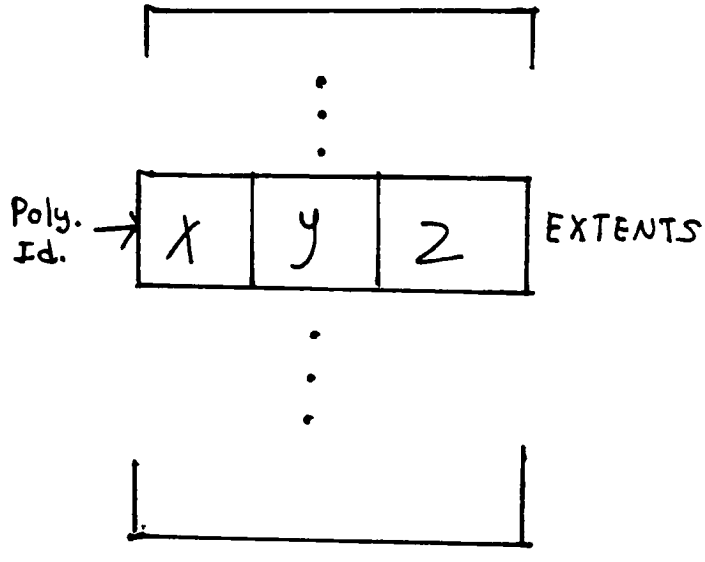
The language in which this algorithm is implemented must be enhanced, or designed, to accommodate the new constructs needed to indicate to the compiler the type of

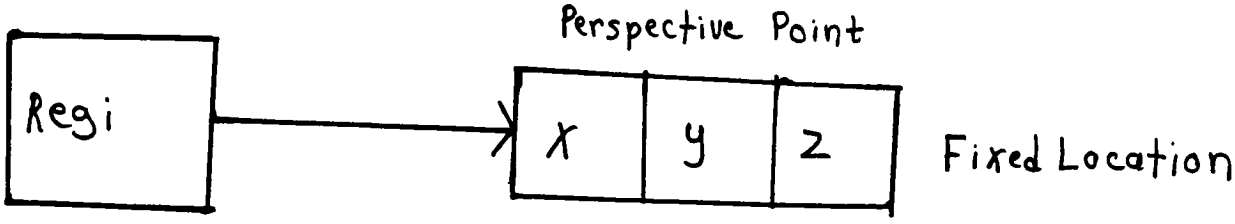
code that must be generated. This implies that the compiler designer must be knowledgeable about the underlying auxiliary systolic architecture. The scanning of the display left-to-right, top-to-bottom, suggests a data type in the language that is automatically accessed in the same manner, and is rectangularly shaped. As this data structure is scanned, the pixel coordinates can be generated with the ray parameters, and sent to the systolic array. Normally, a special purpose architecture, like the systolic arrays, is connected to a traditional, general purpose, host computer. Based on the above algorithm, and on the calculations that must be performed, the parameters that must be made available at the time the systolic array is ready to perform the intersection calculations are shown here:

- STARTING PIXEL LOCATION
- NUMBER OF PIXELS IN THE HORIZONTAL AND VERTICAL DIRECTIONS
- POINT OF VIEW
- POLYGON IDENTIFIER
- COEFFICIENTS OF PLANE EQUATION

The last item in the list will depend on the internal description of the objects in the scene. The internal description is dependent on the mathematical functions used to depict the scene objects. For the polygon planes, preprocessing includes the extraction of

the planar coefficients, the x,y,z extents, and the polygon identifiers quickly from the internal data structure. The interface between the host and the systolic array is a memory buffer with separate input and output ports, between the host and the buffer, and the array and the buffer. Before the calculations begin, the planar coefficients must be preloaded into the cells, as well as, the x,y,z extents, and the perspective point into the lookup table, and the single separate cell, respectively. An entry of the lookup table is as shown here:





The format of the data stream coming from the buffer would be:

A1B1C1D1X1Y1Z1POLY1....AMBMCMDMXMYMZMPOLYM*

Where the '*' is an end-of-datastream indicator. The instructions to be executed should also be preloaded [UME085]. The format of the data stream should be:

I1+I2+I3+...+IK

Where:
k - number of instructions
+ - an interinstruction indicator
# - end-of-instruction stream indicator

These instructions could be sent through the cells like the data, or be broadcast by the host directly to the cells. The microcode should be extremely fast. The instructions need only be loaded once. The idea of loading instructions suggests how systolic arrays can be tailored to accommodate different surface-ray calculations. This would also require the preloading and circulating of different data values. Once the necessary information has been preloaded, the pixel values and the

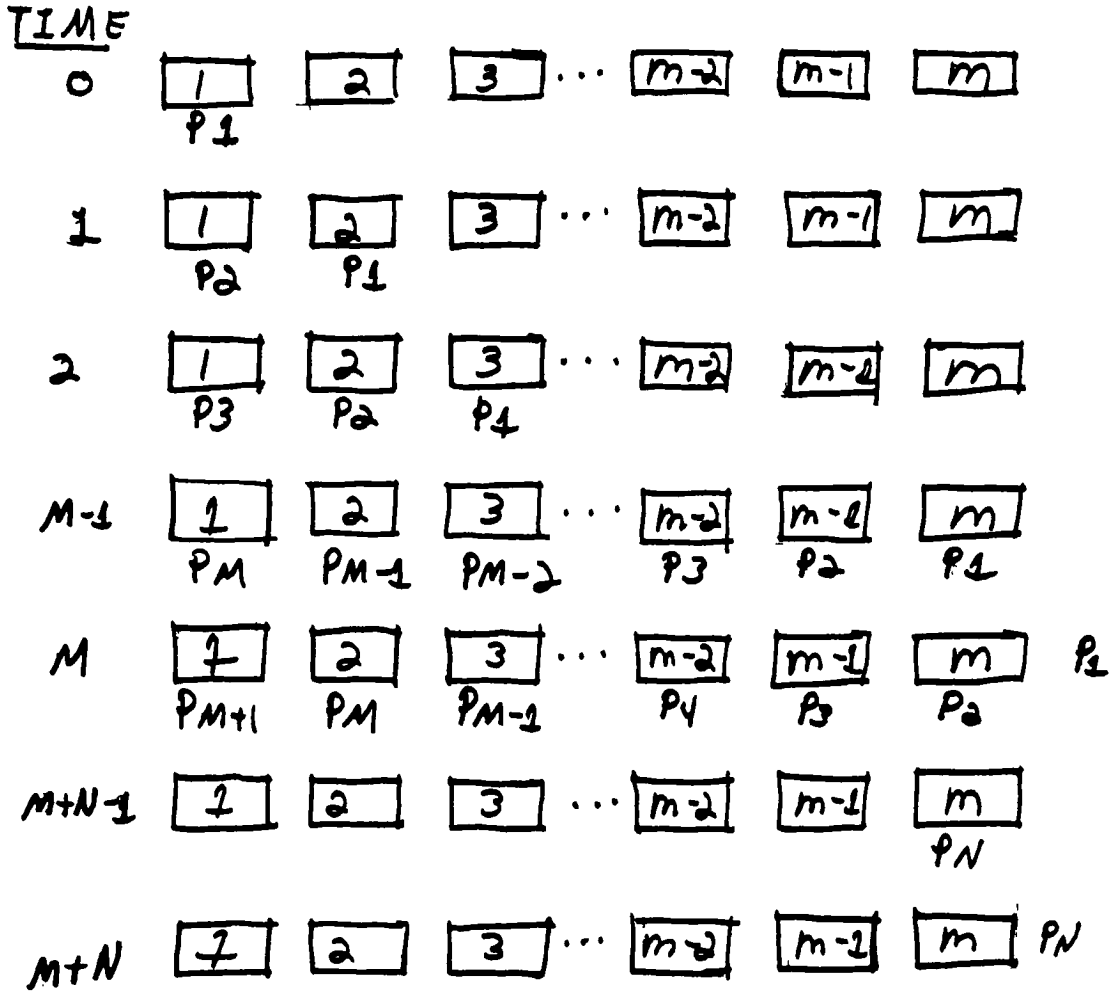
corresponding ray paramemters must be generated, and sent through the systolic array. For hidden surface removal, the pixel parameters need only be sent through the systolic array once. There will be an initial delay until the first t value is output from the last cell. Then, the array should output t values at regular intervals.

2.3.1.1.1 ANALYSIS OF THE USE OF SYSTOLIC ARRAYS

The basic unit of time (B) is the time needed for the cell to compute the t value and to compare the computed t value with the input t value. If there are M cells, the time that must elapse before the first t value is available for further calculations is M*B time units. Every subsequent B unit of time for the remaining number of pixels (N - 1), should yield another t value. The total time required to compute all the t values is B*M + B*(N-1). This is shown schematically in Figure 6.

If the algorithm is executed on a single processor computer, the time needed to calculate the intersections is B*N*M. Additional time is needed to determine the point, and make the extent comparions before the point is available to the remainder of the algorithm. Ideally, this additional time is B units in duration, otherwise the output of the separate single processor is delayed to

the next multiple of B. The delay requires the queuing of the pixel coordinates, the ray parameters, the t value, and the polygon identifier. The number of entries in the queue is proportional to the number of B units the separate single cell uses.



PK - INPUT ASSOCIATED WITH PIXEL PK
TL - OUTPUT T VALUE AT TIME L*B

FIGURE 6: EXAMPLE OF THE TIMING OF THE INTERSECTION CALCULATIONS WITH A SYSTOLIC ARRAY

2.3.1.2 QUADRIC SURFACES

Quadric surfaces are exemplified by the conic sections (I.E.,CIRCLE, ELLIPSE, PARABOLA, AND HYPERBOLA) extended to three dimensions. A quadric surface is of the form:

```
DX**2 + BY**2 + FZ**2 + GXY + HXZ + IYZ + YX +

KY + LZ + M = 0 (1)
```

This expression is also an implicit equation, in which the parametric components of the ray can be substituted for x,y,z. Recalling the parametric form of the ray :

```
X(T) = XO + A*T
Y(T) = YO + B*T
Z(T) = ZO + C*T
```

Substituting into equation (1):

```
D(XO + A*T)**2 + E*(YO + B*T)**2 + F*(ZO + C*T)**2  +
G*(XO + A*T)*(YO + B*T) + H*(XO + A*T)*(ZO + C*T) +
```

```
I*(YO + A*T)*(ZO + C*T) +
J*(XO + A*T) + K*(YO + B*T) + L*(ZO + C*T) + M = 0
```

Expanding:

```
D*(YO**2 + A**2T**2 + 2*XOA*T) +
E*(YO**2 + B**2T**2 + 2*YO*B*T) +
F*(ZO**2 + C**2T**2  + 2*ZO*C*T) +
G*(XO*YO + T*(XO*B + YO*A) + A*B*T**2) +
H*(XO*YO + T*(XO*C + ZO*A) + A*C*T**2) +
I*(YO*ZO + T*(YO*C + ZO*B) + B*C*T**2) +
J*(XO + A*T) + K*(YO + B*T) + L*(ZO + C*T) + M = 0
```

And collecting like powers of t yields:

```
T**2(D*A**2 + E*B**2 + F*C**2 + G*A*B + H*A*C + I*B*C ) +
T*(2*D*A*XO + 2*E*B*YO + 2*F*C*ZO + G*(XO*B + YO*A) +
H*(XO*C + ZO*A) + I*(YO*C +ZO*B) +
J*A + K*B + L*C ) +
(D*XO**2 + E*YO**2 + F*ZO**2 + G*XO*YO + H*XO*ZO + I*YO*ZO +
J*XO + K*YO + L*ZO + M )
= 0   (3)
```

Equation (3) is in the form:

```
A'*T**2 + B *T + C  = 0;
```

The values of t can be solved for by using the quadratic formula:

$$T = (-B' +/- (B'**2 - 4*A*C')**1/2) / 2*A$$

The coefficients are based on the perspective point, and the ray parameters and coefficients of the quadric surface. Complex values of t indicate no intersection with the surface, and can be ignored. Again, negative t values can also be ignored, because the intersection point is not within the scene. Since there are two t values generated, the smaller positive t value is chosen. The t value is put back into the ray component equation to determine the point on the surface. A bounds test is made to ensure that the point is within the sight of the viewer.

The architecture supporting the intersection calculations is a one dimensional array of connected cells, one cell for each surface in the scene. The systolic array communicates with the host via an interface buffer. Preprocessing prior to the intersection calculations include generating the pixel coordinates, and computing the ray parameter and surface coefficients. The pixel coordinates are generated a second time when the coordinates are sent through the array. The inputs and outputs to each cell include only a pixel coordinate,

its ray parameters, a t value, and a surface identifier. The two t values are generated through the quadratic formula. Each cell performs the assembly langauge level-like operations found in Appendix D.

The supporting architecture is shown in Figure 3. The determination of which t value is sent to the next cell is shown in the high level algorithm here:

```
T1,T2 -- CALCULATED T VALUES
TIN --     INPUT T VALUE

IF T1 AND T2 ARE COMPLEX OR T1 AND T2 ARE BOTH <= 0
THEN
   OUTPUT(TIN)

IF TIN IS VALID
THEN
   IF T1 > 0 OR T2 > 0
   THEN
      OUTPUT(MINIMUM(T,TIN))
      IF  T1 AND T2 ARE BOTH > 0
      THEN
         OUTPUT(MINIMUM(T1,T2,TIN))
ELSE
   TIN IS INVALID
   IF T1 > 0 OR  T2 > 0
   THEN
      OUTPUT(T)
   IF T1 AND T2 ARE BOTH > 0
   THEN
      OUTPUT(MINIMUM(T1,T2))
END IF
```

The output from the last cell includes the smallest positive t value--if there is one--the corresponding identifier, the pixel coordinates, and the ray

parameters. The separate single cell performs the same basic calculations as for polygon approximation. The t value is substituted into the parametric ray equations to determine the point on the surface. the surface identifier is used to index a lookup table to retrieve the x,y,z extents. The x,y,z coordinates are compared with the extents. The output of the separate single cell is an intersection point and a surface identifier, or an intersection point and an invalid surface indicator. Again,the output can be circulated through the cell, or sent directly to the interface buffer. The algorithm that must be mapped onto the systolic array is shown below:

```
PICK A PERSPECTIVE POINT
FOR THE NUMBER OF PIXELS IN THE VERTICAL
                                  DIRECTION
  FOR THE NUMBER OF PIXELS IN THE HORIZONTAL
                                    DIRECTION
    DETERMINE THE RAY PARAMETERS
    FOR EACH SURFACE IN THE SCENE
      PERFORM RAY-SURFACE INTERSECTION
                          CALCULATION
    END FOR
    DETERMINE SURFACE POINT
    COMPARE WITH SURFACE EXTENTS
  END FOR
END FOR
```

The algorithm assumes a homogeneous scene; that is, all the objects in the scene are represented as quadric surfaces. The information that must be available at the

start of the intersection calculations is similar to the information that must be available for polygon approximation of a scene. The list of information is:

- STARTING PIXEL COORDINATE
- NUMBER OF PIXELS IN THE HORIZONTAL AND VERTICAL DIRECTIONS
- PERSPECTIVE POINT
- INITIAL SURFACE IDENTIFIER AND T VALUE
- COEFFICIENTS OF THE QUADRIC SURFACE

The initial surface identifier is an invalid one, and the t value is negative. These initial values are implementation dependent, and indicate that the ray does not intersect any object in the scene. Once the ray has circulated through the systolic array, the surface identifier and t value will either still indicate that the ray does not intersect any object in the scene, or the closest intersecting surface and the parametric intersecting t value. The instructions to perform the intersection calculations are loaded during the preprocessing of the surface coefficients, using the same format as described for polygon approximation of a scene. The coefficients are broadcast to the appropriate cells. Again, there will be an initial delay until the first t value is available.

2.3.1.2.1 ANALYSIS OF USE OF THE SYSTOLIC ARRAYS

The basic unit of time (B) is based on the computing of the t values with the quadratric formula, and the comparing of the t values for the smallest positive value. The quadric surfaces are more complex than the planar polygons, and require more time to determine the t value to output to the next cell. The variability in time comes from the square root calculation and the comparison of the t values. The square root can be calculated by using an iterative algorithm, or retrieved from a lookup table. The number of iterations can not be predetermined, and is based on the desired number of digits of accuracy. The lookup table removes this variability, but requires the generation of a globally accessible data structure. The other source of variability with respect to time is the comparison of the t values. Upon the selection of the t value to be input to the next cell, the cell may be idle until the B unit of time has elapsed.

The intial delay until the first t value is available is as before: M*B, where M is the number of cells (I.E., THE NUMBER OF QUADRIC SURFACES). The subsequent t values are available every B units for the remaining pixels (N-1). The total time that must elapse before all the t values are available is:

$$M*B + (N-1)*B = B*(M + N - 1).$$

The time needed to perform the actual point calculations, and the extent comparisons, is the same length as for the polygons. If the time to make these calculations is L, the time that must elapse before the surface point is available to the remainder of the algorithm is M*B + L. The total time that must elapse before all the points are available to the algorithm in the host is:

$$M*B + L + (N-1)\ (B + L) = B*(M + N - 1)$$

For execution on a single processor computer the analysis is as follows. The total time that must elapse before the first t parameter is available is M*B + L. The total time that must elapse before all the intersection points are available is:N*(M*B + L) = N*M*B + N*L. Calculations are based on the following high level algorithm:

```
FOR ALL THE PIXELS(N)
  FOR ALL THE OBJECTS(M)
    CALCULATE THE INTERSECTION POINT(B)
  END FOR
  DETERMINE ACTUAL SURFACE POINT(L)
END FOR
```

Since the time to determine the actual surface point

is the same as for polygon approximation of the scene, and the basic unit of time to perform the t calculations is longer, the number of input t values to the separate single cell that must be queued should be smaller than for polygon approximation, or non-existent.

2.3.1.3 COMPOSITE SOLIDS

2.3.1.3.1 CONSTRUCTIVE SOLID GEOMETRY SOLIDS

Ray tracing has been used to synthesize objects built by solid modelers, in particular, by constructive solid geometry (CSG) modelers [ROTH82, WYVI86, CASA85, FLUN85]. A constructive solid modeler (CSG) modeler builds up an object from a set of primitive solids that are combined hierarchially in a binary tree with boolean operators. The primitive objects typically are: block, sphere, cylinder, torus, and cone. The regularized boolean operators are union, intersection, and difference.

The binary tree consists of the primitive solids as the leaves, and the interior nodes are the boolean operators. The root node represents the final object. Each interior node represents a boolean operation, or a rigid body motion such as translation and rotation, that is applied to the corresponding subtree. When a object is

rendered with ray tracing, each ray must traverse the binary tree, representing the object, starting at the root. The intersections of the straight line ray with the primitive objects, making up the composite object, are calculated, and the ray is subdivided into regions that are outside, on, or inside the boundaries of the solids. The ray is a parametric equation of a line anchored at the viewer, and directed toward the scene. When the intersection calculations with a solid primitive have been performed, the results are combined with the results of the other branch from the same parent node, according to the boolean operator in the parent node. This information along with an indentification of the intersected surface is maintained in a global data structure. The results of one level are combined and returned to the next higher level in the tree until the root is reached.

The sides of a block are usually represented as six half planes. The sphere, cylinder, and cone are examples of quadric surfaces; the torus is represented by a quartic polynomial. The different representations imply that different calculations are needed to accommodate the different surface types, and these calculations require a different amount of time. Also, the hierarchical

relationships among the primitive solids must be maintained. Both of these features of CSG solids preclude the beneficial use of a one dimensional systolic array to perform the intersection calculations. The uniformity criteria for qualification for execution on a systolic array can not be met.

