FAST RASTER GRAPHICS USING PARALLEL PROCESSING

by

Thomas Strothotte

B.Sc., Simon Fraser University, 1980

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the Department

of

Computing Science

SIMON FRASER UNIVERSITY

June, 1981

APPROVAL

Name: Thomas Strothotte

Degree: Master of Science

Title of Thesis: Fast Raster Graphics using Parallel Processing

Examining Committee:

Chairperson: Thomas K. Poiker

Brian V. Funt
Senior Supervisor

Pavol Hell

Richard F. Hobson

Patricia L. Brantingham
External Examiner
Associate Professor
Department of Criminology
Simon Fraser University

Date Approved: _____

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Fast Raster Graphics Using Parallel Processing

Author: _____

(signature)

Thomas Strothotte

(name)

30. June 1981

(date)

Abstract

A theoretical framework for a method of raster graphics image formation using parallel processing is presented. Parallel algorithms have been developed to quickly rotate, scale, translate and display three-dimensional objects. The hardware consists of a large number of individual processors with inter-processor communication restricted to immediate neighbours, organized as if they were uniformly spread about the surface of a sphere. The objects for display are modelled as if they were inside this sphere of processors.

Each processor stores the following information:

1) the intersection points of the object's surface with a radial line extended from the sphere's center, and
2) memory representing part of the screen.

The algorithm involves an organized message-passing scheme to accomplish rotations of the object description and the subsequent display of the image. Estimates based on the results of a simulation of the system using several hundred processors indicate that the perfomance improves in a near-linear fashion with the number of processors.

Acknowledgements

I wish to thank my senior supervisor Brian Funt for his invaluable guidance of my research, and Lolita Wilson for proofreading the manuscript. I wish to express my gratitude to Doreen Godwin, Elma Krbavac and my office-mate Shane Caplin for their moral support throughout my time at Simon Fraser.

Table of Contents

List of Figures

1. Introduction

1.1 Background

Computer produced pictures today provide a direct and useful method of communication between man and computer. The ability to produce realistic images is of immense value in research, education and industry. Computer simulators are used to drive displays for real-time environments for the training of pilots of aircraft, space-craft and ocean-going vessels. Real-life situations are simulated, but at high cost.

With the advent of the Very Large Scale Integrated (VLSI) circuit technology, the benefits of real-time image processing have come within reach of less affluent customers. Interest has already been shown by biologists who wish to supplement their cross-sectional information about specimens with three-dimensional views. Other diverse areas of application include simulation of human movement for dance studies (Calvert et. al(1980)), and advertising in industry (Newman & Sproull (1979)).

1.2 Motivation for Further Research

Most objects are inherently three-dimensional, but computer display screens are only two-dimensional. In real life, a person can walk around an object to see its total exterior. A reasonable substitute for this "walking around" is to smoothly rotate the object on the screen.

The problem is that smooth rotation of an object on a screen requires an enormous number of calculations in a short space of time.

For smooth rotation without flickering, a new image must refresh the screen every 1/30 of a second. This does not leave much time to compute the intensity of the 10,000 to 250,000 pixels found on raster graphics devices. With present uni-processor technology, real-time raster graphics image processing is not possible. In fact, the phrase "digital image rotation" has been used to refer to the rotation of an object in the plane of the screen, as opposed to rotation in three dimensions.*

The prime concern of this thesis is to reduce the time required to rotate and display a three-dimensional object. The problem will be attacked on two fronts. First, the method will involve a high degree of parallelism. The appropriate calculations will be performed simultaneously by many processors, thereby reducing the overall amount of time required for the computation.

The second line of attack will aim to reduce the complexity of the calculations required to rotate an object. By working in spherical coordinates, trigonometric information will be pre-computed and will remain constant throughout the rotation and display of the object.

In this thesis, a design for a parallel graphics system is developed. The methods used address the concerns expressed above. Programs will be described which have been written to simulate the graphics system and test its algorithms.

Following this introduction, the main body of the thesis is organized as follows. First, past methods of object representation will be examined. Then parallel processing as a source of improved

---

*Advertisment for COMTAL Vision ONE/20 in Computer 13, 7 (June, 1980), p. 63.

performance in graphics systems will be considered. Finally, the design of a geodesic structure for processing raster graphics images is presented with a detailed discussion of the algorithms and some results of a simulation.

2. Review of Previous Methods of Object Description

Previous methods of object description can be categorized according to whether they employ polygons, regular surface elements, or parametric surface patches. The most thoroughly studied type of representation is the polygon. Because of the shortcomings of the polygon method of modelling smooth, curved surfaces, several parametric surface representations have been developed. A variation is the method of storing many simple elements, which lends itself very well to simple hidden surface and shading algorithms; however, for arbitrary objects it can use up large regions of memory.

The algorithms for each of these three classes of object descriptions will be discussed in separate sections.