2.3.1.3.2 HETEROGENEOUS SCENE

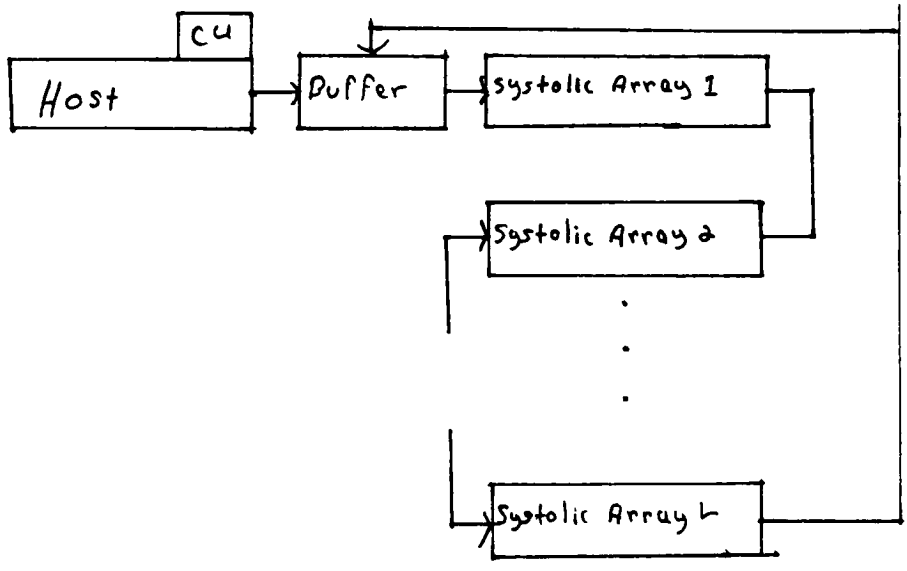
A scene consisting of surfaces of different mathematical (ANALYTICAL) representations can be rendered with a systolic based architecture to facilitate the intersection calculations. The systolic based architecture consists of a number of one dimensional arrays. Each cell of the systolic array calculates the intersection for a particular surface type. The length of each array is equal to the number of objects in the scene with the same mathematical representation. Each cell of the array is preloaded with the corresponding instructions to perform the intersection calculations, and with the coefficients of one object in the scene with the representation corresponding to the array. The basic unit of time, the time to perform one intersection calculation, is different for each array. This difference in time for the intersection calculations must be resolved before the systolic based architecture can be

used.

The systolic arrays should be arranged in decreasing or increasing time to perform the intersection calculations. If the arrays are sorted in increasing time to perform the intersection calculations, the circulated t values, and ray parameters generated from one array will queue up waiting to circulate through the next slower array. The length of the queue depends on the relative slowness of one array with respect to a previous array. This arrangement of the arrays ensures that each array is in continuous operation, once the the values enter the array, but also requires more memory to hold the queued inputs. The alternative arrangement is to order the arrays in decreasing time of the intersection calculations. The subsequent arrays require a shorter time interval to calculate the ray-surface intersections. As a consequence, the subsequent arrays will not be continuously active. The previous arrays will generate t values at a rate slower than the next array, and, therefore, no memory is needed to hold intermediate results. The time interval between consecutive inputs to an array should become shorter, as subsequent arrays become active, and consequently each array must have its own clock. A high level description of a supporting

architecture is given in Figure 7.

The systolic arrays can be tailored to the specific quadrics in the scene (I.E.,SPHERE,ELLIPSOID). The calculation of the total time is based on the following timing diagram in Figure 8. The timing lines for each systolic array are labled O - L. The timing calculations assume that the time through each systolic array is multiplied by the respective unit of time B, that is, the time spent in each systolic cell. The markings on the time line represent the number of systolic cells(M) and the number of pixels(N) for each systolic array. The times selected are those that indicate the time for different number of pixels to circulate through the systolic arrays, one for each surface in the scene. The starting time of one array is relative to the starting time of a previous array. Thus, the starting time of the ith array is the sum of the starting times of the previous (I-1) arrays. For example, the starting time of array (2) is determined as follows. The first array has no previous array, and its starting time is zero. The starting time of the second array relative to the first array is M. The starting time of the third array relative to the second array is M'. The



NOTE: L IS THE NUMBER OF
SYSTOLIC ARRAYS

FIGURE 7: HIGH LEVEL DESCRIPTION OF A SYSTOLIC BASED ARCHITECTURE TO SUPPORT A HETEROGENEOUS SCENE.

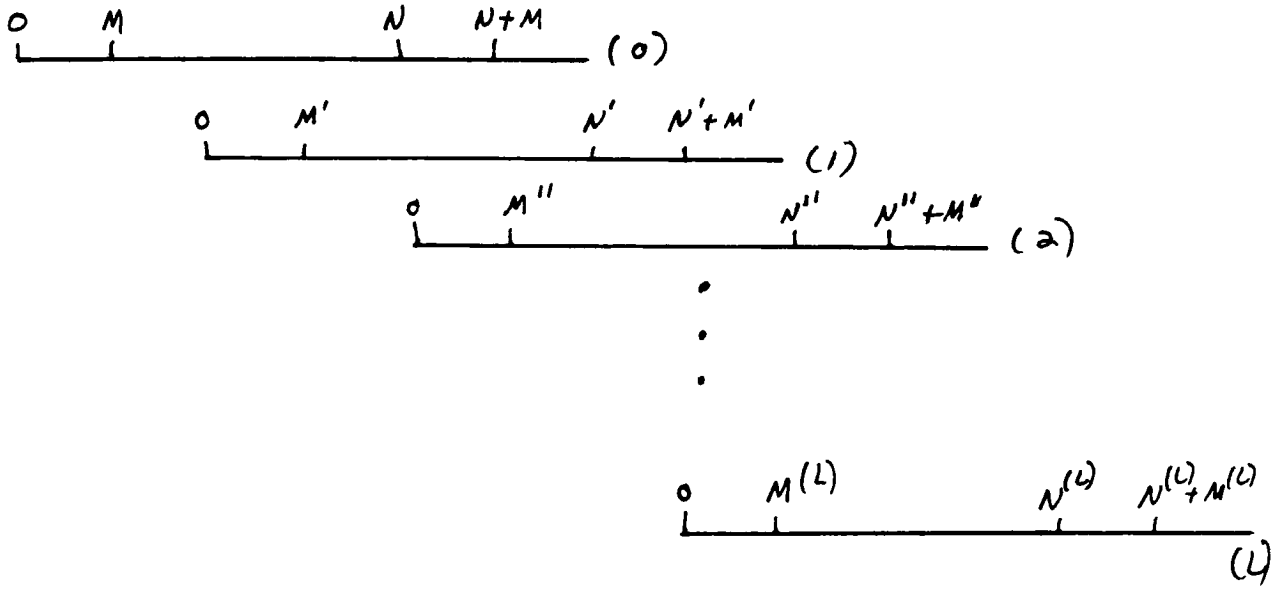


FIGURE 8: TIMING DIAGRAM FOR INTERSECTION CALCULATIONS FOR ARCHITECTURE IN FIGURE 7.

starting time of the third array is: O + M + M'.

Therefore, the starting time of the last array is :

$$\sum_{l=0}^{L-1} M^{(I)}$$

The total time for the pixel data stream to circulate through all the arrays is:

$$\sum_{l=0}^{L-1} M^{(I)} + M^{(L)} + N^{(L)} - 1$$

When a t value is generated, the surface identifier must also retrieve from the lookup table the surface type, or the surface type must be circulated through the arrays. The surface type is used to load the program counter (PC) with the starting address of the appropriate instruction sequence, shown in Figure 9. Thus the starting address must be stored in predefined locations. The different instruction sequences increase the size of needed memory.

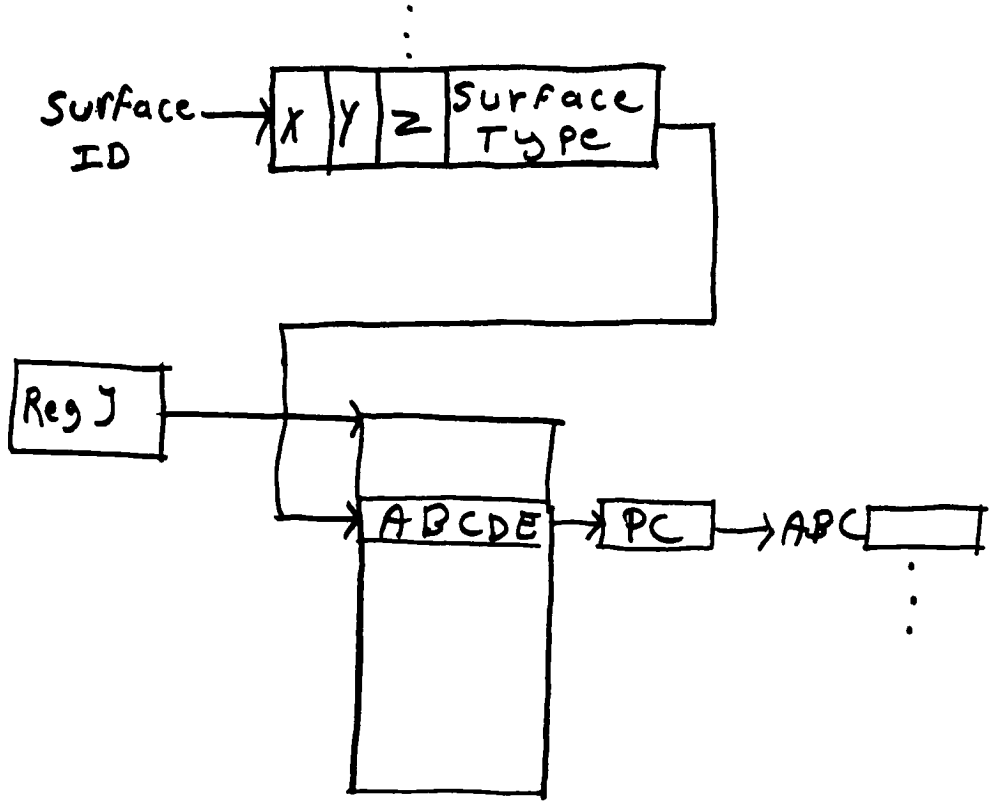


FIGURE 9: CALCULATION OF THE STARTING ADDRESS OF THE INSTRUCTION SEQUENCE USING THE SURFACE IDENTIFIER

Surfaces described by polynomials of degree greater than two have been rendered using ray tracing [WHITT80, HANR83]. Among the algebraic surfaces that have been ray traced are the bicubic, quartic, Kummer's quadruple and Steiner's surface. Analytical solutions exist for some polynomials of degree less than five, but for most polynomials degree less than five, and for those

polynomials of degree at least five, numerical methods and/or combinations of numerical and analytical methods need to be used to determine their intersections with the rays [HANR82]. The numerical methods used depend on the desired accuracy. The number of interations executed depend on the desired precision. These interative methods are suitable for hardware execution, but not by the systolic arrays. There exists no numerical method that can find the roots of a general algebraic surface within a predetermined number of iterations. The intersection calculations are not a straight forward application of an arithmetic formula. Hence, the calculations are not regular. A scene made up of different algebraic surfaces would either require different numerical methods or the same method to find a different number of roots. the time to perform the intersection calculations is variable possibly for each object in the scene.

The ray tracing of parametrically defined patches and surfaces also involve numerical methods [KAJI82, TOTH85, SWEE85]. Again, the intersection calculations are not regular for the same reasons given in the previous paragraph.

2.3.2 RAY TRACING WITH GLOBAL ILLUMINATION

Ray tracing, when used with a global illumination model, yields more realistic looking objects. The increase in complexity, due to the inclusion of simulated, reflected and refracted light rays is mirrored by an increase in the complexity of the supporting systolic based architecture. The type and degree of sophistication in the hardware to support the algorithm is dependent on the sophistication of the illumination model, the type of surface detail, and the internal description of the objects in the scene. In traditional ray tracing, the reflection and refraction rays associated with the initial ray are used to build a binary tree.

The root of the tree is the initial pixel ray-surface intersection. This initial intersection indicates that the corresponding surface point is visible to the observer, and the light ray reaching the observer from that visible point is either a reflected or a refracted ray. The association between the pixel and the surface must be stored in the root node of each binary tree. The reflection ray forms the left branch of the binary tree, and the refraction ray forms the right branch of the binary tree. The contributions of other reflected and refracted light rays reaching the visible

surface point, are calculated based on a model of light requiring the calculation of the surface normal at the intersection point. The contributing reflected and refracted light rays are traced by the global illumination model to their immediate surface or point light sources. The reflected and refracted light sources, that contribute to the intensity of the immediate ray-source-surface intersection points, are also traced by the global illumination model. As the illumination model traces the contributing reflected and refracted rays, a binary tree is built. When all of the rays have either reached their point light source, or continue out of the scene, or have been traced a predetermined number of iterations, the ray tracing stops. The binary tree is traversed in postorder by the shader to determine the intensity of the corresponding pixel. The intensity of the light reaching an intersection point, stored in the binary tree, is reduced by the distance between that intersection point, and the intersection points of the immediate left and right children nodes. The incoming light rays to the intersection point are stored in the parent node. The number of trees generated depend on the position of the viewer, and on the relationships between the objects in the scene.

The recursive application of the global illumination model indicates that the ray coordinates and ray parameters must be sent through the systolic based architecture more than once. Also, the calculations for determining the normal and the coefficients of the reflected and refracted rays, based on the illumination model, require a more sophisticated single separate cell. A high level description of the ray tracing algorithm with a global illumination model, that must be mapped onto a systolic based architecture, is given here:

```
PICK A PERSPECTIVE POINT
FOR THE NUMBER OF PIXELS IN THE VERTICAL
                                    DIRECTION
   FOR THE NUMBER OF PIXELS IN THE HORIZONTAL
                                       DIRECTION
      DETERMINE THE INITIAL RAY
      FOLLOW_RAY(INITIAL_RAY,TREE_PTR)
      TRAVERSE_TREE(TREE_PTR,PIXEL_VALUE)
   END FOR
 END FOR
FOLLOW_RAY(RAY,TREE_PTR)
  FOR EACH SURFACE IN THE SCENE
    DETERMINE THE POINTS OF INTERSECTION(POI)
    SORT THE POI'S AND ELIMINATE THOSE OUT OF RANGE
    IF THERE IS A CLOSEST, VALID INTERSECTION POINT
    THEN
       ADD NODE TO TREE INCLUDING THE POI AND
                    ANY NECESSARY SURFACE INFORMATION
       DETERMINE REFLECTION_RAY
       DETERMINE REFRACTION_RAY
       FOLLOW_RAY(REFLECTION_RAY,LEFT_BRANCH)
       FOLLOW_RAY(REFRACTION_RAY,RIGHT_BRANCH)
    ELSE
       NO POI
       LEFT AND RIGHT BRANCH ARE NULL
    END IF
```

Traverse_tree is a shader that performs the shading

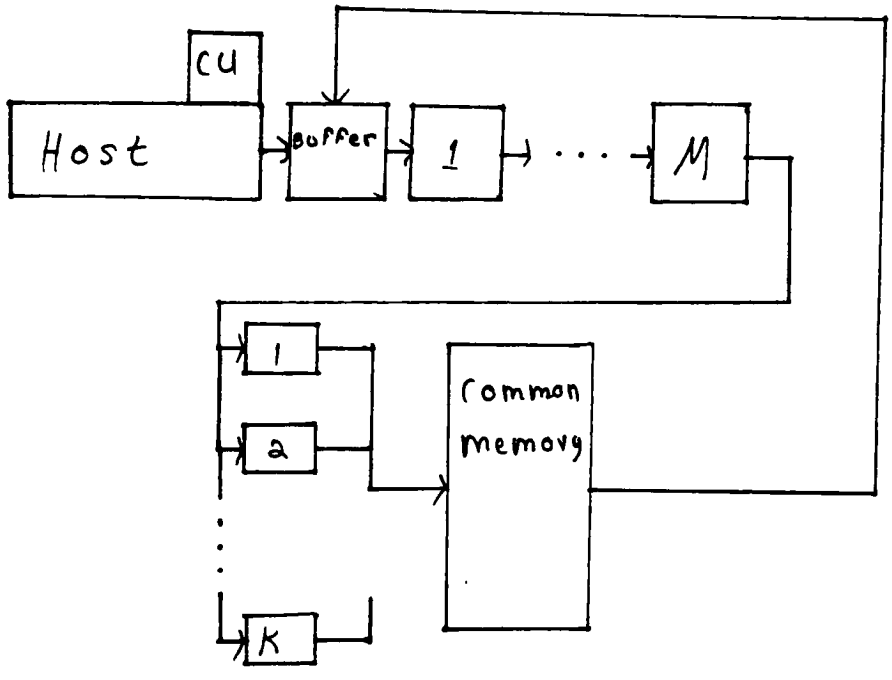
calculations which implement the global illumination model. The systolic architecture onto which the above mentioned algorithm must be mapped is described in this section at a high level.

The major components, as shown in Figure 10, of the systolic based architecture are:

- interface buffer between the host and systolic array
- one dimensional array of linearly connected cells(systolic array)
- control unit
- array of separate cell processors
- memory shared by the separate cell processors and the systolic array

The interface buffer is a data communication buffer between the host computer and the systolic array. The host sends the pixel coordinates to the systolic array via the interface buffer. The intersection point/indicator associated with each pixel is sent to the host via the same buffer, when hidden surface removal is being implemented with ray tracing. The binary true or a no valid indicator, associated with each pixel sent through the systolic array, is sent to the host computer,

when a global illumination model is being implemented with ray tracing.



WHERE:

- = NUMBER OF SYSTEM CPUS
K = INTEGER VALUE OF 0

FIGURE 1.1: LOGICAL BLOCK DIAGRAM TO IMPLEMENT THE DISTRIBUTION CALCULATIONS FOR RAY TRACING.

The user-defined real array of size N elements

cells is the systolic array. Each cell is capable of performing the same calculations. The instruction sequence to implement the calculations are preloaded prior to the execution of the calculations. Data initially enters the systolic array from the interface buffer.

When implementing the ray tracing algorithm for hidden surface removal, once each pixel has circulated through the systolic array, the corresponding t parameter value and the surface identifier are put onto a data bus connected to the interface buffer. These results are then available to the host computer. When implementing the ray tracing algorithm as a part of a global illumination model, the output generated by the systolic array is sent to the first available separate cell processor. Each separate cell uses the output values to build a branch of a binary tree, which is later used by the shader software to calculate the corresponding pixel intensities.

Once a t value enters a separate cell, the corresponding incomplete tree and the data structures used by the tree building algorithm, including the stack, are retrieved from a predofined location of common memory. The reflection and refraction rays are calculated and stored in the binary tree. The refraction ray is

pushed onto the stack and the reflection ray is sent to another predefined location in memory. From that location the reflection ray is loaded onto a bus connected to the interface buffer. Once the coefficients of the rays are in the interface buffer, the coefficients are sent through the systolic array. The intersection values, generated from the stream of input ray coefficients, are also sent to the separate cell processors to continue the building of the binary trees. Hence, this architecture supports the repeated circulation of the data, and the repeated calculation of the reflection and refraction rays as required by the ray tracing algorithm.

The memory accesible to all of the separate cells, stores the incomplete binary tree, and other information pertinent to the building of the binary tree, in particular; a predefined memory location containing the current binary tree which, which indicates the next available separate cell processor. This global variable is incremented (modulo K) + 1.

The number of rays passing through the display and intersecting surfaces in the scene is the number of binary trees that must be simultaneously built. Those rays that do intersect with objects in the scene form the roots of the separate, distinct binary trees. Therefore,

hardware is included to count the number of initial intersections. This number also indicates the length of subsequent data streams that circulate through the systolic array, as well as, the number of separate cell processors. The algorithm to build the binary tree, and to determine what values are sent through the systolic array, is listed as an algorithm.

```
LEVEL_NO <- 0
CREATE ROOT NODE
STORE POI,LEVEL_NO,RAY_TYPE
      (ROOT GIVEN REFLECTION RAY TYPE)
PUSH POI,LEVEL_NO AND RAY_TYPE ONTO STACK
WHILE STACK NOT EMPTY
 POP POI,LEVEL_NO,RAY_TYPE,RAY PARAMETERS
 IF RAY_TYPE = REFLECTION
 THEN
   CURRENT_LEVEL <- LEVEL_NO
   CALCULATE REFLECTION AND REFRACTION RAY
                              PARAMEMTERS AND POI
   PUSH REFRACTION RAY
   IF REFLECTION RAY HAS CLOSEST INTERSECTION POINT
    THEN
      CREATE LEFT NODE
      STORE POI
      INCREMENT LEVEL_NO
      STORE LEVEL_NO AND RAY_TYPE
      PUSH POI,CURRENT_LEVEL AND RAY_TYPE , RAY
                                            PARAMETERS
      END IF
   END IF
 IF RAY_TYPE = REFRACTION
  THEN
    CURRENT_LEVEL <- LEVEL_NO
    IF REFRACTION RAY HAS CLOSEST INTERSECTION POINT
     THEN
       CREATE RIGHT NODE
       STORE POI
       INCREMENT CURRENT_LEVEL
```

```
            STORE CURRENT_LEVEL AND RAY_TYPE
            CALCULATE REFLECTION AND REFRACTION RAY
                                                PARAMETERS
            PUSH POI,CURRENT_LEVEL,RAY PARAMETERS,
                                    REFLECTION RAY
            PUSH POI,CURRENT_LEVEL,RAY PARAMETERS,
                                    REFRACTION RAY
        END IF
      END IF
END WHILE
```

This algorithm is implemented by each active separate cell processor. Included in the pertinent information for building the tree are the stack, containing the branches to be added, and the current level number of the binary tree. The binary tree is built in a depth first (preorder) order that can be sent as a linear array to the host via the interface buffer. A comparison of the length of time to perform the intersection calculations and of the length of time to calculate the normal, and the refraction and reflection ray coefficients also are a means to determine the number of separate cell processors that are to be included as a part of the systolic based architecture. Both calculations depend on the complexity of the description of the surface, and the normal, the reflection and refraction ray calculations are also dependent on the global illumination model being implmented.

If the length of time to build one branch of a

binary tree is less than or equal to the time to perform the required intersection calculation, then only one separate cell processor is needed to build one branch of a binary tree. If the length of time to build one branch of a binary tree is longer than the time to perform the intersection calculation, then more than one separate cell processor is needed to prevent the queuing of the t values. The length of time to build a branch of the binary tree should be rounded up to the next integer multiple of the length of time to perform one intersection calculation (B). This integer is the number of separate cell processors needed to build the branches of the binary trees, and to prevent the output t values from queuing on input to the separate cell processors. These cell processors must access a common memory that stores the binary trees. Should the number of the number of binary trees that must be simultaneously generated exceed the number of separate cell processors, the above mentioned scheme causes the t values to queue up outside the separate cell processor

The first data stream sent through the systolic array consists of the pixel coordinates, the ray parameters, t values, and surface identifiers. The subsequent data streams consist of the surface points,

the ray parameters, t values, and surface identifiers. For each subsequent data stream sent through the systolic array, a different set of rays will intersect with surfaces in the scene. The members of each set can not be predetermined. The nonintersecting rays indicate the terminations of branches in the binary trees. For those rays that do not intersect, a placeholder is needed to indicate the termination of a branch for the branch calculations, and to maintain the constant length of the data stream.

As indicated by the algorithm, the left branch (REFLECTION RAY) is generated first, until either, the tree has reached a predetermined length, or the ray intersects with no object in the scene. In terms of execution on the systolic array, this means that a data stream will be sent through the array a predetermined number of times, or until all the rays in the data stream exit the scene. For each reflection ray a refraction ray is calculated and traced (RIGHT BRANCH OF THE BINARY TREE). The refracted ray data stream executes on the systolic array similiarly to the reflected ray data stream. Once all the binary trees have been generated, the linear arrays representing the binary trees, are sent to the shader in the host. A tree depicted as shown in

Figure 11.

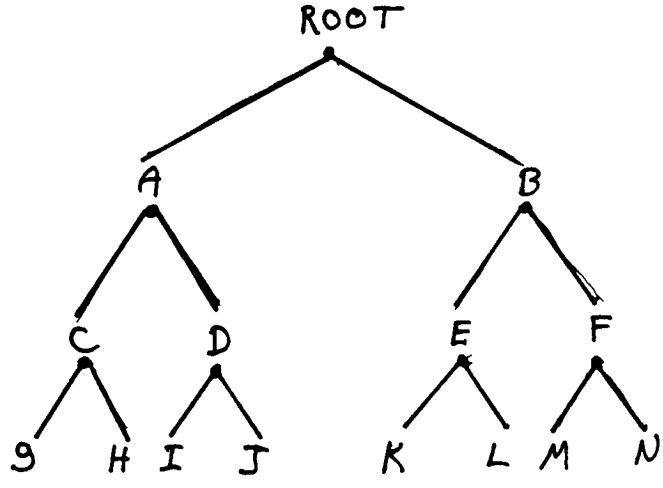


FIGURE 11: LINEAR REPRESENTATION OF A BINARY TREE PRODUCED BY AN ALGORITHM IMPLEMENTED ON A SYSTOLIC ARRAY

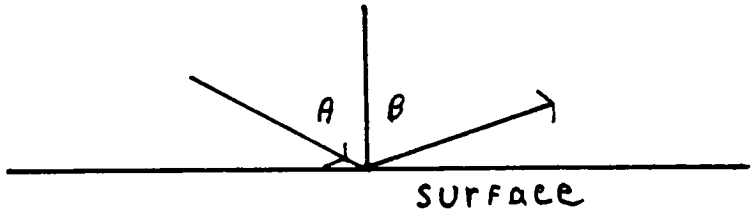
Is sent as an array of nodes:

ROOT,A,C,G,H,D,I,J,B,E,K,L,F,M,N

Each node contains:

- RAY POINT
- SURFACE IDENTIFIER
- RAY TYPE
- RAY PARAMETERS

When a light ray reaches the surface of an object, the portion of light that is reflected and transmitted (REFRACTED) determines the visibility of the object. The angle of an incident light ray striking the surface at a point is measured with respect to the normal at the surface point, and is called the angle of incidence. The angle of a reflected light ray is also measured with respect to the normal, and is called the angle of reflection. If the light ray strikes a smooth surface, the angle of incidence equals the angle of reflection. This phenomena is called regular reflection. The angle of reflection and the angle of refraction are in the same plane.



A = B

A diffuse surface that reflects light equally in all directions produces diffuse reflection and is called a perfect diffuser. The intensity of the reflected light from a perfect diffuser is based on Lambert's Cosine Law:

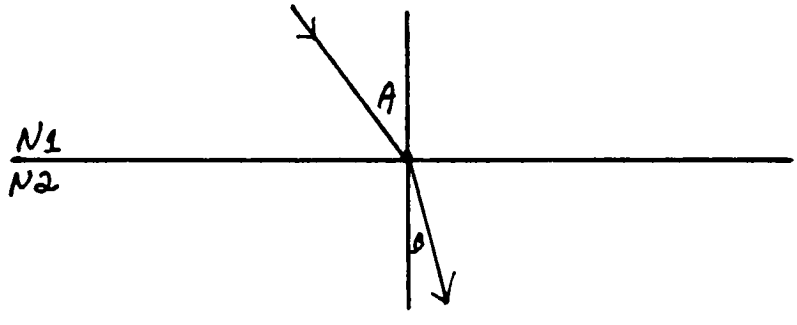
$$I = IL*KD*COS(A)$$

WHERE:

I = INTENSITY OF THE REFLECTED LIGHT RAY
IL = INTENSITY OF THE INCIDENT LIGHT RAY FROM A POINT LIGHT SOURCE
KD = A DIFFUSE REFLECTION CONSTANT (0 <= KD <= 1) THAT DEPENDENT ON THE SURFACE MATERIAL
A = DIRECTION OF THE REFLECTED LIGHT RAY WITH RESPECT TO THE NORMAL
(0<= A <= 90) (A > 90 MEANS THE LIGHT SOURCE IS BEHIND THE OBJECT)

This calculation is included in the shader as a part of the illumination model if reflection is being implemented for a smooth surface.

The transmitted light (REFRACTED) is shown schematically here:



And follows Snell's Law:

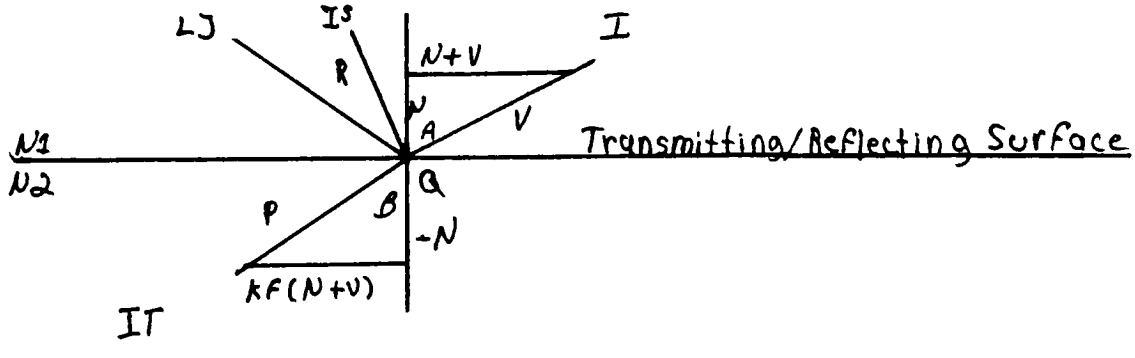
$$N1*SIN(A) = N2*SIN(B)$$

WHERE:

A,B - ARE IN THE RANGE (0,90)

N1,N2 - ARE THE INDICIES OF REFRACTION OF THE RESPECTIVE MEDIA
(I.E., THE DEGREE OF TRANSPARENCY OF THE SURFACES)

When the angle of incidence equals or exceeds 90 degrees, B, the angle of refraction is the critical angle. The incident angle obeys the laws of regular reflection, and the light rays are reflected totally within the first medium. The calculations of the reflection and refraction components indicate intersection calculations that must be performed by the separate cell processors. The calculation of the direction of the reflection and refraction light rays is included in traditional ray tracing as a part of the global illumination model as shown here:



WHERE:

N1,N2 - THE INDICES OF REFRACTION
Q - POINT OF INTERSECTION
IS - INTENSITY OF LIGHT DUE TO SPECULAR HIGHLIGHTS

FROM OTHER OBJECTS/SURFACES IN THE SAME SCENE
IT - INCOMING INTENSITY OF REFRACTED LIGHT
-R - DIRECTION OF INCIDENT LIGHT FROM A POINT LIGHT SOURCE
-V - DIRECTION OF LINE OF SIGHT OF THE VIEWER
P - DIRECTION OF THE REFRACTED LIGHT RAY
LJ - JTH POINT LIGHT SOURCE
The reflection and refraction rays are defined as follows:

```
R = V' + 2N
P = KF(N + V') - N
V' = V/MAGNITUDE(V.N)
KF = (KN**2 * MAGN(V')**2 - MAGN(V' + N)**2)**-1/2
KN = N2/N1
```

A complex value for kf indicates total internal reflection. The index of refraction is needed to determine the direction of the refracted light ray. This information must be avalilable for each surface through which refracted rays pass. The surface identifier could then be used as an pointer into the lookup table that should also contain the pointer of refraction for that surface. An entry in the lookup table would contain the following information is shown in Figure 12. The illumination dependent information is loaded into the separate cells at the same time the instructions to perform the reflection and refraction ray parameter calculations are loaded. The index of refraction of air is 1 for calculations.

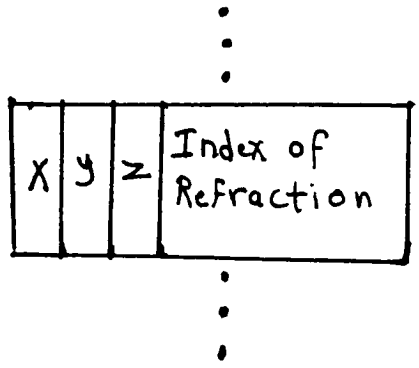


FIGURE 12: EXAMPLE OF AN ENTRY IN THE LOOKUP TABLE USED BY THE SINGLE SEPARATE CELL TO CALCULATE THE REFLECTION AND REFRACTION RAYS.

The illumination dependent information is loaded into the separate cells at the same time the instructions to perform the reflection and refraction ray parameter calculations are loaded. The index of refraction of air is 1 for calculations.

2.3.2.1. POLYGON APPROXIMATION

For a scene approximated with polygons, the architecture is as depicted in the previous section. The intersection calculations are as described in an earlier section. The calculation of the reflection and refraction rays depend on the mathematical representation of the surface. The initial ray travels along the line of sight through a pixel coordinate to a point on the surface. Hence, three points on the line segment (RAY) are known. Using the position of the viewer (B) and the point of intersection on the polygon surface (A), the directed

line segment BA is:

$$V = (AX - BY)I + (AY - BY)J + (AZ - BZ)K$$

Given that the form of the planar equation is:

$$DX + EY + FZ + G = 0$$

The vector normal to the plane is:

$$DI + EJ + FK$$

The unit normal is:

$$N = (D/L)I + (E/L)J + (F/L)K$$

WHERE:

$$L = (D**2 + E**2 + F**2)**1/2$$

The calculation of the reflection ray is a straightforward substitution of the ray equation:

$$R = V' + 2N$$

$$V' = ((AX - BX)I + (AY - BY)J + (AZ - BZ)K) / MAGN(V,N)$$

WHERE:

$$V.N = (AX - BX)(D/L)I + (AY - BY)(E/L)J + (AZ - BZ)(F/L)K$$

$$M = MAGN(V.N) = (((AX - BX)(D/L))**2 + ((AY - BY)(E/L))**2 + ((AZ - BZ)(F/L))**2)$$

$$2N = 2\left(\frac{DI}{L} + \frac{EJ}{L} + \frac{FK}{L}\right)$$

$$R = \frac{(AX - BX)}{M} + \frac{2D}{L}I + \frac{(AY - BY)}{M} + \frac{2E}{L}J +$$

$$\frac{((AZ - BZ) + 2F)K}{M \qquad L}$$

The calculation of the reflection ray is based on the two points, and on the coefficients of the plane. The information that must be stored in a node of the binary tree for the reflection rays includes: the intersection point, the surface identifier of the surface containing the point of intersection, and the coefficients of the reflection ray (I.E., THE DIRECTION OF THE REFLECTED LIGHT). These are the same values that must be stored on the stack to implement the algorithm. The refraction ray uses the same information, except for the index of refraction. The calculations are a straightforward substitution of the equation:

$$P = KF(N + V') - N$$

The assembly language like instructions needed to determine the reflection and refraction rays is given in Appendix D. The high level description of a separate cell is given in Figure 13.

There must be a global data structure, in a fixed location, used to indicate the next binary tree to be updated: the current binary tree. This number is a pointer into a table that contains a pointer to an area

of memory, which contains an incomplete binary tree, and the local data strucutres (I.E., STACK AND POINTER). As each t value enters a separate cell, for the second and subsequent data streams, the current binary number is incremented (MODULO P) + 1. The first data stream determines the value of P.

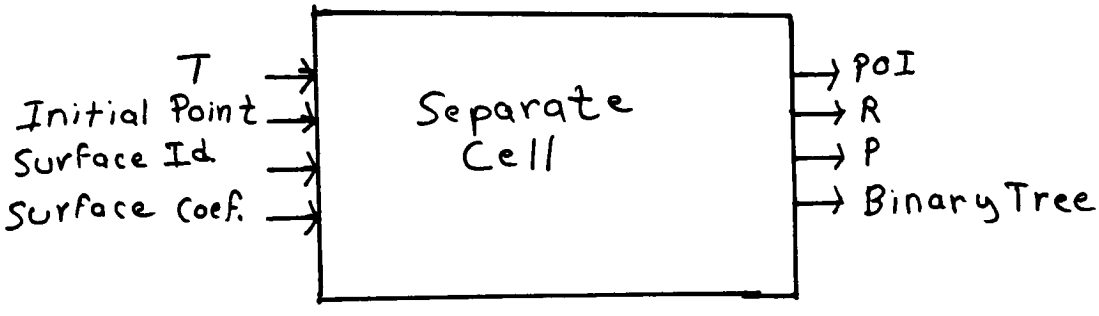


FIGURE 13: SEPARATE CELL FOR DETERMING THE REFLECTION AND REFRACTION RAYS.

There should be no contention by the separate cells for access to the current binary number. When the t values are input to the separate cells, the incremented current binary tree number will initiate access to memory for the corresponding binary tree, and data structures needed to build the next branch.

2.3.2.1.1 ANALYSIS OF USE OF SYSTOLIC ARRAY

The time that must elapse before the first t value is available is M*B units. The time to build one branch

of a binary tree is K*B units. The situation depicted pictorially is the data stream of pixels going through the systolic array in Figure 14. Additional time is needed to retrieve and store the incomplete binary trees as well as to send the data stream to the interface buffer to be recirculated through the systolic array.

At time, t=N, the first reflection/refraction ray can be circulated through the systolic array. Assuming also that the data is to be circulated through the array a predetermined number of times (I.E.,2**(L+1)-2, where L is the length of the tree), at time, t=N, the situation is as depicted in Figure 15.

The length of the data streams for the reflection/refraction rays can range from zero to N (I.E., N >= P >= 0). Assuming that P=N the situation continues as depicted in Figure 16.

For the initial pixel coordinates going through the systolic array, at time t, the (T-M+1)th parameter t value is generated. For the subsequent data streams with length equal to the number of pixels (N), with K =1, and with 2** (L+1) -3 equal to the number of reflection and/or refraction data streams to be sent through the systolic array, at time t, the (T-M+1)th parameter t

value is generated.

The situation for which N > M, N > P, and K > 1 hold, is depicted in the foldout of Appendix E.