2.1 Polygon Based Object Descriptions

The most popular approach to object modelling has been to approximate surfaces with collections of polygons (Newman & Sproull (1979)). This works well for familiar objects such as cubes, parallelepipeds, wedges and polygonal prisms. By increasing the number of faces, a polyhedron can be constructed that will approximate any solid object. This completeness property makes the polyhedron attractive as a primitive representation. The drawback of the polyhedron approximation is that the processing and display times can become intolerable.

One of the useful properties of the polyhedral representation is that it is conceptually easy to work with. An object can be modelled by a set of faces where each face is a planar polygon defined by the

(x,y,z) coordinates of the endpoints of its edges. Rotations are performed by multiplying each endpoint $\underline{p}$ by a rotation matrix R to produce the rotated point $\underline{p}'$, viz.,

$$\underline{p}' = R\ \underline{p} \tag{1}$$

For example, for a rotation of θ about the x-axis, we would have

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{2}$$

Many of the computations on the polyhedra can be done in parallel. Transformations such as rotation are performed on each point, and the calculations on any point are independent of the calculations on any other point. The same is true for calculating the screen coordinates and pixel intensities. Display can be done by a multiple processor z-buffer scheme as demonstrated by Parke (1980).

Despite the fact that the operations can be done in parallel, this scheme does not perform well due to the long processing time for each of the polygons. In addition to several multiplications and additions needed to project a polygon onto the screen, calculating its intensity involves numerous trigonometric operations. Furthermore, performance gains of current multiple-processor z-buffer schemes at attractive only with small numbers of processors.

2.1.1 Hierarchical Organization of Polygon Data

In order to speed up clipping and visibility operations, Clarke (1976) proposed organizing the polygon data by the use of a hierarchical representation. It consists of trees whose branches represent bounding volumes and whose terminal nodes represent

primitive object elements, usually polygons.

A sample scene is shown in Figure 1a. Objects close to one another are grouped together. They are placed into "bounding volumes", and organized as in the tree structure illustrated in Figure 1b.

Rubin & Whitted (1980) suggest that the bounding volumes be parallelepipeds, oriented to minimize their size. With this representation any surface can be rendered, since in the limit the bounding volumes make up a point representation of the object. The advantage is that the visibility calculation consists only of a search through the data structure to delete the correspondence between terminal level bounding volumes and the current pixel. For ray-tracing algorithms, this means that the point of intersection of each ray need only be calculated for a small number of bounding volumes, not for every object.

While ray-tracing can be performed in parallel, there nonetheless remains an enormous number of calculations to be done. A ray from each pixel must be traced through the hierarchical representation.

The bounding volume polygon organization is particularily effective when a large amount of detail is being stored. When the detail contained in the bounding volumes becomes finer than the resolution of the display device, no more detailed information need be processed. "Zooming in" on the data structure is accomplished by the simple operation of traversing down the tree. Clarke(1976) and Rubin & Whitted (1980) note that this means that the entire data structure could be stored on disk, and the relevant portions

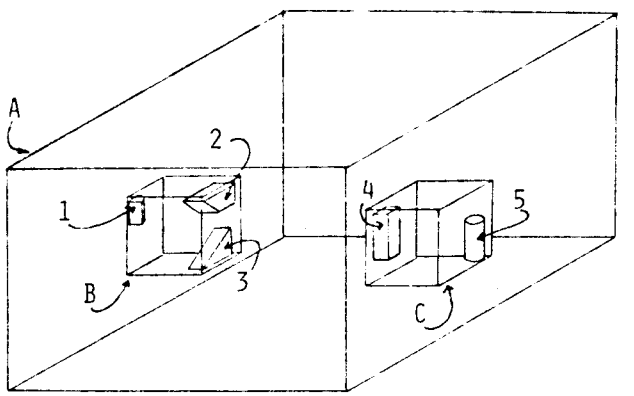


Figure 1a

Method of hierarchial organization of a scene. The entire scene is enclosed in the polygon labelled A. Two sets of subordinate objects are then grouped together into volumes B and C. The grouping simplifies ray-tracing algorithms, since initial intersections must be calculated only on the bounding volumes, not on the individual objects.

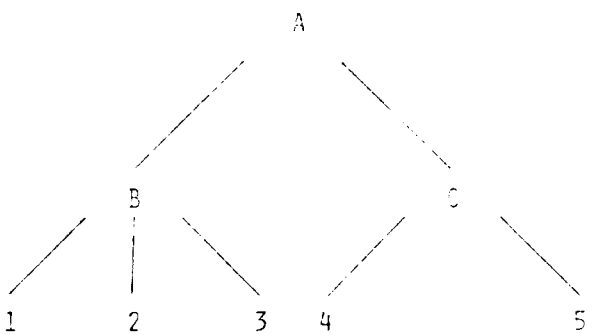


Figure 1b

Tree Organization for scene above. Ray-tracing proceeds by stepping down the tree, looking for intersections. Once the size of a bounding volume becomes less than the resolution of the display device, no lower levels need be considered.

retrieved as needed.