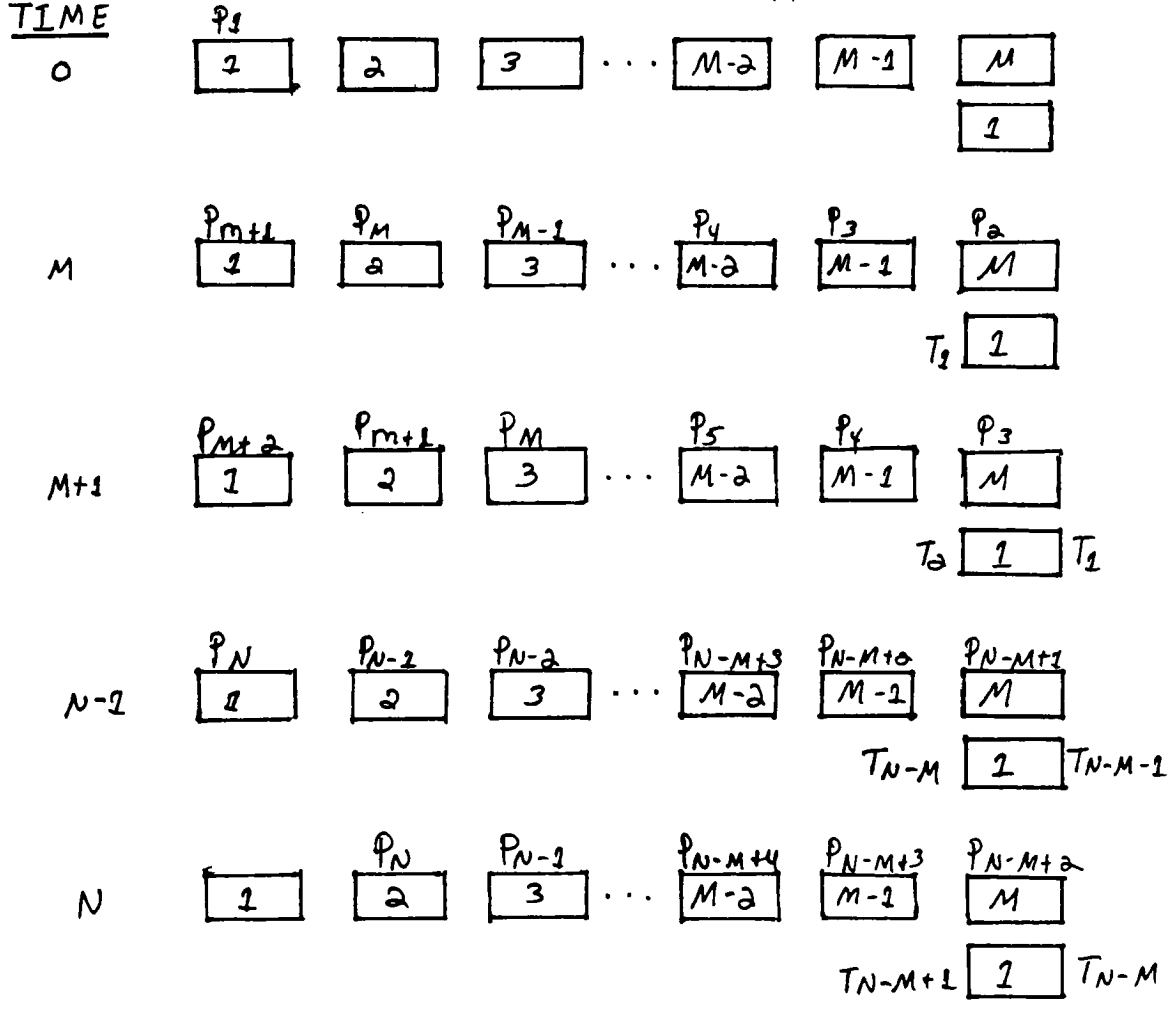
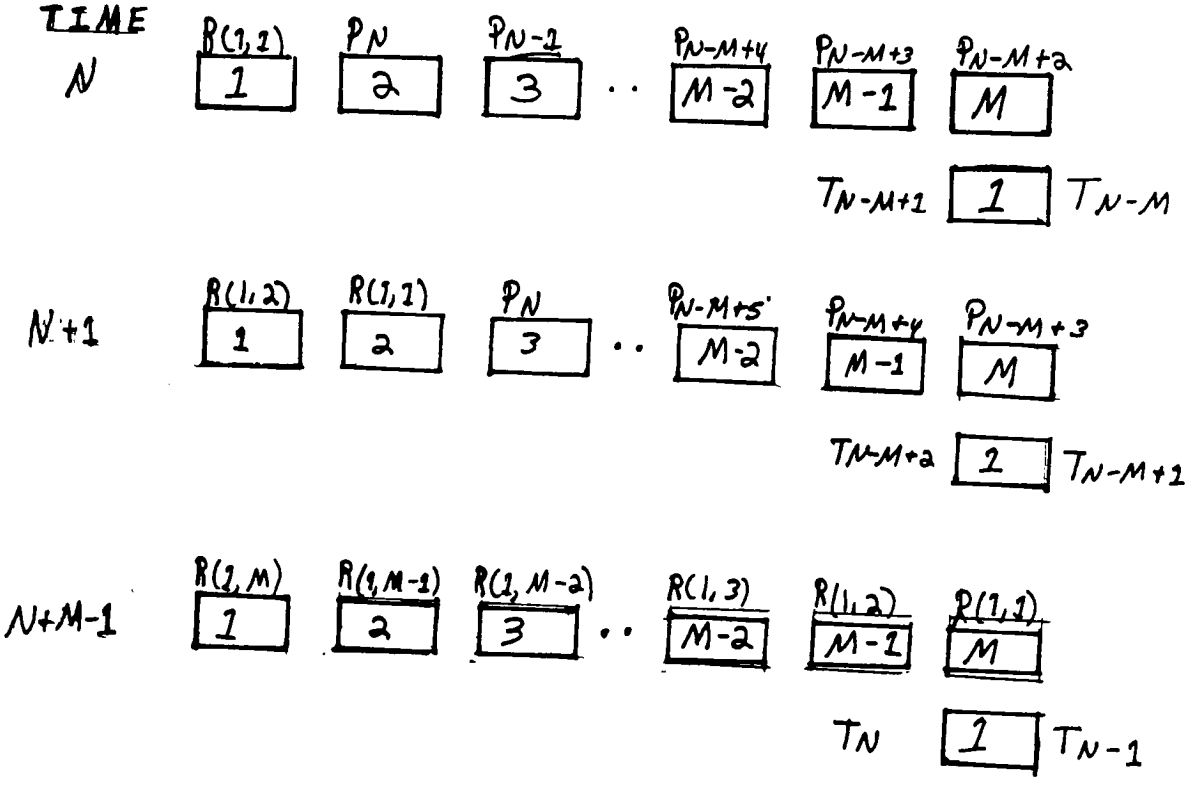


FIGURE 14: PICTORIAL DEPICTION OF THE TIMING OF A DATA STREAM CIRCULATING THROUGH A SYSTOLIC ARRAY.



NOTATION R(X,Y) IS AS FOLLOWS: XTH DATA STREAM
YTH POSTION WITHIN
THE XTH DATA STREAM

FIGURE 15: PICTORIAL DEPICTION OF THE TIMING OF A DATA STREAM RECIRCULATING THROUGH A SYSTOLIC ARRAY.

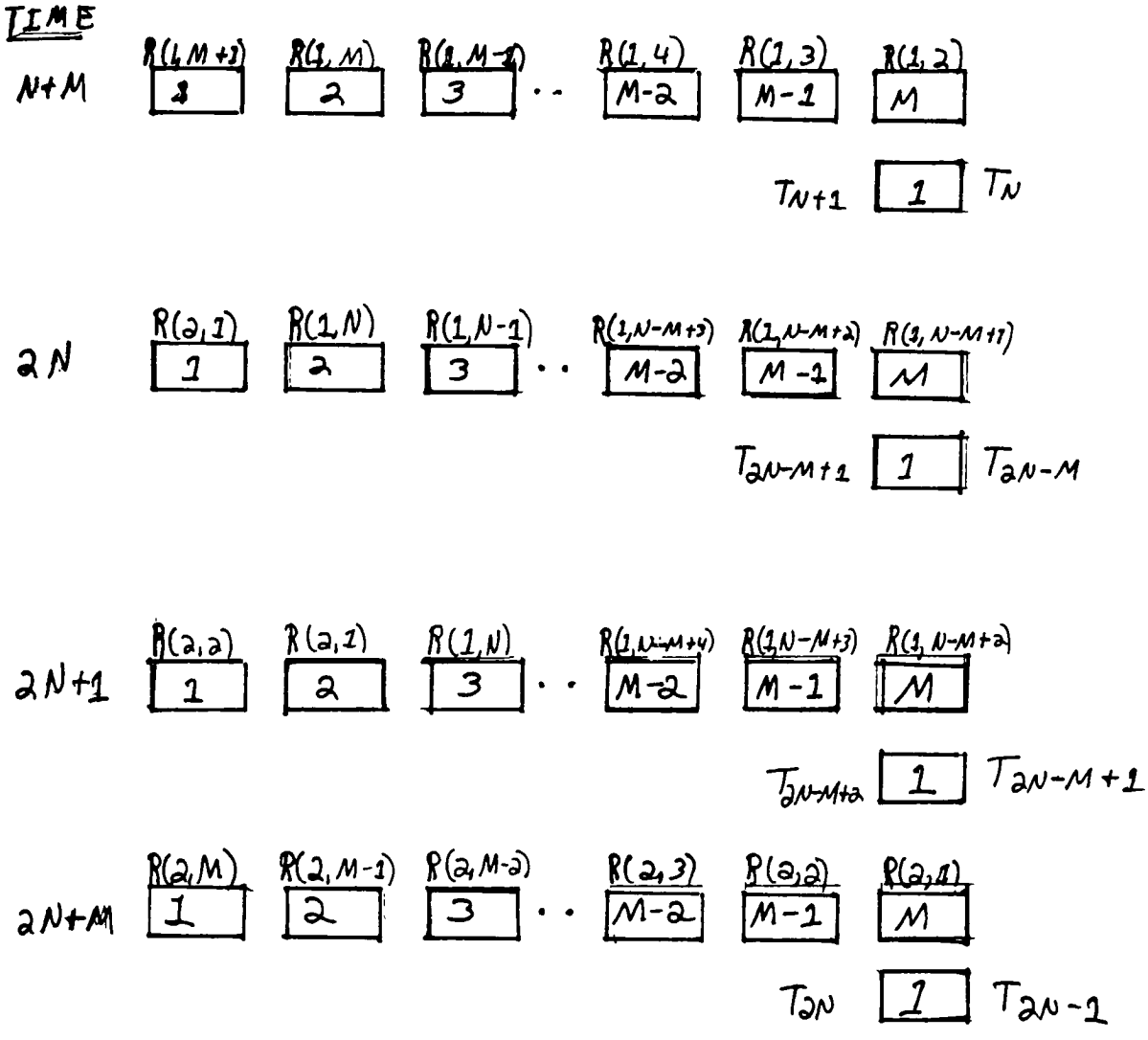
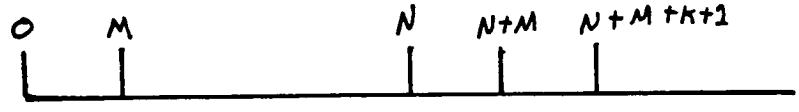


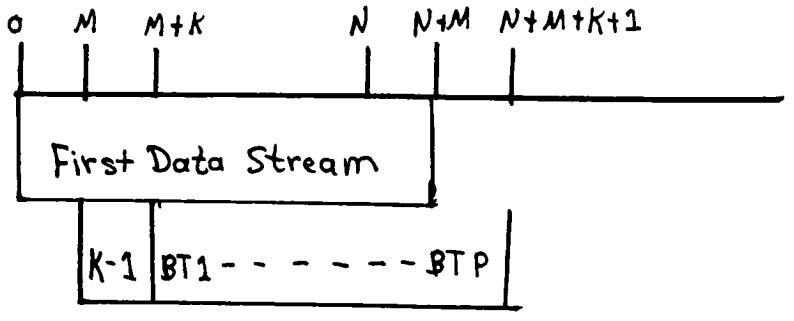
FIGURE 16: PICTORIAL DEPICTION OF THE TIMING OF THE DATA STREAM RECIRCULATING THROUGH A SYSTOLIC ARRAY WITH BUILDING OF THE REFLECTION/REFRACTION BINARY TREES.

The total time to build each binary tree is determined with the aid of the following timimg diagrams.



TIME	ACTION
0	FIRST PIXEL ENTERS THE SYSTOLIC ARRAY
M	FIRST PIXEL ENTERS A SEPARATE CELL
N-1	LAST PIXEL OF THE FIRST DATA STREAM ENTERS THE SYSTOLIC ARRAY
N	FIRST RAY OF THE SECOND DATA STREAM ENTERS THE SYSTOLIC ARRAY
N+M	LAST PIXEL OF THE FIRST DATA STREAM ENTERS A SEPARATE CELL
N+M+1	FIRST RAY OF THE SECOND DATA STREAM ENTERS A SEPARATE CELL
N+M+K	LAST PIXEL OF THE FIRST DATA STREAM EXITS A SEPARATE CELL PROCESSOR
N+M+K+1	FIRST RAY OF THE SECOND DATA STRAM EXITS A SEPARATE CELL

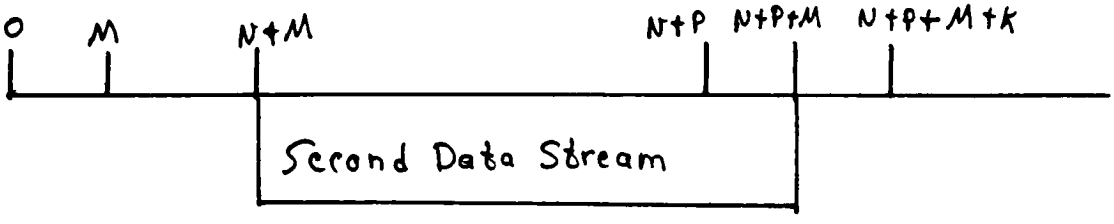
The time to establish the binary tress is depicted here.



ESTABLISHING THE BINARY TREES

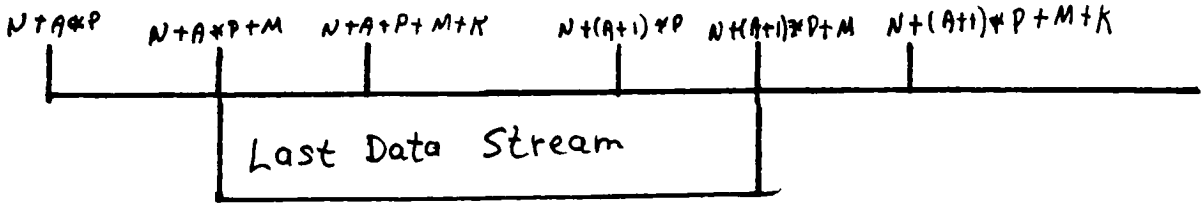
TIME	ACTION
M+K	FIRST PIXEL ENTERS A SEPARATE CELL

There is a delay of (K-1) units before the root of the next binary tree (BT) is established. This delay time is needed to compute the reflection/ refraction rays, and to determine the point of intersection. Every B unit of time thereafter, either the root of a binary tree will be established, or a no-valid intersection indicator will be issued. The number of binary trees established is P. The current binary tree number is incremented each time a tree is established. Based on the timing diagram the total time to build one branch/establish P binary tree is (N + K)units. The timing for the second and subsequent data streams is as shown here.


0
M
N+M
N+P
N+P+M
N+P+M+K
Second Data Stream

TIME	ACTION
N+M	FIRST RAY OF THE SECOND DATA STREAM ENTERS THE SYSTOLIC ARRAY
N+P-1	LAST RAY OF THE SECOND DATA STREAM ENTERS THE SYSTOLIC ARRAY
N+P	FIRST RAY OF THE THIRD DATA STREAM ENTERS THE SYSTOLIC ARRAY ARRAY
N+P+M	LAST RAY OF THE SECOND DATA STREAM EXITS THE SYSTOLIC ARRAY AND ENTERS A SEPARATE CELL PROCESSOR
N+P+M+1	FIRST RAY OF THE THIRD DATA STREAM ENTERS A SEPARATE CELL PROCESSOR
N+P+M+K	LAST RAY OF THE SECOND DATA STREAM EXITS A SEPARATE CELL PROCESSOR
N+P+M+K+1	FIRST RAY OF THE THIRD DATA STREAM EXITS A SEPARATE CELL PROCESSOR
N+2P+M	LAST VECTOR OF THE THIRD DATA STREAM ENTERS A SEPARATE CELL PROCESSOR

As the first element of a data stream enters a separate cell, the current binary tree number should be reinitialized to zero. When the last data stream enters the architecture the situation is thus:



NOTE: A IS 2**(L+1) -4
A+1 IS 2**(L+1) -3

TIME	ACTION
N+A*P+M	FIRST RAY OF THE LAST DATA STREAM ENTERS THE SYSTOLIC ARRAY
N+A*P+2M	FIRST RAY OF THE LAST DATA STREAM ENTERS A SEPARATE CELL PROCESSOR
N+A*P+2M+K	FIRST RAY OF THE LAST DATA STREAM EXITS A SEPARATE CELL PROCESSOR
N+(A+1)*P-1	LAST RAY OF THE LAST DATA STREAM ENTERS THE SYSTOLIC ARRAY
N+(A+1)*P+M	LAST RAY OF THE LAST DATA STREAM ENTER A SEPARATE CELL PROCESSOR
N+(A+1)*P+M+K	LAST RAY OF THE LAST DATA STREAM EXITS A SEPARATE CELL PROCESSOR

Based on the above diagram, the total time to build P binary trees of length L is: N + (2**(L+1) - 3)*P + M + K. Not every ray of the second and subsequent data streams will intersect with an object in the scene (I.E., NOT ALL THE TREES WILL BE BALANCED). Therefore, the separate cell processor must be delayed and remain idle for K units. The current binary tree number still needs to be incremented, but nothing will be added to the binary tree. Also, when the binary trees are being established, those nonintersecting pixel rays, passing through a pixel, are determined by separate special purpose hardware, and sent immediately to the host for shading.

2.3.2.2 QUADRIC SURFACES

The design and the analysis of the operation of a systolic based architecture for implementing ray tracing with a global illumination model, for objects in a scene defined by quadrics is similar to the systolic based architecture for supporting a scene approximated by polygons. The differences are due to the longer time needed to perform the intersection calculation, and to the different method of determining the normal to the surface at the point of intersection. The inputs and the outputs to each cell are the same. The cell is preloaded with a different instruction sequence to calculate the t values. The separate cell processors are preloaded with a different instruction sequence to determine the reflection and refraction rays. The determination of the t values are described in an earlier section.

The calculation of the normal is based on the following analysis. Given point f(X,Y,Z) and i,j,k orthogonal unit vectors, grad (f) evaluated at a point (X0,Y0,Z0), yields the normal to the surface at that point.

GRAD(F) IS DEFINED AS: $\frac{\partial F}{\partial X} I + \frac{\partial F}{\partial Y} J + \frac{\partial F}{\partial Z} K$

The normalized vector is obtained by dividing the i,j,k components by the length of the vector. Given the general form of a quadric:

$$F = AX^{**}2 + BY^{**}2 + CZ^{**}2 + DXY + EYZ + FXZ + GX + HY + IZ + J = 0$$

$$\frac{\partial F}{\partial X} = 2AX + DY + FZ + G$$

$$\frac{\partial F}{\partial Y} = 2BY + DX + EZ + H$$

$$\frac{\partial F}{\partial Z} = 2CZ + EY + FX + I$$

Each partial of f is evaluated at the intersection point. Then the length of the vector is calculated as shown earlier. Once, the normal has been found, the calculations proceed as before in determining the reflection and refraction rays. The corresponding assembly like language program to compute the normal vector using the gradient is given in Appendix D.

2.3.2.2.1 ANALYSIS OF THE USE OF THE SYSTOLIC ARRAYS

The determination of the time needed for performing the intersection calculation, and for building the binary

shading trees is identical to polygon approximation of a scene. Again, the differences are due to the time, to perform the necessary intersection calculation, and to the different method of vector normal calculation.

2.3.2.3 COMPOSITE SURFACES

2.3.2.3.1 HETEROGENEOUS SCENE

At the time the first t value is generated, the corresponding partially built tree is brought into the memory of the next available separate cell processor. The tree is updated according to the t value, and the calculations of the reflection and refraction rays is an integer multiple of the time to perform the corresponding intersection calculation. The time of the intersection calculation varies according to the type of the intersected surface. The heterogeneous scene does not appear to be amenable to execution on systolic based architectures. Each surface type would require a different clock and a different basic unit of time for the intersection calculation. There would have to be a separate series of cell processors for each surface type. The reflection/refraction rays would appear to be generated at random time intervals. The time bottle neck would be the surface requiring the longest time to perform the calculations. The regular calculations are

giving way to asynchronous calculations. The complex timing scheme makes it difficult to determine when the next data stream would be ready to circulate through the systolic arrays. The architecture, to support the intersection calculations, is no longer systolic or regular. Rather, the asynchronous oriented calculations require a more sophisticated architecture and timing scheme. This architecture includes the use of interrupts to indicate when the next data stream is to be circulated through the systolic array. The components and calculations are not regular.

2.4 ANALYSIS OF THE APPLICATION OF SYSTOLIC ARRAYS TO RAY TRACING

The use of systolic arrays for the intersection calculations in ray tracing has limited applicability. The successful application of the systolic architecture to the intersection calculations that are performed as a part of ray tracing, depend on certain characteristics of the scene. These characteristics are:

1- the degree of object homogeneity

2- the mathematical representation of the objects

3- the internal representation of the entire scene

When systolic arrays have been used to perform image processing calculations such as convolution, the

calculations are a straightforward applications of arithmetic formulae on each pixel and its neighborhood. Minimal decision making is involved, and no exceptions need to be taken into account when the calculations are made. Once the values enter the array the computations proceed without interruption until completed. The regularity comes from the repeated execution of the same instruction sequence. This regularity can sometimes be achieved with ray tracing.

For hidden surface removal, the application of systolic arrays is feasible. A homogeneous scene, that is, a scene consisting of objects with the same mathematical description, is exemplified by a scene approximated by planar polygons. All the surfaces require that the same ray/surface intersection calculations be performed. The polygon surface coefficients should be sent to the cells of the array without regard to the hierarchy of the objects and surfaces in the scene. While the polygon aproximation of a scene makes the scene amenable to execution on systolic array, the polygon approximation precludes the derivation of the accurate surface properties for a more realistic rendering of the scene, when ray tracing is combined with a global illumination model.

Regularity of calculations alone does not guarantee suitability for execution on systolic arrays. The representation of the objects in a scene by parallelpipeds certainly simplifies and standardizes the ray/surface calculations, but the parallelpipeds are maintained in a hierarchy to reduce the number of parallelpiped-ray intersection calculations that must be performed for a given ray. The hierarchy representation is a software technique to speed up the display of the scene. This technique obviates the need for systolic arrays. This is not to say that the parallelpiped representation would not benefit from execution on a different supporting architecture. The hierarchial representation does not require that all of the parallelpiped-ray intersection calculations be performed, which can be done efficiently with a systolic array.

A scene consisting of all quadrics could have the planar objects as degenerate or special cases. The same calculations will have to be performed with no optimization for the planar objects, in order to maintain the regularity of calculation. The same amount of time is used to determine the plane/ray intersection calculation, as well as, the quadric/ray intersection calculation, even though the plane/ray intersection calculations are

simpler and shorter. Therefore, some of the cells in the array will perform unneccesary calculations, but the regularity in the calculations will permit the use of a systolic array for a heterogeneous scene. A scene consisting solely of polygons and quadrics may not always be interesting or realistic.

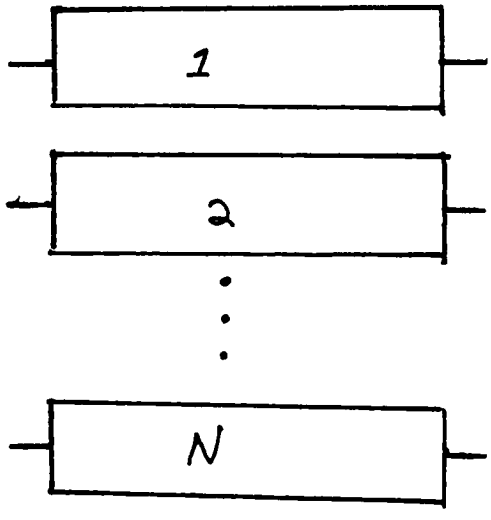
Scenes depicting objects represented as polynomials of degree greater than two, or as parametric patches and surfaces, require numerical methods to perform the surface ray intersection calculations. Numerical methods are iterative, rather than straight forward, one time applications of formulae. Hence, numerical methods are not regular calculations for implementation on systolic arrays.

When ray tracing is incorporated with a global illumination model, the criteria for execution on the systolic arrays are harder to meet. Scenes that are ray traced for hidden surface removal may not be ray traced with a global illumination model when the systolic arrays are used to perform the intersection calculations. This is the case for a heterogeneous scene, as described in an earlier section.

The systolic based architecture can designed to

achieve different optimal times for the intersection calculations performed in the ray tracing algorithms. In general, the reduced time for the intersection calculations is obtained by duplicating the systolic architecture, as well as the data streams.

The execution time can be reduced to M*B, that is proportional to the number of surfaces in the scene by the following architecture.

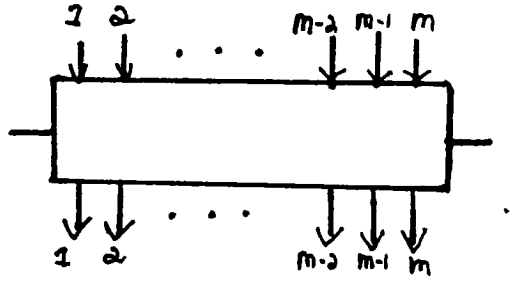


WHERE:

THE LENGTH OF EACH ARRAY IS M CELLS
N IS THE NUMBER OF PIXELS INVOLVED IN THE DISPLAY

The input to each array is one of the N pixel-ray parameters. The arrays execute simultaneously,each starting at the same time. Each pixel-ray circulates through a systolic array in M*B time units. For implementation with a global illumination model, all the needed binary trees will be generated simultaneously.

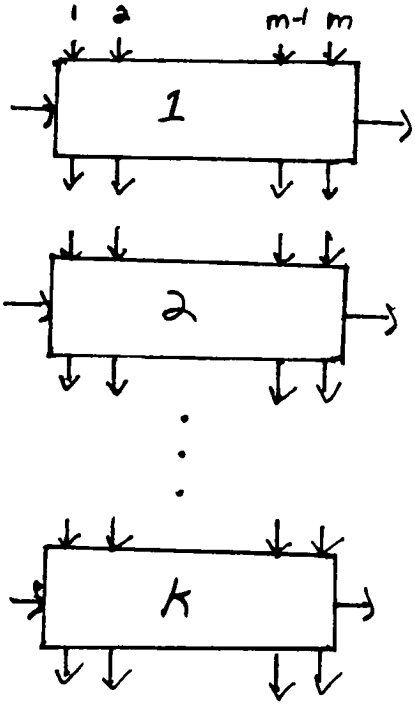
Hence, there will be no need for the current binary tree number. The number of separate cell processors matches the number of systolic arrays. After the first data stream has circulated through the systolic arrays, P of the N systolic arrays will generate valid t values. Therefore, only those P systolic arrays will be active on subsequent intersection calculations. The execution time can be reduced to K*B. The supporting architecture is as follows:



There is one systolic array with M input and output lines. The input to the systolic array is the first M pixel-rays, at time t = 0. At time, t = 1, the next m pixel-rays are input to the systolic array. Each input of m pixel-rays requires b units of time to perform the intersection calculations. The value of K is $\lfloor (N/M) + 1$. There is a need to make an association between the group of m and the position within that group, and the corresponding current binary tree number. Again, not every input line will be active on subsequent intersection calculations. If the time to build one

branch of the binary tree is greater than the time to perform the intersection calculation then, the generated t values will queue up outside the separate cells unless, there are N separate cell processors.

The execution time for the intersection calculations can be reduced to B units of time with the following architecture.



WHERE:

$$K \leftarrow L(N/M) + 1$$

There are K systolic arrays. The input to each of the

systolic arrays is a diferent group of the M pixel-rays. The systolic arrays are given input at the same time. The time for each systolic array to perfrom the intersection calculations is B time units.

CHAPTER 3

SUMMARY AND CONCLUSION

Many different representations of objects have been realistically synthesized on a computer using a variation of the ray tracing algorithm. Those representations, that make the intersection calculations of ray tracing amenable to execution on systolic arrays, are those representations that use analytical methods to find the intersection points. The analytical methods yield a solution in a predetermined amount of time, and are regular calculations. This situation is exemplified by a scene approximated by planar polygons. The value of t is found through an analytical method that can be executed by a cell as a sequence of instructions.

For a homogeneous scene, the time to perform the intersection calculations is linearly dependent on the length of the data stream, that is, on the number of pixels involved with the display, and on the number of surfaces on the scene. When a global illumination model is implemented with the ray tracing algorithm to synthesize a homogeneous scene, the length of time to perform the intersection calculations is linearly dependent on the number and depth of the binary trees, as well as, on the number of surfaces in the scene and the

length of the data stream. The length of time to perform the intersection calculations can be reduced by duplicating the input data stream and the systolic arrays.

CHAPTER 4

SUGGESTIONS FOR FURTHER RESEARCH

Once the systolic based architecture has been designed, the architecture could be simulated in software to uncover any unforeseen flaws and problems. The simulation of the operation of the architecture could also be used to more accurately calculate the time needed for the intersection calculations. At this time, the microcode used to implement the intersection calculations could be more finely tuned. After the design of the systolic architecture has been verified, the architecture could be implemented, used and tested. The optimized architectures presented in an earlier section could also be designed, verified and implemented in a similar fashion.

The area of computer graphics, including ray tracing has been using special purpose architecture to facilitate the accompanying extensive calculations. The viability of incorporation of the systolic architecture into a more encompassing special purpose architectures could be a worthwhile endeavor. This incorporation could result in a simplification of the overall architecture. For instance, systolic architectures have been implemented with

reconfigurable architectures [JAL+85]. This architecture could be used to build a special purpose environment to facilitate the entire ray tracing algorithm. Included in such an architecture would be the systolic arrays to implement the intersection calculations.

APPENDIX A

GLOSSARY

ALIASING - WHEN AN IMAGE IS NOT SAMPLED FINELY OR FREQUENTLY ENOUGH, AND IT BECOMES IMPOSSIBLE TO COMPLETELY RECONSTRUCT THE ORIGINAL IMAGE.

BAND-PASS FILTER - A FILTER THAT SUPPRESSES FREQUENCIES OUTSIDE A GIVEN RANGE.

COLOR - PHYSIOLOGICAL PHENOMENON RELATED TO HUMAN RESPONSE TO DIFFERENT WAVELENGTHS IN THE VISIBLE ELECTROMAGNETIC SPECTRUM[BALL82]

DIFFUSELY REFLECTED LIGHT - LIGHT THAT HAS PENETRATED BELOW THE SURFACE AND THEN HAS RE-EMITTED, SCATTERED EQUALLY IN ALL DIRECTIONS; INTENSITY IS INDEPENDENT OF THE VIEWER POSITION.

GLOBAL ILLUMINATION MODEL - THE INTENSITY OF LIGHT REACHING AN OBSERVER WHICH IS DEPENDENT ON REFLECTED AND REFRACTED LIGHT RAYS, AND RAYS FROM LIGHT SOURCES.

HALF-PLANE(CLOSED) - ALL THE POINTS IN A PLANE ON ONE SIDE OF A LINE IN A PLANE AND THE POINTS ON THE LINE.

HIGH-PASS FILTER - A FILTER THAT SUPPRESSES LOW FREQUENCIES.

LOW-PASS FILTER - A FILTER THAT SUPPRESSES HIGH FREQUENCIES.

MIMD(MULTIPLE INSTRUCTION STREAM, MULTIPLE DATA STREAM)- CLASSIFICATION OF A COMPUTER ARCHITECTURE THAT CONSISTS OF MULTIPLE PROCESSORS, EACH EXECUTING A DIFFERENT INSTRUCTION SEQUENCE WITH A DIFFERENT DATA STREAM.

OCTREE - AN OBJECT SPACE SUBDIVISION TECHNIQUE

THAT RECURSIVELY SUBDIVIDES SPACE INTO OCTANTS UNTIL THE OCTANT IS EMPTY, OR CONTAINS A PREDEFINED PORTION OF THE OBJECT.

REGULARIZED BOOLEAN OPERATORS- IN SOLIDS MODELING THE BOOLEAN OPERATORS ARE DEFINED SO THAT ONLY OTHER SOLIDS ARE PRODUCED, RATHER THAN DANGLING LINES OR PLANES.

RENDERING - PROCESS OF GENERATING A REALISTIC IMAGE OR PICTURE.

SHADER - SOFTWARE THAT PERFORMS THE SHADING CALCULATIONS TO DETERMINE THE INTENSITY OF A PIXEL ACCORDING TO A GLOBAL ILLUMINATION MODEL.

SIMD(SINGLE INSTRUCTION STREAM, MULTIPLE DATA STREAM) - CLASSIFICATION OF A COMPUTER ARCHITECTURE THAT IS EXEMPLIFIED BY A SINGLE GLOBAL CONTROLLER THAT DRIVES MULTIPLE PROCESSING ELEMENTS WHICH EXECUTES THE SAME INSTRUCTION SEQUENCE ON

DIFFERENT DATA STREAMS.

SPECULARLY REFLECTED LIGHT - LIGHT REFLECTED FROM THE OUTER SURFACE OF AN OBJECT; SEEN AS HIGHLIGHTS ON A SHINY OBJECT AND ARE CONCENTRATED ALONG THE DIRECTION OF REFLECTION; MOVES WITH THE VIEWER.

TRADITIONAL RAY TRACING - ALGORITHM THAT SYNTHESIZES IMAGES BY TRACING A SINGLE RAY THROUGH EACH PIXEL INVOLVED IN THE DISPLAY AND COMPUTES THE INTERSECTION OF EACH RAY WITH EVERY SURFACE IN THE DISPLAY; NO SOFTWARE OR HARDWARE TECHNIQUES ARE USED TO REDUCE THE COMPUTATION TIME.

APPENDIX B

REFERENCES

[DUFF81]

M.J.B. DUFF AND S. LEVIALDI(EDITORS), LANGUAGES AND ARCHITECTURES FOR IMAGE PROCESSING, ACADEMIC PRESS,INC., 1981.

Each chapter describes a high level language that makes use of the architecture for particular image processing applications. Some chapters describe only languages or architectures.

[FU--82]

K.S. FU AND TADAO ICHIKAWA,(EDS.), SPECIAL PURPOSE COMPUTERARCHITECTURES FOR PATTERN PROCESSING, CRC PRESS,INC., 1982.

Each chapter describes a different architecture with applications pertaining to image processing and pattern recognition.

[FUJI86]

AKIRA FUJIMOTO, TAKAYUKI, AND KANSEI IWATA,

"ART: ACCELERATED RAY TRACING SYSTEM", IEEE CG&A,

7(4), APRIL 1986, PP. 16 - 27.

This article presents a software solution to reducing

the number of intersection calculations for a scene synthesized with tracing. The software solution includes a SEADS (Spatially Enumerated Auxiliary Data Structures), that models the scene as a 3-D raster grid, which is encoded as a hybrid octree. The SEADS is traversed by a 3DDDA (Three Dimensional Differential Analyzer), which incrementally generates the coordinates of 3-D pixels that are pierced by the ray.

[GLAS84]

ANDREW GLASSNER, "SPACE SUBDIVISION FOR FAST RAY TRACING",

IEEE CG&A, 4(10), OCTOBER 1984, PP. 15 - 22.

A software technique is presented in this article as one means to reduce the number of intersection calculations. The solution is based on representing the synthesized scene as an octree, taking advantage of the spatial coherency in the scene.

[GOLD71]

R.A. GOLDSTEIN AND R. NAGEL, "3-D VISUAL SIMULATION",

SIMULATION, JANUARY 1971, PP. 25 - 31.

This early article presents ray tracing as the modeling of the photographic process. Light from a single point source is traced to the surface of an object. The reflected component is caluculated and traced, hopefully, to the surface of the film in the camera. The film is the display of the graphics terminal. To reduce the number of rays that must be traced, the article suggests following the rays in reverse order from the display.

[HANR83]

P. HANRAHAN, "RAY TRACING ALGEBRAIC SURFACES",

COMPUTER GRAPHICS, 17(3), JULY 1983, PP. 83 - 90.

Surfaces described by polynomials of any positive, integer degree can be synthesized using the ray tracing technique. The intersection, between a ray and a nontrivial surface, is calculated using a hybrid technique (numerical and analytical) to find the roots of a polynomial within a certain tolerance.

[JALA85]

SUDKAR JALAMANCHIL AND J.K. AGGARWAL, "RECONFIGURATION STRATEGIES FOR PARALLEL ARCHITECTURES", COMPUTER, 18(2), DECEMBER 1985, PP. 44 - 61.

The article explains the mapping of tasks to parallel VLSI configurations including the qualifications of a task for mapping onto s systolic based architecture.

[KUNG82]

H.T. KUNG, "WHY SYSTOLIC ARCHITECTURES", COMPUTER, 15(1), JULY 1982 PP. 37 - 46.

This article gives detailed descriptions of different systolic architectures to implement the convolution calculation.

[MEAD80]

CARVER MEAD AND LYNN CONWAY, INTRODUCTION TO VLSI SYSTEMS, ADDISON-WESLEY PUBLISHING COMPANY, 1980, PP. 271 - 292.

A section in this book describes systolic architectures for implementing matrix manipulation operations.

[PLUN85]

DAVID PLUNKETT AND MICHEAL BAILEY,"THE VECTORIZATION OF A RAY- TRACING ALGORITHM FOR IMPROVED EXECUTION SPEED", IEEE CG&A 5(8), AUGUST 1985, PP. 52 - 60.

The implementation of the ray tracing algorithm on a SIMD architecture is the topic of this article.

[ROGE85]

DAVID ROGERS, PROCEDURAL ELEMENTS FOR COMPUTER GRAPHICS, MCGRAW-HILL, 1985.

One section of the book describes ray tracing as a hidden surface removal algorithm; another section has

a detailed implementation of ray tracing with a global illumination model.

[ROTH82]

S.D. ROTH, "RAY CASTING FOR MODELING SOLIDS", COMPUTER GRAPHICS & IMAGE PROCESSING, 18(2), FEBRUARY 1982, PP. 109 - 144.

This article explains the use of ray tracing of CSG-like solids for rendering, and for finding the volume and center of these solids.

[RUBI80]

S.M. RUBIN AND T. WHITTED, "A THREE DIMENSIONAL REPRESENTATION FOR FAST RENDERING OF COMPLEX SCHEMES", COMPUTER GRAPHICS, 19(3), JULY 1980, PP. 110 - 116.

This article details a method for presenting objects with with parallepipeds of varying size, down to a point. The representation simplifies the ray intersection calculation, but requires the maintenance of a hierarchy of the parallelpipeds.

[SYMA85]

DAVID A. SYMANOW, DONALD J. SVETKOFF AND PATRICK F. LEONARD, "SYSTOLIC ARCHITECTURES FOR MACHINE VISION", ELECTRONIC IMAGING, FEBRUARY 1985, PP. 46 - 53.

This article describes the architecture of a machine vision system that incorporates VLSI systloic technology.

[UHR-84]

LEONARD UHR, ALGORITHM STRUCTURED COMPUTER ARRAYS AND NETWORKS: ARCHITECTURES FOR IMAGING, PERCEPTS, MODELS AND INFORMATION, ACADEMIC PRESS, INC., 1984.

The book describes the trend toward parallel architectures evolving as a means to solving compute intensive problems that exceed the capability of single processor computers.

[UMEO85]

HIROSHI UMEO," A CLASS OF SIMD MACHINES SIMULATED BY SYSTOLIC SYSTOLIC VLSI ARRAYS",IN VLSI:ALGORITHMS AND ARCHITECTURES, P. BERTOLAZZI AND F. LUCCIO(EDITORS), NORTH HOLLAND, 1985, PP. 39 - 48.

Theorems explaining the mapping from SIMD architectures to systolic array technology, and detailing the corresponding amount of time neede to perform the SIMD operations on a systolic array, are discussed in this article.

[WHIT80]

TURNER WHITTED, "AN IMPROVED ILLUMINATION MODEL FOR SHADED DISPLAY", CACM, 23(6), JUNE 1980, PP. 343 - 349.

The use of a global illumination model in conjunction with with the ray tracing of polygon and bicubic surfaces is the topic of this article.

[WYVI86]

GEOFF WYVILL, TSIYASU L. KUNII AND YASUTO SHIRAI, "SPACE DIVISION FOR RAY TRACING IN CSG", IEEE CG&A, 7(4), APRIL 1986, PP. 28 - 33.

This article depicts the use of ray tracing in

conjunction with a solids modeling system, based on Constructive Solid Geometry(CSG). Each solid is represented as a Directed Acyclic Graph(DAG) with the bottom nodes as the primitive solids, and the interior nodes representing the Boolean operations, used to build the final object. The DAG is modified to remove voxels below certain resolution, and to contain the final orientation and position of each primitive, yielding a tree. Through the use of spatial coherency the number of intersection calculations is reduced.