This method of representation is useful for scenes in which several different orders of magnitude of detail are desired. Moreover, the scheme is most useful for scenes in which the objects are clustered, particularily when these clusters are relatively far apart. The ray-tracing is then simple because most of the rays miss most of the bounding volumes.

For objects which do not fall into these categories, performance of the hierarchical representation scheme degenerates very quickly. Bounding volumes become extraneous information which must be stored and processed by the ray-tracing algorithms.

2.1.2 Spherical Organization of Polygons

If polygons are to be used as the form of representation, a method of selecting their positions must be defined. Brown (1979) suggested choosing polygon vertices from piecewise planar functions of a sphere called "well-tesselated surfaces". A geodesic dome is spread around the object, and each of the triangular faces of the dome is allowed to contract radially inward until the vertices rest on the surface of the object. The plane of each triangle is determined by the points at which the vertices touch the object. The collection of triangles then describes the object.

Brown discusses a very simple hidden-surface algorithm for his modelling scheme. The faces are sorted in decreasing order of angle at the origin between any face point and the viewing direction. Brown cites the example of a globe: if the viewport is above the north pole, the sort is on the minimum latitude. Faces are then

projected onto the screen and displayed in the sorted order. In this way, portions of the screen corresponding to hidden surfaces are overwritten. This display scheme is an inherently sequential process which cannot be done in parallel. The method as presented by Brown has no advantages over the unstructured polygon modelling. Indeed, it results in a great deal of unnecessary calculation in cases where parts of the object could be modelled with considerably fewer polygons, and there is no return on this investment of extra work.

2.2 Parametric Surface Representations

Although the use of polygons is a simple and workable method of modelling objects with flat surfaces, problems arise when polygons are used to represent smooth, curved surfaces. To handle these problems, more refined models of parametric surface patches were proposed, by Bezier (1974) and DeBoor (1972), who developed numerous methods to parameterize surfaces. The essential concepts are illustrated in Figure 2. A function of f(x,y) is defined, which gives the z-coordinate of the surface. The function is evaluated at regular intervals when a display is required. The patches can be defined so that they require no more, and often less, data than corresponding polygon-based surface descriptions.

Surfaces described in a parameteric form can be generated in parallel, since calculations of the z-values are independent of each other. However, Clarke(1976) notes that complications arise because the mathematics is no longer linear; it can be very time consuming to calculate the intersections of surfaces and to calculate the points above a clipping plane.

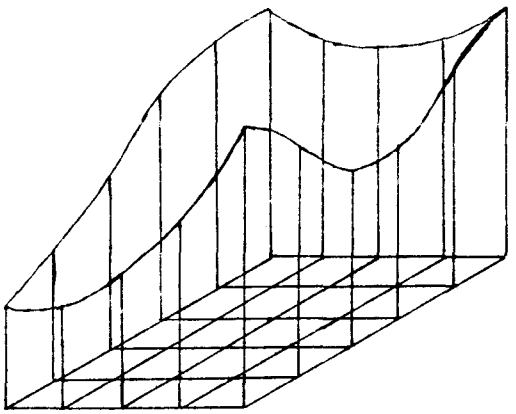


Figure 2

Parametric specification of curved surfaces. At each of the vertices on the grid, a height value is calculated. For clarity, only the values at the edges are shown.

2.3 Simple Surface Elements

Computer tomography stimulated an alternative approach to representing three-dimensional objects (Herman (1979)). The surface is modelled by a large number of identical surface elements which are both small and simple. The method is in sharp contrast to methods of modelling objects by a relatively few complex, albeit easily "parameterizable", elements.

Consider the two-dimensional situation illustrated in Figure 3. The plane of the paper is divided into a series of squares, and a geometric figure, in this case a circle, is represented by shading certain of the squares.

We can extend this notion to describe the objects in the real

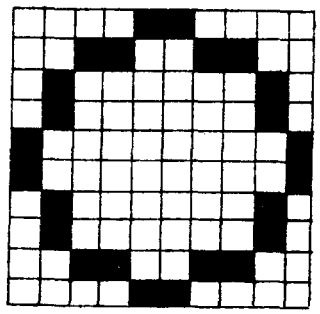


Figure 3

Method of modelling a geometric figure via a grid of squares. The set of shaded squares form a circle.

world. The region of space which is to be modelled is divided into a large number of cubes, and an object surface, called a cuberille, consists of a subset of the collection of those cubes. In order to model a general object accurately, a large number of voxels are required. Herman states that the surface of an human organ can typically be modelled by 10,000 to 25,000 faces. Since all the voxels are in the same orientation, the shading and screen projection calculation for each facet is quite simple. The visible surfaces of the voxels have only one of three possible orientations, simplifying intensity calculations. This has the result, however, that a surface may have a checkered appearance on the display.