APPENDIX C

BIBILIOGRAPHY

[AMAN84]

J.AMANTIDES,"RAY TRACING WITH CONES",
COMPUTER GRAPHICS,18(3),JULY 1984, PP. 129 - 135.

[APPE67]

A. APPEL,"THE NOTION OF QUANTITATIVE INVISIBILITY AND THE MACHINE RENDERING OF SOLIDS", PROC. ACM 14, 1967, PP. 387 - 393.

[APPE68]

A. APPEL,"SOME TECHNIQUES FOR SHADING MACHINE RENDERINGS OF SOLIDS", AFIPS, 1968 SJCC, PP. 37 - 45.

[ATH78A]

PETER ALTHERTON,"POLYGON SHADOW GENERATION WITH APPLICATION TO SOLAR RIGHTS", MASTERS THESIS, CORNELL UNIVERSITY,1978.

[ATH78B]

PETER ALTHERTON,KEVIN WEILER AND DONALD GREENBERG, "POLYGON SHADOW GENERATION", COMPUTER GRAPHICS, 12(3),1978,PP. 275 - 281.

[BALL82]

DANA H. BALLARD AND CHRISTOPHER M. BROWN, "COMPUTER VISION", PRENTICE-HALL, INC., 1982, PP. 31.

[BALL81]

D.H. BALLARD,"STRIP-TREES: A HIERARCHIAL REPRESENTATION FOR CURVES", CACM, 24(5), MAY 1981, PP. 310 - 321.

[BARR81]

A.H. BARR,"SUPER QUADRICS AND THE ANGLE PRESERVING TRANSFORMATION", IEEE TRANSACTIONS ON COMPUTER GRAPHICS AND APPLICATIONS, 1(1), JANUARY 1981, PP. 11 - 23.

[BLIN76]

JAMES F. BLINN AND MARTIN NEWELL,"TEXTURE AND REFLECTION IN COMPUTER GENERATED IMAGES", CACM, OCTOBER 1976, PP. 542 - 547.

[BLIN77]

JAMES F. BLINN,"MODELS OF LIGHT REFLECTION FOR COMPUTER SYNTHESIZED PICTURES", COMPUTER GRAPHICS, 11(2), SUMMER 1977, PP. 192 - 198.

[BLIN78A]

JAMES F. BLINN,"SIMULATION OF WRINKELED SURFACES", COMPUTER GRAPHICS, 12(3), JULY 1978, PP. 286 - 292.

[BLIN78B]

JAMES F. BLINN,NOTES ON GEOMETRIC MODELING, SIGGRAPH '80 TUTORIAL.

[BLIN80]

JAMES F. BLINN, L.C. CARPENTER, J.M. LANE, AND T. WHITTED, "SCAN LINE METHODS FOR DISPLAYING PARAMETRICALLY DEFINED SURFACES", CACM, 23(1), 1980, PP. 23 - 24.

[BLIN82]

JAMES F. BLINN,"A GENERALIZATION OF ALGEBRAIC SURFACE DRAWING", ACM TOG, 1(3), JULY 1982, PP. 235 - 236.

[BOUC70A]

JACK BOUCKNIGHT AND KARL C. KELLY,"AN ALGORITHM FOR PRODUCING HALF-TONE COMPUTER GRAPHICS REPRESENTATIONS WITH SHADOWS AND MOVEABLE LIGHT

SOURCES",SJCC, 1970, AFIPS, VOL. 36, PP. 1 - 10.

[BOUC70B]

JACK BOUCKNIGHT,"A PROCEDURE FOR GENERATION OF THREE DIMENSIONAL HALF-TONED COMPUTER GRAPHICS PRESENTATIONS", CACM, VOL. 13, 1970, PP. 527 - 536.

[BOUV85]

CHRISTIAN BOUVILLE, " BOUNDING ELLIPSOID FOR RAY-FRACTAL INTERSECTIONS", COMPUTER GRAPHICS, 19(3), JULY 1985, PP. 45 - 51.

[BRAI73]

I.C. BRAID,"DESIGNING WITH VOLUMES",DOCTORAL THESIS, UNIVERSITY CAMBRIDGE, ENGLAND(1973).

[BRON85]

WILLIAM F. BRONVORT AND FOPKE KLOK, "RAY TRACING GENERALIZED CYLINDERS", ACM TOG, 4(4), OCTOBER 1985,

PP. 291 - 302.

[BUI-75]

PHONG BUI-TUONG,"ILLUMINATION FOR COMPUTER GENERATED IMAGES",CACM, 18(6), JUNE 1975, PP. 311 - 317.

[CARP80]

LOREN C. CARPENTER,"COMPUTER RENDERING OF FRACTAL CURVES AND SURFACES", PP. 1 - 8, SUPPL. TO PROC. SIGGRAPH 80, AUGUST 1980.

[CASA85]

MALCOLM S. CASALE AND EDWARD L. STANTON, "AN OVERVIEW OF ANALYTIC SOLID MODELING", IEEE CG&A, 5(2), FEBRUARY, 1985, PP. 45 - 56.

[CATM74]

EDWIN CATMULL,"A SUBDIVISION ALGORITHM FOR COMPUTER DISPLAY OF CURVED SURFACES", DOCTORAL THESIS, UNIVERSITY OF UTAH, 1974.

[CATM75]

EDWIN CATMULL,"COMPUTER DISPLAY OF CURVED SURFACES", PROC. IEEE CONF. ON COMPUTER GRAPHICS, PATTERN RECOGNITION, AND DATA STRUCTURES, MAY 1975, PP. 11 - 17.

[CATM78]

EDWIN CATMULL," A HIDDEN SURFACE ALGORITHM WITH ANTI-ALIASING COMPUTER GRAPHICS, 12(3), AUGUST 1978, PP. 6 - 11.

[CHOW78]

Y.S.CHOW AND H.TEICHER, PROBABILITY THEORY: INDEPENCE, INTERCHANABILITY, MARTINGATES, VERLAG HEIDELBERG,1978.

[CLAR76]

J.H.CLARK,"HIERARCHIAL GEOMETRIC MODELS FOR VISIBLE SURFACE ALGORITHMS", CACM, 19(10), OCTOBER 1976, PP. 547 - 554.

[CLAR82]

JAMES CLARK,"THE GEOMETRICAL MACHINE: A VLSI SYSTEM FOR GRAPHICS, COMPUTER GRAPHICS , 16(3), JULY 1982, PP. 127 - 133.

[CLEA83]

WYVILL CLEARY, BIRTWISTLE, AND VATTI, "MULTIPROCESSOR RAY TRACING TECH. REP. NO. 83/128/17, DEPT. OF COMPUTER SCIENCE, THE UNIVERSITY OF CALGARY, OCTOBER 1983.

[COLL76]

G.E. COLLINS AND A.G. ARITAS,"POLYNOMIAL REAL ROOT ISOLATION USING DESCARTES' RULE OF SIGNS", PROC. 1976 ACM SYMPOSIUM ON SYMBOLIC AND ALGEBRAIC COMPUTATION, 1976, PP. 272 - 276.

[COLL82]

G.E. COLLINS AND R. LOOS,"REAL ZEROS OF POLYNOMIALS", COMPUTING SUPPL. 4, 1982, PP. 83 - 94.

[COOK82A]

ROBERT COOK,"A REFLECTION MODEL FOR REALISTIC IMAGE SYNTHESIS", MASTERS THESIS,CORNELL UNIVERSITY(1982).

[COOK82B]

ROBERT COOK AND K.E. TORRENCE,"A REFLECTANCE MODEL FOR COMPUTER GRAPHICS",ACM TOG, 1(1), 1982, PP. 7 - 24.

[COOK83]

ROBERT COOK,"ANTI-ALIASING POINT SAMPLES",TECHNICAL MEMO NO. 95, LUCASFILM, LTD., SAN RAFAEL, CA, OCTOBER 1983.

[COOK84]

ROBERT COOK, THOMAS PORTER AND LOREN CARPENTER, "DISTRIBUTED RAY TRACING",COMPUTER GRAPHICS, 18(3), JULY 1984, PP 137 - 145.

[COOK--]

ROBERT L. COOK, TURNER WHITTED, AND DONALD GREENBERG, "A COMPREHENSIVE MODEL FOR IMAGE SYNTHESIS", UNPUBLISHED REPORT.

[COQU84]

SABINE COQUILLART AND MICHEL GANET, "SHADED DISPLAY OF DIGITAL MAPS", IEEE CG&A 4(7), JULY 1982, PP. 35 - 42.

[CROW77A]

F. CROW,"SHADOW ALGORITHMS FOR COMPUTER GRAPHICS", COMPUTER GRAPHICS, 11(3), JULY 1977, PP. 242- 248.

[CROW77B]

F. C. CROW,"THE ALIASING PROBLEM IN COMPUTER GENERATED SHADED IMAGES", CACM, 20(11), NOVEMBER 1977, PP. 799 - 805.

[DADO83]

NORM DADOUN, DAVID KIRKPATRICK, AND JOHN WALSH,

"HIERARCHIAL APPROACHES TO HIDDEN SURFACE INTERSECTION TESTING ,PROC. OF GRAPHICS INTERFACE '82, MAY 1982, PP 49 - 56.

[DAVI82]

J. DAVIS, M. BARLEY, AND D. ANDERSON, "PROJECTING REALISTIC IMAGES OF GEOMETRIC SOLIDS", COMPUTERS IN MECHANICAL ENGINEERING, 1(1), AUGUST 1982, PP. 6 - 13.

[DIPP84]

MARK DIPPE AND JOHN SWENSEN," AN ADAPTIVE SUBDIVISION ALGORITHM AND PARALLEL ARCHITECTURE FOR REALISTIC SYNTHESIS", COMPUTER GRAPHICS, 18(3), JULY 1984, PP. 149 - 158.

[DIPP85]

MARK A. Z. DIPPE AND ERLIN HENRY WOLD, "ANTIALIASING THROUGH STOCHASTIC SAMPLING", COMPUTER GRAPHICS, 19(3), JULY 1985, PP. 69 - 78.

[DUDA73]

R.O DUDA AND P.E. HART, PATTERN RECOGNITION AND SCENE ANALYSIS, WILEY-INTERSCIENCE, NY, 1973.

[DUFF79]

T. DUFF,"SMOOTH SHADED RENDERING OF POLYHEDRAL OBJECTS ON RASTER DISPLAYS", COMPUTER GRAPHICS, VOL. 13, 1979, PP. 275 - 279.

[EDWA82]

B.E. EDWARDS, "IMPLEMENTATION OF A RAY TRACING ALGORITHM FOR RENDERING SUPERQUADRIC SOLIDS", MASTERS THESIS, RPI, 1983.

[FOUR80]

ALAN E. FOURNIER AND DON FUSSEL,"STOCHASTIC MODELING IN COMPUTER GRAPHICS", PP. 1 - 5, SUPPL. TO PROC. SIGGRAPH 80, AUGUST 1980.

[FOUR82]

A. FOURNIER, D. FUSSELL AND L. CARPENTER, "COMPUTER RENDERING OF STOCHASTIC MODELS", CACM, 25(6), JUNE 1982, PP. 371 - 384.

[FRIS85]

PATRICE FRISON AND PATRICE QUINTON, " AN INTEGRATED SYSTOLIC SYSTOLIC MACHINE FOR SPEECH RECOGNITION", IN VSLI: ALGORITHMS AND ARCHITECTURES, P. BERTOLAZZI AND F. LUCCIO (EDS.), NORTH HOLLAND, 1985, PP. 175 - 176.

[FUCH81]

H. FUCHS AND B. POULTON,"A VLSI-ORIENTED DESIGN FOR A RASTER GRAPHICS ENGINE", VLSI DESIGN, NO. 3, 1981, PP. 20 - 28.

[GOUR71]

H. GOURAUD,"COMPUTER DISPLAY OF CURVED SURFACES", DOCTORAL THESIS, UNIVERSITY OF UTAH, 1971.

[GUERS5]

C. GUERRA AND T. KANADE, "A SYSTOLIC ALGORITHM FOR STEREO MATCHING", IN VLSI:ALGORITHMS AND ARCHITECTURES, P.BERTOLAZZI AND F.LUCCIO(EDS.), NORTH HOLLAND, 1985, PP. 103 - 112.

[HALL83A]

ROBERT A. HALL , "A METHODOLOGY FOR REALISTIC IMAGE SYNTHESIS", MASTERS THESIS, CORNELL UNIVERSITY(1983).

[HALL83B]

ROBERT HALL AND DONALD GREENBERG,"A TESTBED FOR REALISTIC IMAGE SYNTHESIS", IEEE CG&A, 3(8), NOVEMBER 1983, PP. 10 - 20.

[HANR84]

PAUL HANRAHAN AND PAUL HECKBERT, "INTRODUCTION TO BEAM TRACING", PROC. INT'L CONF. ON ENGINEERING AND COMPUTER GRAPHICS", BEIYING, CHINA, AUGUST 1984.

[HECK84]

P. HECKBERT AND P. HANRAHAN, "BEAM TRACING POLYGONAL OBJECTS", COMPUTER GRAPHICS,18(3), JULY 1984, PP. 119 - 127.

[HOOK84]

T. VAN HOOK, "ADVANCED TECHNIQUES FOR SOLID MODELING", COMPUTER GRAPHICS WORLD, 7(11), NOVEMBER 1984, PP. 45 -- 54.

[HUDS05]

RT.W.H.T. HUDSON, KRUMMER'S QUARTIC SURFACE,

CAMBRIDGE UNIVERSITY PRESS, 1905.

PRESS,1905.

[JONE71]

C.B. JONES, "A NEW APPROACH TO THE HIDDEN LINE PROBLEM", THE COMPUTER JOURNAL, 14(3), AUGUST 1971, PP. 232 - 237.

[KAJI82]

J. T. KAJIYA, "RAY TRACING PARAMETRIC PATCHES", COMPUTER GRAPHICS, JULY 1982, PP. 245 - 254.

[KAJI83A]

JAMES T. KAJIYA,"NEW TECHNIQUES FOR RAY TRACING PROCEDURALLY DEFINED OBJECTS", COMPUTER GRAPHICS, VOL. 17, 1983, PP. 91 - 102.

[KAJI83B]

JAMES T. KAJIYA, SIGGRAPH 83 TUTORIAL ON RAY TRACING, PROC. OF SIGGRAPH 83, COURSE 10 NOTES, 1983.

[KAJI84]

JAMES T. KAJIYA AND BRIAN VON HERZEN, "RAY TRACING VOLUME DENSITIES", COMPUTER GRAPHICS,18(3), JULY 1984, PP. 165 - 174.

[KAWA83]

YAICHIRO KAWAGUCHI,"GROWTH: MYSTERIOUS GALAXY", SIGGRAPH 83, FILM AND VIDEO SHOWS, P. 5.

[KAY-79A]

DOUGLAS S. KAY, "TRANSPARENCY, REFRACTION AND RAY TRACING FOR COMPUTER SYNTHESIZED IMAGES", CORNELL UNIVERISTY, JANUARY 1979.

[KAY-79B]

DOUGALS S. KAY AND DONALD GREENBERG, "TRANSPARENCY FOR

COMPUTER SYNTHESIZED IMAGES", COMPUTER GRAPHICS, 13(3), JULY 1979, PP. 158 - 164.

[KELL70]

KARL C. KELLY, "A COMPUTER GRAPHICS PROGRAM FOR GENERATION OF HALF-TONED IMAGES WITH SHADOWS", MASTERS THESIS, UNIVERSITY OF ILLINOIS, 1970.

[LANE79]

J.M. LANE AND L.C. CARPENTER,"A GENERATED SCAN LINE ALGORITHM FOR COMPUTER DISPLAY OF PARAMETRICALLY DEFINED SURFACES", CG&IP, VOL. 11, 1979, PP. 290 - 297.

[LEE-85]

MARK E. LEE, RICHARD A. REDNER AND SAMUEL USELTON, "STATISTICALLY OPTIMIZED SAMPLING FOR DISTRIBUTED RAY TRACING", COMPUTER GRAPHICS, 19(3), JULY 1985,

PP. 61 - 67.

[MAND68]

B. MANDELBROT, "FRACTAL BROWNIAN MOTIONS, FRACTIONAL NOISES AND APPLICATIONS", SIAM REVIEW, 10(4), OCTOBER 1968, PP. 422 - 437.

[MAND77]

B. MANDELBROT, FRACTALS: FORM, CHANCE AND DIMENSION, W.H. FREEMAN, SAN FRANCISCO, 1977.

[MAXW82]

GREG MAXWELL AND VICTOR GOLDSCHMITT, "AN APPLICATION OF THE RAY FIRING METHOD FOR DETERMINING RADIATION VIEW FACTORS", REPORT 0469-1, H2 83-35P, PURDUE UNIVERSITY, SCHOOL OF MECHANICAL ENGINEERING, 1(1), AUGUST 1982.

[MUAR83]

K. MUARKAMI AND H. MATSUMOTO, "RAY TRACING FOR CSG REPRESENTATION USING STATUS TREE TECHNIQUE", 27TH INFORMATION PROCESSING CONFERENCE, OCTOBER 1983, PP. 1535 - 1537.(IN JAPANESE)

[MEYE75]

ALLEN MEYERS, "AN EFFICIENT VISIBLE SURFACE PROGRAM", REPORT TO NSF, DIVISION OF MATHEMATICS AND COMPUTER SCIENCE, COMPUTER GRAPHICS RESEARCH GROUP, OHIO STATE UNIVERSITY, JULY 1975.

[NEWE72]

M.E. NEWELL, R.G. NEWELL, AND T.L. SANCHA," A SOLUTION TO THE HIDDEN SURFACE PROBLEM",PROC. ACM ANN. CONF., BOSTON, AUGUST 1972, PP. 443 - 450.

[NISH83]

H. NISHIMURA, H. OHNO, T KAWATA, I. SHIRAKAWA AND K. OMURA, "LINKS-1: A PARALLEL PIPELINED MULTIMICROCOMPUTER SYSTEM FOR IMAGE CREATION", PROC. 10TH SYMPOSIUM ON COMPUTER ARCHITERCTURE, 1983, 387 - 394.

[PAVI82]

THEO PAVILIDES, ALGORITHMS FOR GRAPHICS AND IMAGE PROCESSING,COMPUTER SCIENCE PRESS, 1982.

[POTM81]

M. POTMESIL AND I. CHAKRAVARTY,' A LENS AND APPERTURE CAMERA MODEL

CAMERA MODEL FOR SYNTHETIC IMAGE GENERATION', COMPUTER GRAPHICS, 15(3), MONTH, 1981, PP. 297 - 305.

[POTM82]

M. POTMESIL AND I. CHAKRAVARTY, "SYNTHETIC IMAGE GENERATION WITH A LENS AND APPERTURE CAMERA MODEL",

ACM TOG, 1( ), MONTH, 1982, PP. 83 - 102.

[POTM83]

M. POTMESIL AND I. CHAKRAVARTY,"MODELING MOTION BLUR IN COMPUTER GENERATED IMAGES, COMPUTER GRAPHICS, 17(3), JULY 1983, PP. 389 - 395.

[RALS65]

A. RALSTON, A FIRST COURSE IN NUMERICAL ANALYSIS, MCGRAW-HILL,1965.

[RALS78]

A. RALSTON AND P. RABINOWITZ, A FIRST COURSE IN NUMERICAL ANALYSIS, 2ND. ED., MCGRAW-HILL, NY, 2ND. ED., MCGRAW-HILL, NY, 19XX, PP. 380. - 383.

[REIC80]

A.A.G. REICHA, " REPRESENTATIONS FOR RIGID SOLIDS:

THEORY, METHODS AND SYSTEMS", ACM COMPUTING SURVEYS, VOL. 12(4), DECEMBER 1980, PP. 437 - 464.

[SEDE83]

T.W. SEDEBERG, "IMPLICIT PARAMETRIC CURVES AND SURFACES FOR COMPUTER AIDED GEOMETRIC DESIGN, PH.D. THESIS. SCHOOL OF MECHANICAL ENGINEERING, PURDUE UNIVERSITY, AUGUST 1983.

[SEDE84]

THOMAS SEDERBERG AND DAVID ANDERSON, "RAY TRACING OF STEINER PATCHES", COMPUTER GRAPHICS 18(3), JULY 1984, PP. 159 - 161.

[SEYB84]

H. SEYBOLD, "CONSTRUCTION OF FUNCTIONS FOR REPRESENTATION OF SURFACES", IN SURFACES IN COMPUTER-AIDED GEOMETRICAL DESIGN(COPENHAGEN, SEPTEMBER 12 - 14), R. E. BARNHILL AND W. BOELM,

EDS.,NORTH HOLLAND, AMSTERDAM, 1984, PP. 73 - 82.

[STEI84]

HERBERT STEINBERG,"A SMOOTH SURFACE BASED ON BIQUADRATIC PATCHES", IEEE CG&A, 9(9), SEPTEMBER, 1984, PP. 20 - 23.

[SWEE86]

MICHAL A.J. SWEENY AND RICHARD H. BARTIES,"RAY TRACING FREE FORM B-SPLINE SURFACES", IEEE CG&A, 6(2), FEBRUARY 1986, PP. 41 - 49.

[TORR67]

K.E. TORRENCE AND E.M. SPARROW, "THEORY OF OFF-SPECULAR REFLECTION FROM ROUGHENED SURFACES", J. OF OPTICAL SOCIETY OF AMERICA, VOL. 57, PP. 1105 - 1114, 1967.

[TOTH85]

DANIEL L. TOTH, "ON RAY TRACING PARAMETRIC SURFACES",

COMPUTER GRAPHICS, 19(3), JULY 1985, PP. 171 - 179.

[ULLN83]

M.K. ULLNER, "PARALLEL MACHINES FOR COMPUTER GRAPHICS", PH.D. THESIS, CAL. OF TECHN., PASADENA, CA., 1983.

[USPE48]

J.V. USPENSKY, THEORY OF EQUATIONS, MCGRAW-HILL, 1948.

[VERB84]

C. VERBECK AND DONALD GREENBERG,"A COMPREHENSIVE LIGHT SOURCE DESCRIPTION FOR COMPUTER GRAPHICS REPRESENTATIONS", IEEE CG&A, 4(7), JULY 1984, PP. 66 - 75.

[WALK50]

R.J. WALKER, ALGEBRAIC CURVES, SPRINGER VERLAG, 1950.

[WARN83]

DAVID R. WARN, "LIGHTING CONTROLS FOR SYNTHETIC IMAGES", COMPUTER GRAPHICS, 17(3), MONTH, 1983, PP. 18 - 21.

[WARN69]

J. WARNOCK," A HIDDEN-SURFACE ALGORITHM FOR COMPUTER GENERATED HALF-TONED PICTURE", UNIVERSITY OF UTAH, COMP. SCI. DEPT., TR 4 - 15, 1969, NTIS, AD-573 671.

[WEGH84]

HANK WEGHORST, GARY HOOPER, AND DONALD GREENBERG, "IMPROVED COMPUTATIONAL METHODS FOR RAY TRACING", ACM TOG, 3(1), JANUARY 1984, PP. 52 - 69.

[WEIN81]

RICHARD WEINBERG,"PARALLEL PROCESSING IMAGES SYNTHESIZED ANTI-ALIASING", COMPUTER GRAPHICS, 12( ), AUGUST 1981, PP. 56 - 62.

[WIJK84A]

J.J.VAN WIJK, "RAY TRACING OBJECTS DEFINED BY SWEEPING A SPHERE",

A SPHERE", PROC. OF EUROGRAPHICS 84 CONF., K. BO AND

H.A. TUCKER,EDS., NORTH HOLLAND, AMSTERDAM, 1984,

PP. 73 - 82.

[WIJK84B]

J.J. VAN WIJK, "RAY TRACING OBJECTS DEFINED BY SWEEPING

PLANAR CUBIC SPLINES", ACM TOG, 3(3), JULY 1984,

PP. 223 - 237.

[WILL78]

LANCE WILLIAMS, "CASTING CURVED SHADOWS ON CURVED

SURFACES", COMPUTER GRAPHICS, 12(9), 1978,

PP. 270 - 274.

[WILL83]

LANCE WILLIAMS, "PYRAMIDAL PARAMETRICS", COMPUTER

GRAPHICS, 17(3), JULY 1983, PP. 1 - 11.

Missing Page

APPENDIX D

The assembly-like instructions that must be implemented by a systolic cell to perform the intersection calculations for a planar surface.

```
LOAD.REG R1,A
MULT.REG R1,XO
STORE.MEM R1,MEM1              AXO

LOAD.REG R1,B
MULT.REG R1,YO
STORE.MEM R1,MEM2              BYO

LOAD.REG R1,C
MULT.REG R1,ZO
STORE.MEM R1,MEM3              CZO

LOAD.REG R1,A
MULT.REG R1,E
STORE.MEM R1,MEM4              AE

LOAD.REG R1,B
MULT.REG R1,F
STORE.MEM R1,MEM5              BF

LOAD.REG R1,C
MULT.REG R1,G                  CG

ADD R1,MEM5
ADD R1,MEM4                    AE + BF + CG

LOAD.REG R3,D
ADD R3,MEM3
ADD R3,MEM2
ADD R3,MEM1                    AXO + BYO + CZO + D

DIVIDE R3,P1                   T

CMP     R1,0
BNEG    *+K       NEGATIVE CALCULATED T VALUE

LOAD.REG R2,T_IN
CMP     R1,R2
BEQ     *+K-3
```

```
    BEQ     *+K-3   CALCULATED T VALUE SAME AS INPUT T
    BGT     *+K-4   CALCULATED T GREATER THAN INPUT T
    BZR     *+K-5   CALCULATED T VALUE ZERO
;
;
;   ASSUME CALCULATED T VALUE IS POSITIVE AND LESS
;   THAN THE INPUT T VALUE
;
    MOVE.REG R1,T_OUT
    MOVE.REG  ID_IN,ID_OUT
    BR      * + 3
;
;   BRANCH HERE IF THE INPUT T VALUE IS TO BE USED
;
    MOVE.REG  T_IN,T_OUT
    MOVE.REG  ID_IN,ID_OUT
```

The following assembly-like instructions are performed by the sepaarte single cell in order to determine the point of intersection with a planar surface in the scene. Then, a check is made by the separate single to make sure the point is within the extents defined for the surface.

```
    CMP   ID_IN,'INVALID'
    BEQ   *+K                   NON-INTERSECTING RAY

    LOAD.REG     R1,T
    MULT.REG     R1,E
    ADD.REG      R1,XO
    STORE.MEM    R1,MEM1        XO + ET

    LOAD.REG     R1,T
    MULT.REG     R1,F
    ADD.REG      R1,YO
    STORE.MEM    R1,MEM2        YO + FT
```

```
LOAD.REG     R1,T
MULT.REG     R1,G
ADD.REG      R1,ZO
STORE.MEM    R1,MEM3    ZO + GT


ASSUME R7 CONTAINS THE BASE ADDRESS OF THE TABLE
       R6 CONTAINS L, THE SIZE OF EACH ENTRY IN THE TABLE

LOAD.REG     R1,R7
SUBT.REG     R1,R6
LOAD.REG     R2,ID_IN
MULT.REG     R2,R6
ADD.REG      R3,R1,R2  POINT TO THE ENTRY FOR THE ID

LOAD.REG     R4,0(R3)  GET XMIN
LOAD.REG     R5,1(R3)  GET XMAX
LOAD.REG     R8,MEM1   GET X COORDINATE
CMP          R8,R4
BLT          *+K       LESS THAN XMIN
CMP          R8,R5
BGT          *+K-C     GREATER THAN XMAX


X COORDINATE WITHIN RANGE

LOAD.REG     R4,2(R3)  GET YMIN
LOAD.REG     R5,3(R3)  GET YMAX
LOAD.REG     R8,MEM2   GET Y COORDINATE
CMP          R8,R4
BLT          *+K-D     LESS THAN YMIN
CMP          R8,R5
BGT          *+K-D-2   GREATER THAN YMAX


Y COORDINATE WITHIN Y EXTENT

LOAD.REG     R4,4(R3)  GET ZMIN
LOAD.REG     R5,5(R3)  GET ZMAX
LOAD.REG     R8,MEM3   GET Z COORDINATE
CMP          R8,R4
BLT          *+K-E     LESS THAN ZMIN
CMP          R8,R5
BGT          *+K-E-2   GREATER THAN ZMAX


Z COORDINATE WITHIN Z EXTENT

LOAD.REG     ID_OUT,ID_IN
BR           *+2
```

```
;     INVALID COORDINATE OR INVALID INPUT ID
;
      LOAD.REG    ID_OUT,'INVALID'
      LOAD.REG    X_OUT,MEM1
      LOAD.REG    Y_OUT,MEM2
      LOAD.REG    Z_OUT,MEM3
```

The following assembly-like instructions are performed by a systolic cell to find the parametric intersection value of a ray and a surface defined by a polynomial of degree two.

```
      LOAD.REG   R1,B
      MULT.REG   R1,B    B**2

      LOAD.REG   R2,A
      MULT.REG   R2,C
      SHFT,LEF   R2,2    4AC
      SUBT.REG   R1,R2   B**2 - 4AC

      BRCH.NEG   * + K   STORE A COMPLEX NUMBER IDENTIFIER
;
;     ALGORITHM NEEDED TO  CALCULATE THE SQUARE ROOT
;     OR A LOOKUP TABLE
;     TO RETRIEVE THE VALUE -- ASSUME THE VALUE IS IN R1
;
      LOAD.REG   R2,A
      SHFT.LEF   R2,1    2A

      LOAD.REG   R3,B
      NEG.REG    R3      -B

      LOAD.REG   R4,R3
      ADD.REG    R4,R1   -B + ( B**2 - 4AC)**1/2
      DIV.REG    R4,R2   ( -B + ( B**2 - 4AC)**1/2)/2A
      STOR.MEM   R2,MEM1

      LOAD.REG   R4,R3
      SUBT.REG   R4,R1   -B - ( B**2 - 4AC)**1/2
      DIV.REG    R4,R2   (-B - ( B**2 - 4AC)**1/2
      STOR.MEM   R2,MEM2
;
```

```
;     COMPARE THE CALCULATED AND THE INPUT T VALUES
;
      LOAD.REG   R1,MEM1
      CMP        R1,0
      BLE        *+4
      LOAD.REG   R2,MEM2
      CMP        R2,0
      BLE        *+4-3
      CMP        R2,0
      BNE        *+
      STORE.MEM  R1,MEMT
      BR         *+
      CMP        R1,R2
      BNEG       *+3

;
;     T2 < T1
;

      STORE.MEM  T2,MEMT
      BR         *+1
;
;     T1 < T2
;
      STORE.MEM  T1,MEMT
      CMP        T_IN,'VALID'
      BNE        *+
;
;     COMPARE T_IN WITH MEMT
;
      LOAD.REG   R3,MEMT
      CMP        R3,T_IN
      BLT        *+
;
;     T_IN <= CALCULATED T
;
      LOAD.REG   T_OUT,T_IN
      LOAD.REG   ID_OUT,ID_IN
      BR         *+
;
;     CALCULATED T < T_IN
;
      LOAD.REG   T_OUT,R3
      LOAD.REG   ID_OUT,OWN_ID
      BR         *+
;
;     T_IN INVALID
;
      LOAD.REG   T_OUT,MEMT
```

```
        LOAD.REG   ID_OUT,OWN_ID
        BR         *+
;
;       T2 <= 0 AND T1 > 0
;
        CMP        T_IN,'VALID'
        BNE        *+
        LOAD.REG   R1,MEMT
        CMP        R1,T_IN
        BLT        *+4
;
;       T_IN < MEMT
;
        LOAD.REG   T_OUT,T_IN
        LOAD.REG   ID_OUT,ID_IN
        BR         *+
;
;       MEMT < T_IN
;
        LOAD.REG   T_OUT,R1
        LOAD.REG   ID_OUT,OWN_ID
        BR         *+
;
;       T_IN INVALID
;
        LOAD.REG   T_OUT,MEMT
        LOAD.REG   ID_OUT,OWN_ID
        BR         *+
;
;       T1 AND T2 ARE INVALID
;
        LOAD.REG   T_OUT,T_IN
        LOAD.REG   ID_OUT,ID_IN
```

The following assembly-like instructions are performed by the single separate cells to compute the reflection and refraction rays for a global illumination model.

```
;
;       DETERMINE THE REFLECTION RAY    R = V' + 2N
;
```

```
;
;       CALCULATE   V'
;



        LOAD.REG      R1,MEM1
        LOAD.REG      R2,MEM2
        LOAD.REG      R3,MEM3

        SUB.REG       R1,PX
        SUB.REG       R2,PY
        SUB.REG       R3,PZ
;
;       SAVE:(AX-BX)I  ;  (AY-BY)J  ;  (AZ-BZ)K
;
        STOR.MEM      R1,MEM4
        STOR.MEM      R2,MEM5
        STOR.MEM      R3,MEM6
;
;       CALCULATE 2N
;
        LOAD.REG      R1,D
        LOAD.REG      R3,E
        LOAD.REG      R5,F

        LOAD.REG      R2,R1
        LOAD.REG      R4,R3
        LOAD.REG      R6,R5

        MULT.REG      R1,R2     D**2
        MULT.REG      R3,R4     E**2
        MULT.REG      R5,R6     F**2

        ADD.REG       R1,R3
        ADD.REG       R1,R5     D**2 + E**2 + F**2
        SQRT          R1,R2     L
        STOR,MEM      R2,L

        LOAD.REG      R1,MEM4
        DIVIDE        R1,R2,R3
        STOR.MEM      R3,MEMDL

       LOAD.REG       R1,MEM5
       DIVIDE         R1,R2,R3
       STOR.MEM       R3,MEMEL

       LOAD.REG       R1,MEM6
       DIVIDE         R1,R2,R3
```

```
        STOR.MEM    R3,MEMFL
;
;       CALCULATE   V.N
;
        LOAD.REG    R1,MEM4
        LOAD.REG    R2,MEMDL
        MULT.REG    R1,R2
        STOR.MEM    R1,MEM7

        LOAD.REG    R1,MEM5
        LOAD.REG    R2,MEMEL
        MULT.REG    R1,R2
        STOR.MEM    R1,MEM8

        LOAD.REG    R1,MEM6
        LOAD.R1     R2,MEMFL
        MUTL.REG    R1,R2
        STOR.MEM    R1,MEM9
;
;       CALCULATE ABS(V.N)
;

        LOAD.REG    R1,MEM7
        LOAD.REG    R2,R1
        MULT.REG    R1,R2
        LOAD.REG    R10,R1

        LOAD.REG    R1,MEM8
        LOAD.REG    R2,R1
        MULT.REG    R1,R2
        LOAD.REG    R11,R1

        LOAD.REG    R1,MEM9
        LOAD.REG    R2,R1
        MULT.REG    R1,R2
        LOAD.REG    R12,R1

        ADD.REG     R10,R11
        ADD.REG     R10,R12
        SQRT        R10,R11
        STOR.MEM    R11,MEMVN
;
;       CALCULATE V'
;
        LOAD.REG    R1,MEM4
        DIVIDE      R1,R11,R12
        STOR.MEM    R12,MEMVX

        LOAD.REG    R1,MEM5
```

```
        DIVIDE          R1,R11,R12
        STOR.MEM        R12,MEMVY

        LOAD.REG        R1,MEM6
        DIIVIDE          R1,R11,R12
        STOR.MEM        R12,MEMVZ
;
;       CALCULATE R = V' + 2N
;

        LOAD.REG        R1,D
        LOAD.REG        R2,E
        LOAD.REG        R3,F


        LOAD.REG        R4,VX
        LOAD.REG        R5,VY
        LOAD.REG        R6,VZ

        SHFT.LFT        R4,1
        SHFT.LFT        R5,1
        SHFT.LFT        R6,1

        ADD.REG         R1,R4
        ADD.REG         R2,R5
        ADD.REG         R3,R6

        STOR.MEM    R1,RX
        STOR.MEM    R2,RY
        STOR.MEM    R3,RZ
```

```
;
;       CALCULATE THE REFRACTION RAY
;


;
;       CALCULATE KF;    ASSUME N1 = 1.0 FOR AIR
;
;
;       CALCULATE KN** 2
;
        LOAD.REG    R1,N2
        LOAD.REG    R2,R1
        MULT.REG    R1,R2
        STOR.MEM    R1,KN2
;
;       CALCULATE V**2
;
        LOAD.REG    MEM4
        LOAD.REG    R3,MEM5
        LOAD.REG    R5,MEM6

        LOAD.REG    R2,R1
        LOAD.REG    R4,R3
        LOAD.REG    R6,R5

        MULT.REG    R1,R2
        MULT.REG    R3,R4
        MULT.REG    R5,R6

        ADD.REG     R1,R3
        ADD.REG     R1,R5
        STOR.MEM    R1,MEMV2
;
;       CALCULATE V' + N
;
        LOAD.REG    R1,VX
        LOAD.REG    R3,VY
        LOAD.REG    R5,VZ

        ADD.REG     R1,MEMDL
        ADD.REG     R3,MEMEL
        ADD.REG     R5,MEMFL

       LOAD.REG     R2,R1
       LOAD.REG     R4,R3
       LOAD.REG     R6,R5

       MULT.REG     R3,R4
```

```
        MULT.REG    R5,R6

        ADD.REG     R1,R3
        ADD.REG     R1,R5
        STOR.MEM    R1,MEMNXT

        LOAD.REG    R1,KN2
        ADD.REG     R1,MEMV2
        SUB.REG     R1,MEMNXT
        DIVIDE      R1,1,0,R2
        SQRT        R2,R3
        STOR.MEM    R3,KF
;
;       CALCULATE   N + V'
;
        LOAD.REG    R1,DL
        LOAD.REG    R3,EL
        LOAD.REG    R5,FL

        ADD.REG     R1,VX
        ADD.REG     R3,VY
        ADD.REG     R5,VZ
;
;       CALCULATE   KF(N+V')
;
        LOAD.REG    R2,KF
        MULT.REG    R1,R2
        MULT.REG    R3,R2
        MULT.REG    R5,R2
;
;       CALCULATE    P = KF(N+V') -N
;

        SUB.REG     R1,DL
        SUB.REG     R3,EL
        SUB.REG     R5,FL
```

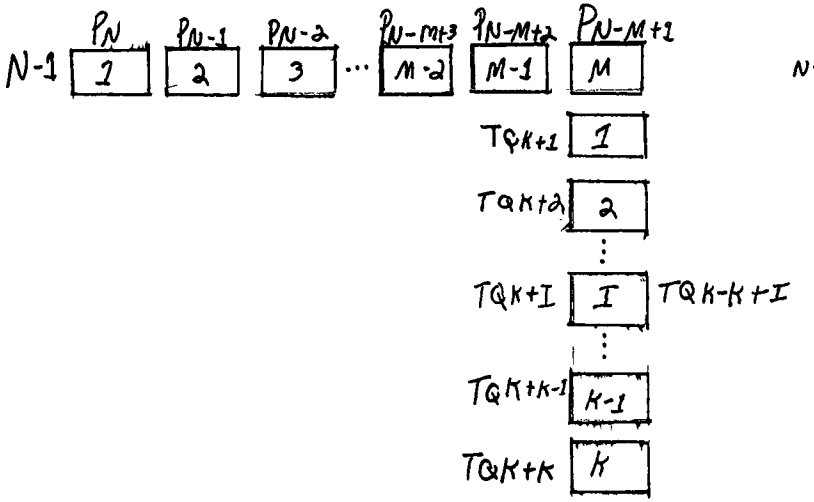
The following assembly-like instructions are used to calculate the normal at the point of intersection for a surface described by a polynomial of degree two.