The voxel representation lends itself well to parallel processing.

Transformations such as rotation can be performed independently on each voxel, and the z-buffering scheme is easily implemented to eliminate hidden surfaces. However, the calculations for each individual rotation involves a great deal of computation, as with the polygonal representation.

3. Parallel Processing

Parallel processing offers attractive computational gains. Indeed, Crow (1980) indicates that parallel processing holds the greatest promise for improvements in image processing. In this section, parallel processing in the context of graphics will be reviewed. First, the potential gains which may be realized with parallel processing will be examined. Following this will be consideration of multi-processors and network computers. Finally, techniques for constructing parallel solutions will be discussed.

3.1 Performance Improvement

The most obvious potential advantage to a multi-processor or network computer is an improvement in the machine's performance, both in terms of terminal response time and the time required for complex computations.

Consider a set of operations which takes T time units to complete on a uni-processor. It might be reasonable to expect that n processors could complete the task in T/n time units. In practice such an improvement is rarely realized. The problems encountered are similar to our every-day experiences. If we have a large staff working on a task, a certain amount of staff time is consumed in the management, delegation, supervision and checking of work. Additional time is used in inter-staff communication about related sub-tasks which are performed by different persons. Moreover, a given task may involve a great deal of sequential work. For example, in the newspaper industry, the paper must first be written

by reporters, then printed, and finally distributed. It is crucial that the operations be done in that order.

These same problems plague multi-processor computers. Some processors can be added to reduce the amount of overall time taken by a task. At a certain point, however, performance no longer improves, and in some cases actually declines, with more processors. A graph showing a typical speedup curve is shown in Figure 4. If the amount of time take by n processors to complete a task is T(n), the speedup S(n) is defined as T(1)/T(n). The peak of the actual response curve is often found at less than 10 processors (Chu et. al (1980)).

3.2 Multi-processors

Multi-processor architectures have two important distinguishing features. First, they include multiple, autonomous processors, and second, all processors share most, and often all, of primary memory. A great many multi-processor architectures have been proposed in the literature, though only a few have actually been built. In the few which have reached the building stage, there exist some interesting results.

Jones & Schwartz (1980) give a status report of experience using multi-processrs, citing three major systems. Programming efforts with the C.mmp/HYDRA, the Cm*/StarOS, and the PLURIBUS have indicated that a near linear speedup is attainable for some problems. In fact, one searching program actually showed a better-than-linear speedup in performance. However, all these results were attained with modest numbers of processors (less than 50). It

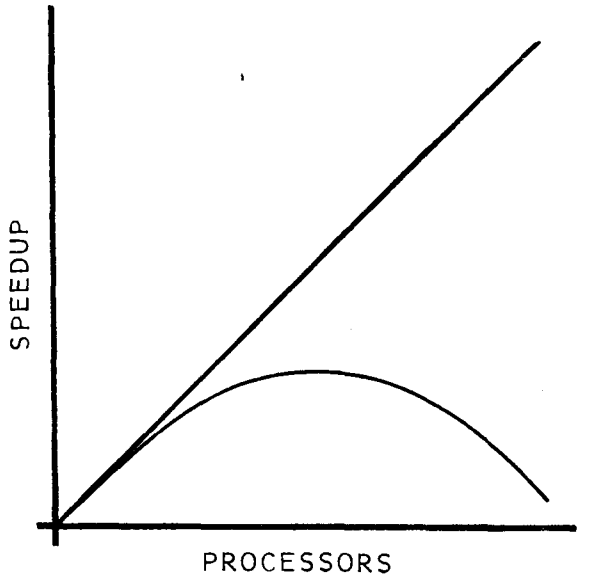


Figure 4

Graph showing improvement as a result of increasing the number of processors. The straight line is the ideal linear speedup, while the curve is a typical result.

is not clear whether these trends can be extrapolated to a significantly larger number of processors.

Jones & Schwartz go on to make several important statements about their experience. They assert that parallel programs are not qualitatively different from sequential programs, nor are they more difficult to write. The authors further predict that no major new programing language will be required to write software for multi-processors, although they do not provide a complete justification of this statement. Indeed, Feldman (1979) has designed such a language, though its usefulness remains in question.

3.3 Network Computers

The development of very large scale integrated (VLSI) circuit technology has made it possible to have a "computer on a chip", consisting of a complete micro-processor. Serving as a node in a network, each chip can communicate, via messages, with a small number of other nodes. Wittie (1980) surveys numerous connection methods, or topologies, for linking networks.

An important factor in evaluating a network is the maximum number of nodes a message must visit to travel from a node A to any other node B in the network. Related to this is the message traffic density, which measures the system bottlenecks in the network. Another consideration is the ease with which a network can be laid out on a two-dimensional plane for a VLSI chip implementation. Also for fault-tolerance, the network should be able to recover from single-point failures.

3.4 Techniques for Constructing Parallel Solutions

Several broad techniques for constructing parallel solutions to problems have been proposed. The most important step in finding such a solution is to identify the smallest inherently independent subpart of the problem. This unit is referred to as a grain, and its size is the granularity of the problem. Once this has been identified, the problem may be solved independently on each of the grains, or on groups of grains.

For problems in which computation can be performed independently on subsets of the data, the input data can be partitioned into groups, with each group assigned to one processor.

The code to be executed can be replicated on each processor. Parke (1980) used this idea in his method of parallel processing in a z-buffer system. Using an interlace pattern, pixels of the screen are distributed among "image-processors". Polygons are input to "splitter-processors", which are organized in a tree structure. The splitters decompose the polygons into pixels and broadcast the intensities and z-buffer information to the image processors. Parke's system suffers from the saturation effect discussed earlier: the peak in his speedup curve is found at between 16 and 64 processors.

If the problem which must be solved consists of several inherently sequential segments, these segments can be handled by a pipeline. Output from one phase is treated as input to the next phase. Each phase can in itself then be processed in parallel. Clarke(1980) made use of this method in designing his Geometry Engine, a VLSI system for transforming polygon-data. The system scales, translates and clips polygon data, preparing it for a z-buffer system such as that of Parke. Parallelism is implemented at the level of the arithmetic processing within the processor, where polygon data is processed in a pipeline. Clarke's system is capable of transforming about 900 polygons in the 1/30 second required for real-time processing.

4. Design of a Geodesic Structure

In the last section, we reviewed several broad categories of object descriptions, noting several attempts at information structuring. In this section, we will discuss a scheme by which a rotation of the object description can be performed with relatively few calculations. First proposed by Funt (1981), the method amounts to structuring polygon data in a spherically symmetric manner.

After an overview of the method, a formal description is given. The distribution of the polygons will be discussed, followed by an analysis of the interaction between the hardware and software of the system.

4.1 Overview

One of the design criteria was to have the rotation calculation involve only simple arithmetic. A construction which satisfies this criterion in two dimensions is illustrated in Figure 5. Suppose that we have eight people standing in a circle around an object of interest. Each person records in a log book what he sees when he looks radially inward, and then turns his back to the object. The group collectively now has a complete description of the object. No one person can see every part of the object, but every part is seen by someone.

Suppose now that we wanted to rotate the object description by 45 degrees. One way to do this, is to have each person pass his log book to the person to the right. Without having moved the object, the description is rotated. A rotation of a further 45 degrees

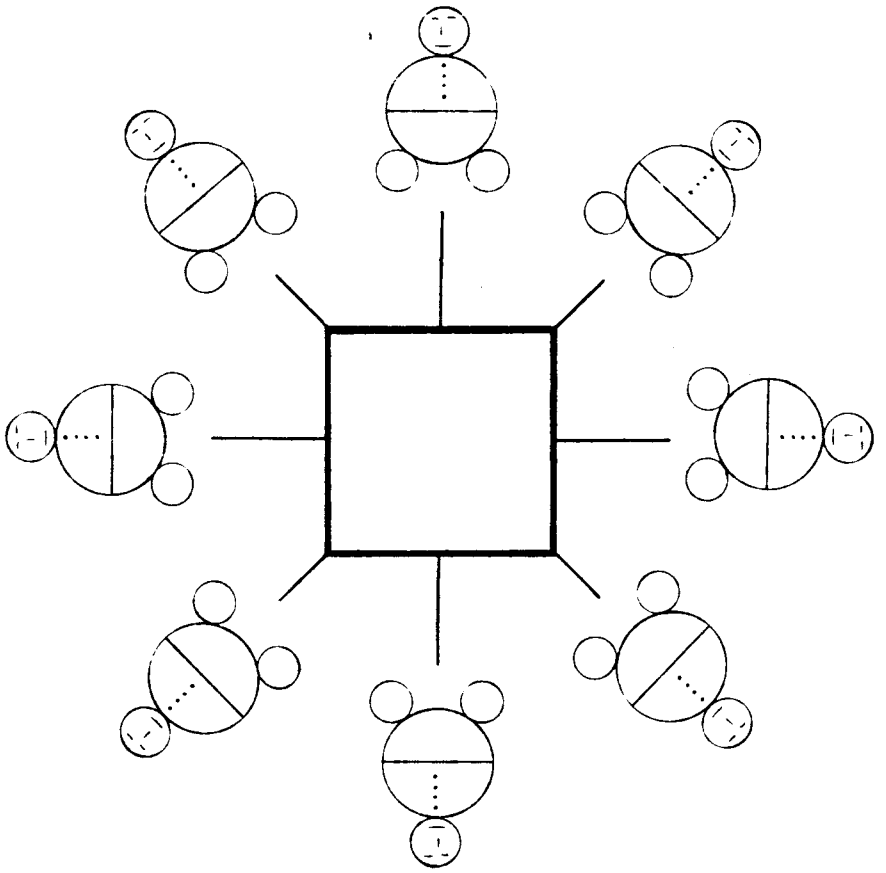


Figure 5

Method of modelling objects in two dimensions. A group of people, standing around an object, observe what they see by looking radially inward. A rotation is accomplished by passing the description to their neighbors.