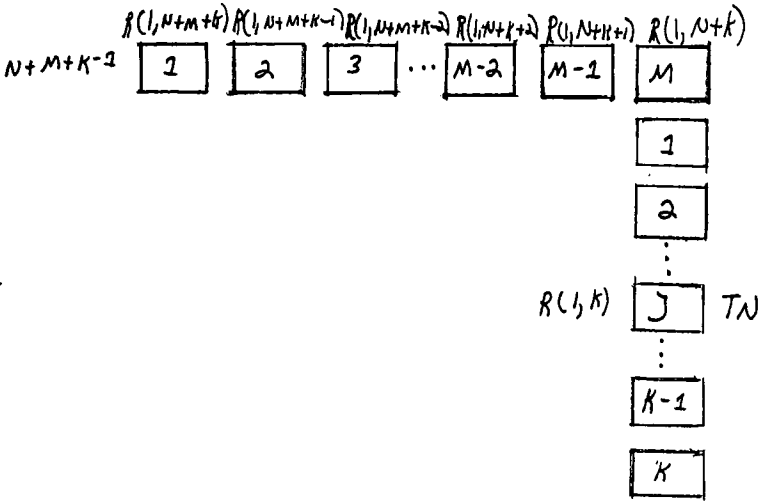
```
;
;       CALCULATE THE INTERSECTION POINT AS BEFORE
;
```

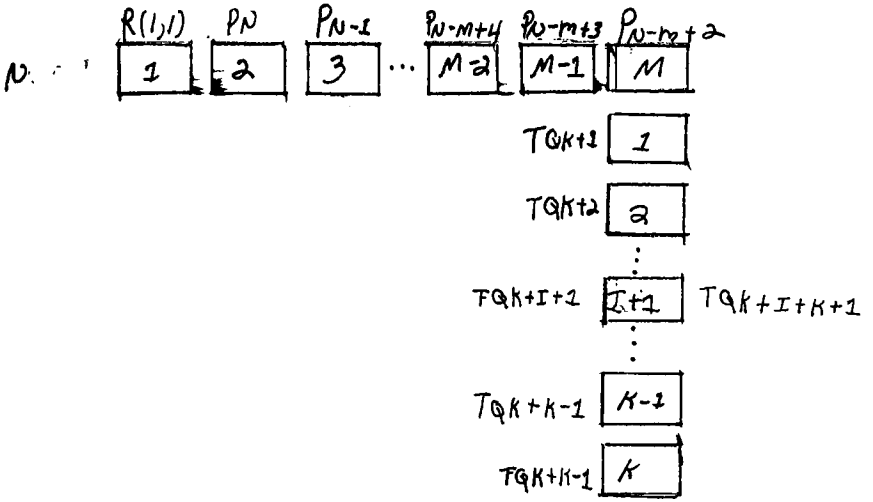
```
;
;     COMPUTE THE  VECTOR NORMAL TO THE
;     POINT OF INTERSECTION
;

      LOAD.REG    R1,A    FROM MEMORY
      SHFT.LFT    R1,1    2A
      LOAD.REG    R2,X    X COORDINATE OF INTERSECTION POINT
      MULT.REG    R1,R2   2AX
      STOR.MEM    R1,MEM4

      LOAD.REG    R1,D
      LOAD.REG    R2,Y
      MULT.REG    R1,R2   DY
      STOR.REG    R1,MEM5

      LOAD.REG    R1,F
      LOAD.REG    R2,Z
      MULT.REG    R1,R2   FZ
      ADD.REG     R1,G    G + FZ
      ADD.REG     R1,MEM5  G + FZ + DY
      ADD.REG     R1,MEM4  G + FZ + DY + 2AX
      STOR.REG    R1,MEMX

      LOAD.REG    R1,B
      SHFT.LFT    R1,1    2B
      LOAD.REG    R2,Y    Y COORDINATE  OF INTERSECTION POINT
      MULT.REG    R1,R2   2BY
      STOR.MEM    R1,MEM6
      LOAD.REG    R1,D
      LOAD.REG    R2,X
      MULT.REG    R1,R2   DX
      STOR.REG    R1,MEM7

      LOAD.REG    R1,E
      LOAD.REG    R2,Z
      MULT.REG    R1,R2   EZ
      ADD.REG     R1,H    H + EZ
      ADD.REG     R1,MEM7 H + EZ + DX
      ADD.REG     R1,MEM6 H + EZ + DX + 2BY
      STOR.REG    R1,MEMY

      LOAD.REG    R1,C
      SHFT.LFT    R1,1   2C
      LOAD.REG    R2,Z   Z COORDINATE OF INTERSECTION POINT
      MULT.REG    R1,R2   2CZ
      STOR.MEM    R1,MEM8
      LOAD.REG    R1,E
      LOAD.REG    R2,Y
      MULT.REG    R1,R2   EY
```

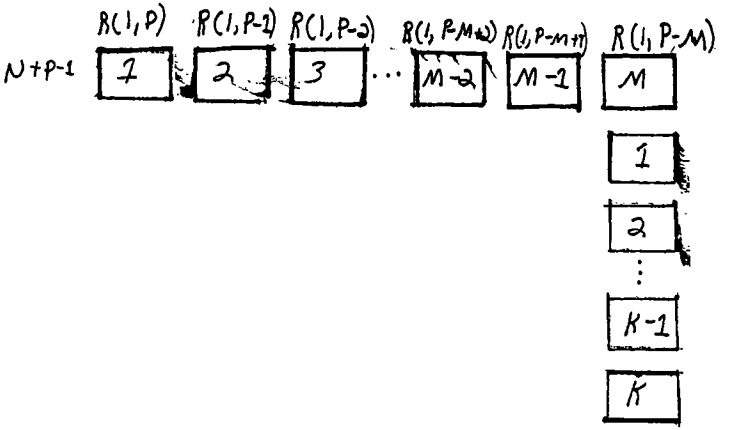
```
        STOR.REG    R1,MEM9

        LOAD.REG    R1,F
        LOAD.REG    R2,X
        MULT.REG    R1,R2     FX
        ADD.REG     R1,I      I + FX
        ADD.REG     R1,MEM9   I + FX + EY
        ADD.REG     R1,MEM8   I + FX + EY + 2CI
        STOR.REG    R1,MEMZ
;
;       CALCULATE THE NORMALIZED NORMAL VECTOR
;
        LOAD.REG    R1,MEMX
        LOAD.REG    R2,R1

        LOAD.REG    R3,MEMY
        LOAD.REG    R4,R3

        LOAD.REG    R5,MEMZ
        LOAD.REG    R6,R5

        MULT.REG    R1,R2
        MULT.REG    R3,R4
        MULT.REG    R5,R6

        ADD.REG     R1,R3
        ADD.REG     R1,R5
        SQRT        R1,R2   R2 CONTAINS THE LENGTH OF THE VECTOR

        LOAD.REG    R3,MEMX
        LOAD.REG    R4,MEMY
        LOAD.REG    R5,MEMZ

        DIVIDE      R3,R2
        DIVIDE      R4,R2
        DIVIDE      R5,R2

        STOR.REG    R3,MEMXN
        STOR.REG    R4,MEMYN
        STOR.REG    R5,MEMZN
;
;       CALCULATE THE REFELCTION AND REFRACTION
;       RAYS AS BEFORE
;
```

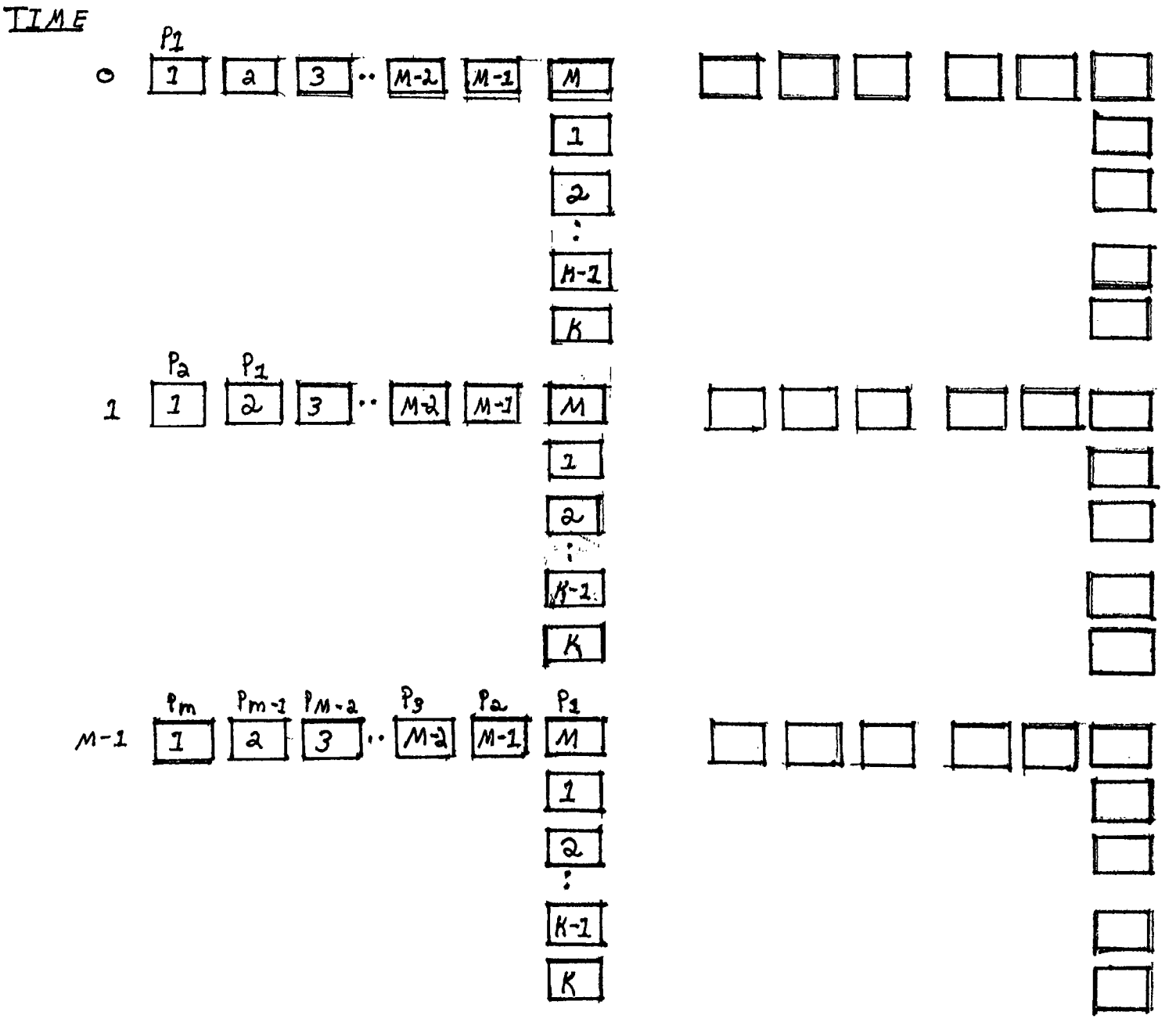
APPENDIX E

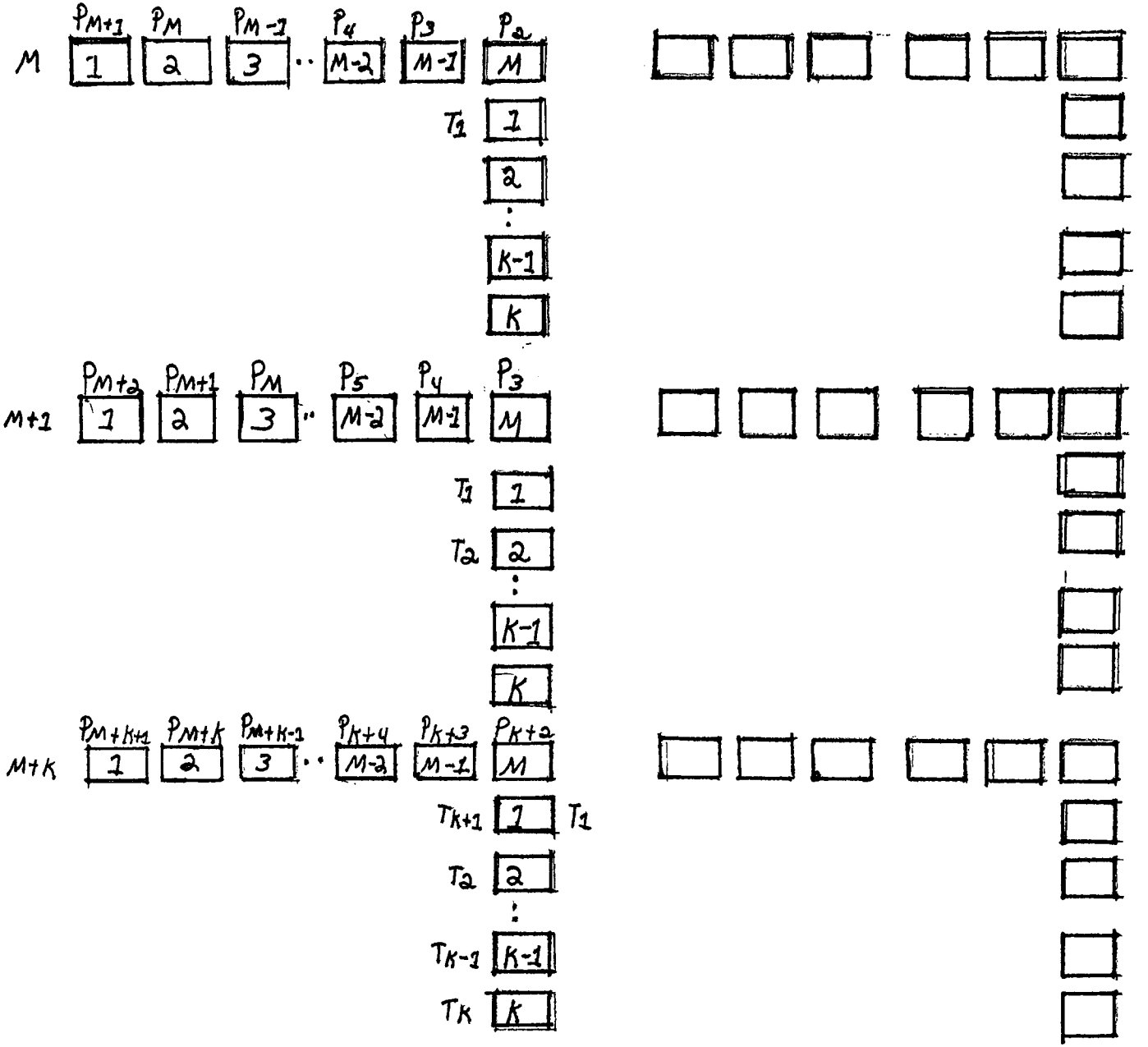

N-1
PN
PN-1
PN-2
PN-M+3
PN-M+2
PN-M+1
1
2
3
M-2
M-1
M
TQK+1
TQK+2
TQK+I
I
TQK-K+I
TQK+K-1
K-1
TQK+K
K

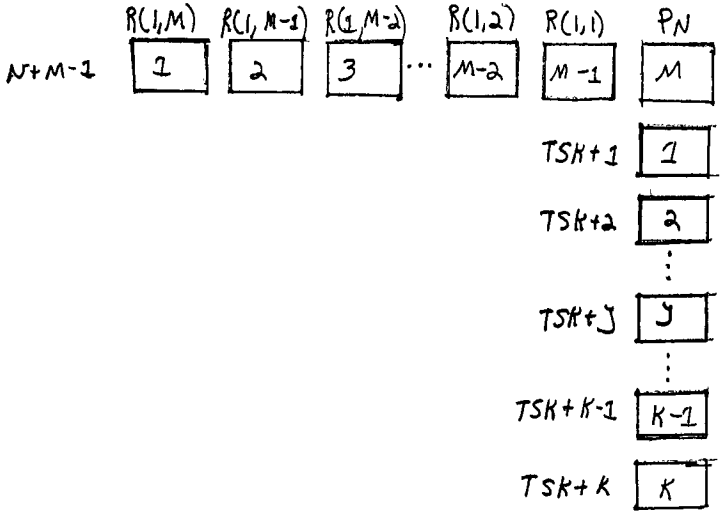

N+M+K-1
R(1,N+M+K)
R(1,N+M+K-1)
R(1,N+M+K-2)
R(1,N+K+2)
R(1,N+K+1)
R(1,N+K)
1
2
3
M-2
M-1
M
R(1,K)
J
TN
K-1
K

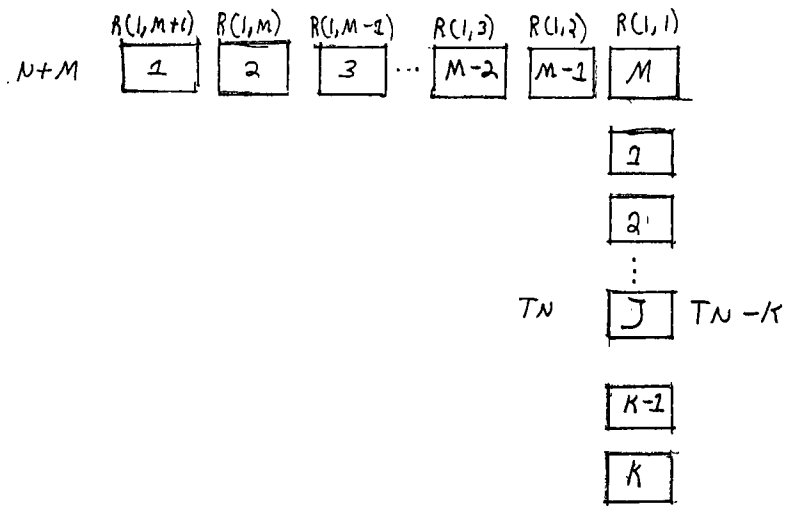

N
R(1,1)
PN
PN-1
PN-M+4
PN-M+3
PN-M+2
1
2
3
M-2
M-1
M
TQK+1
TQK+2
FQK+I+1
I+1
TQK+I+K+1
TQK+K-1
K-1
FQK+K-1
K


N+P-1
R(1,P)
R(1,P-1)
R(1,P-2)
R(1,P-M+2)
R(1,P-M+1)
R(1,P-M)
1
2
3
M-2
M-1
M
K-1
K


TIME
P1
0
1
2
3
M-2
M-1
M
1
2
K-1
K
P2
P1
1
1
2
3
M-2
M-1
M
1
2
K-1
K
Pm
Pm-1
PM-2
P3
P2
P1
M-1
1
2
3
M-2
M-1
M
1
2
K-1
K


M
P_{M+1} P_M P_{M-1} P_4 P_3 P_2
1 2 3 .. M-2 M-1 M
T_1 1
2
K-1
K
M+1
P_{M+2} P_{M+1} P_M P_5 P_4 P_3
1 2 3 .. M-2 M-1 M
T_1 1
T_2 2
K-1
K
M+K
P_{M+K+1} P_{M+K} P_{M+K-1} P_{K+4} P_{K+3} P_{K+2}
1 2 3 .. M-2 M-1 M
T_{K+1} 1 T_1
T_2 2
T_{K-1} K-1
T_K K


R(1,M)
R(1,M-1)
R(1,M-2)
R(1,2)
R(1,1)
PN
N+M-1
1
2
3
M-2
M-1
M
TSK+1
1
TSK+2
2
TSK+J
J
TSK+K-1
K-1
TSK+K
K


R(1,M+1)
R(1,M)
R(1,M-1)
R(1,3)
R(1,2)
R(1,1)
N+M
1
2
3
M-2
M-1
M
1
2
TN
J
TN-K
K-1
K