would be accomplished by passing the logbook to the right once more, and so on.

This scheme can be extended to three dimensions. Imagine a hollow sphere with a three dimensional object placed inside it. Equally spaced around the sphere are people who look radially inward, and record what they see between where they are, and the center of the sphere. A rotation is accomplished by each of the observers passing his view to his neighbour along the axis of

rotation, as illustrated in Figure 6.

It is interesting to note that this scheme of managing the information is the inverse of the vision system called a "compound eye" which is found in some animals, such as insects. Some 1,000 to 40,000 hexagonal "facets" are spread about a hemishere on either side of the animal's head. Each facet is able to receive only a pencil of light falling perpendicular to its face. The animal can differentiate motion and velocity by determining the speed at which an object crosses the field of view of successive facets.

The important feature of this scheme of data organization is that there are no sine or cosine calculations necessary to rotate the description of the object. This simplicity comes as a result of working in polar coordinates. In the next section, we will see how these concepts can be exploited in a computer model.

4.2 Computer Model

The informal notions introduced in the last section exhibit properties which make them very attractive for actual implementation. The key point is that the actions of each person can be performed by an individual processor in a multiple-processor environment. Communication between processors is limited to a small number of neighbours, making it realistic with present-day technology. Further, the operations performed by each of the processors are very simple. To rotate, for example, requires only a small number of multiplications and additions to determine where the information must be sent and then passing it to the correct neighbour.

Each of the "people" on the "sphere" is modelled by a data

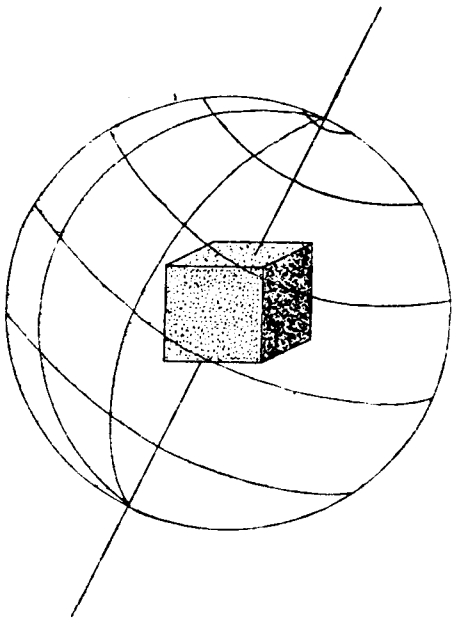


Figure 6

Rotation of an object in 3-space, analogous to the 2-space rotation of Figure 5. Information is passed to the neighboring node along the axis of rotation. Note that the speed of rotation is limited by the speed of the message-passing at the equator.

structure called a facet. Each facet knows its (ϕ,θ) coordinates, and the distance to the points at which a surface of the object crosses a line extended from the facet to the center of the sphere. The surface normal at that point is also recorded for later use in the shading calculation. In addition, the facet will have a list of the neighbouring facets to which it can send information.

4.3 Distributing Facets Evenly About a Sphere

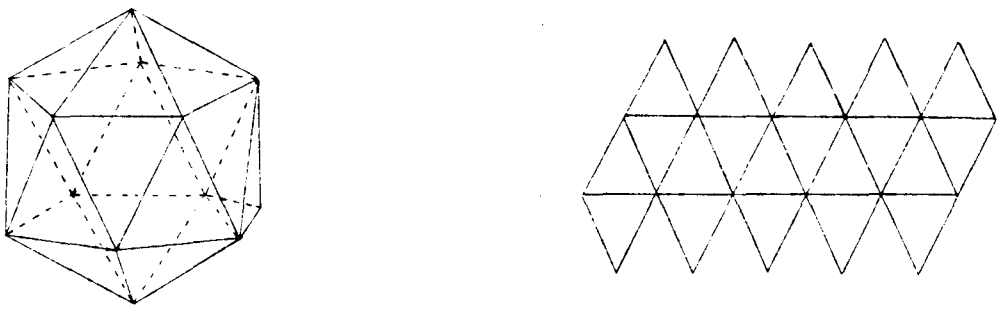
Ideally, we want the facets to cover small "patches" of the surface of the sphere. These patches should be as round as possible, so that the distance to the object surface, when measured at the centers of the patches, is representative of the patch as a

whole. Furthermore, to preserve symmetry in the data structure, we want all the facets to be identical.

Due to the geometric constraints of the real world, these ideal conditions cannot be met. First, the facets can clearly not be completely round, but must be polygons, otherwise there would be sizable "inter-facet" gaps, leaving parts of the surface unrepresented. Furthermore, it is well known that there is no way to completely cover the surface of a sphere with many identical polygons (Harris(1977)). A good approximation is the icosahedron.

A diagram of an icosahedron is shown in Figure 7a. The icosahedron has 12 vertices, 20 faces called Principle Polyhedral Triangles (PPT's) and 30 edges. A "flattened-out" version of the icosahedron is shown in Figure 7b.

We need more than the 20 facets which the PPTs provide. The PPTs are then subdivided as in Figure 8. Each edge is divided into n equal segments. Each point is connected with a line segment to a point on each of the other edges, so that all the line segments are parallel to one of the edges of the PPT. The result is a 3-way grid on the PPT consisting of equilateral triangles. Finally, each vertex is translated outward until it is at unit distance from the center of the sphere. Hexagonal facets are constructed by collapsing groups of six triangles. The triangles located around the vertices of the PPT's form pentagonal facets.



(a) (b)

Figure 7

An icosahedron (a) and the same object spread out in two dimensions (b). Note that each vertex has 5 edges. The vertices of the triangles at the top and bottom of (b) actualy correspond to the north and south poles, respectively.

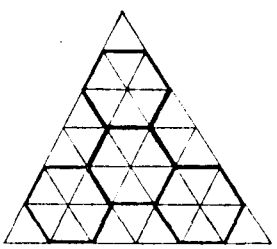


Figure 8

Method of subdividing triangles of icosahedron. Each principle polyhedral triangle is divided into smaller ones, which are then grouped to form hexagons.

4.4 Overview of Architecture

Now that a scheme for modelling the objects has been developed, attention will be shifted to an implementation of the model. While we will not intimately concern ourselves with the details of the hardware, we nonetheless must maintain a conceptual overview of the system so as to understand the environment in which the model and its programs will function.

An overview of the architecture to be used is illustrated in Figure 9. At the top level, a user inputs commands which will define the object. Later the user inputs other commands to manipulate and rotate it. These commands are read by the "data and command handler", which translates the high-level descriptions from the user into commands on the group of facets. The facet-commands are then passed on to the "processor supervisor" which issues instructions to all the individual processors. The processors themselves contain local memory for the facet information they currently hold, and they hold memory for pixel intensities which the video generator can read and display on the graphics device. Each processor can communicate with five or six other processors.

The significant feature of the system is that the processors' workload is composed of three distinct segments:

1. Object description and manipulation. Facets are stored by the processors and the facets are manipulated by the processors to reflect rotations requested by the user.
2. Message-passing. This operation occurs at two different times. One such time is during object rotations, since they involve passing the object description to a neighbouring processor. Since

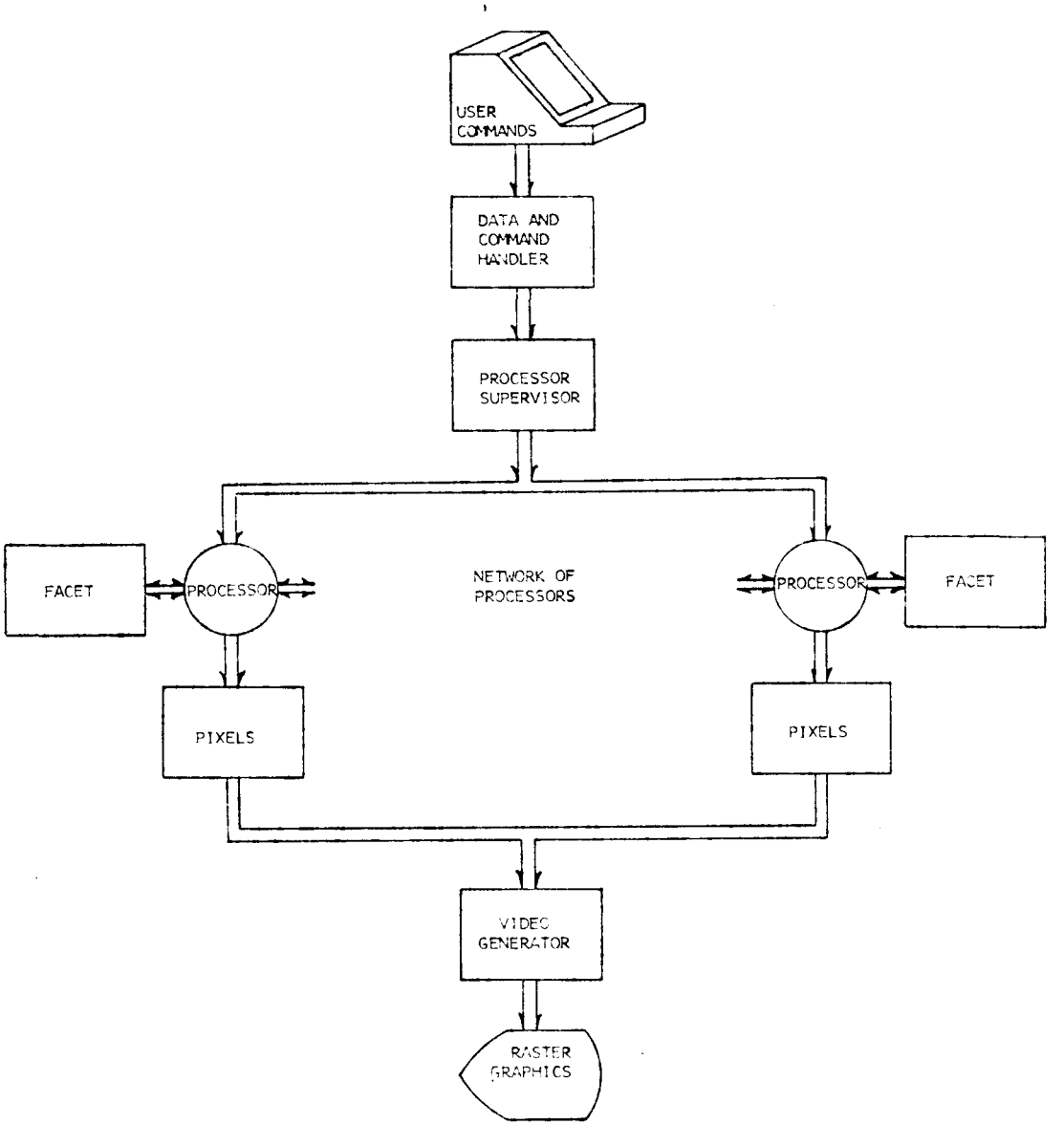


Figure 9

Architecture of overall system. Information is supplied by the user at the top level and then handled by the interconnected network of processors. Each processor has storage for some facets, and separate storage for some pixels. The video generator reads pixel values to display on the raster output device.

any one rotation step is only through a small angle, these messages only have to traverse a small number of processors. The second time message-passing is performed is when the facet information is converted to pixel intensities. At this time, the pixel information must be sent to the processor which actually holds the corresponding pixels.

3. Z-buffer. Every processor holds data corresponding to certain pixels, and must maintain a z-buffer for them.

4.5 Overview of Software

Although from the point of view of the hardware, the processors can be visualized as being in a planar arrangement, the graphics programs view them as being uniformly spread about the surface of a sphere. Each processor holds one facet. The following steps are involved in displaying an object on the screen. Each point held by a facet is converted to rectangular Cartesian coordinates. The first two components are treated as the (x,y) coordinates of that point on the screen, while the third component is treated as the z-distance. Neighbouring processors hold facets which denote adjacent portions of the same surfaces: the rectangular Cartesian coordinates of these adjacent facets are combined to form triangles which approximate the surface. Each triangle is then sent, via inter-processor messages, to the processor which holds the portion of the screen on which the triangles are to be displayed. A polygon-filling algorithm then fills in the triangles and the pixel data inserted into the z-buffer.

How well the system performs depends heavily on how the inter-processor communication is handled. Clearly, the more nodes a

message must traverse, the more time is required. In the next section, the merits of our inter-processor connection scheme will be examined.

4.6 Connecting Processors: A Problem in Graph Theory

In order to make the design reasonable, we must limit the number of connections between processors. Information must be moved around between processors, because what a piece of information means is determined by which processor is holding it and how it is related to the rest of the object description. In this section, we examine the factors which must be considered in deciding which processors to connect. The problem will be discussed in terms of graph theory.

4.6.1 Basic Definitions

Let us begin by defining some of the relevant terms from graph theory.

Graph — A Graph G(V,E) consists of a set of points called vertices, V, and a set of edges, E, connecting vertices.

Order — The order of a graph is the number of vertices in the graph.

Adjacent — Two edges are said to be adjacent if they have at least one endpoint in common.

Degree — For a vertex x, the degree d(x) is the number of edges with x as an endpoint. The degree of a graph is the maximum of all d(x).

Distance — The distance d(x,y) between two vertices x and y is the length of the shortest path between x and y, where "length" is defined as the number of arcs which must be traversed.

Associated number

The associated number e(x) of a vertex is defined as e(x) = maximum of all d(x,y).

A traveller at x can reach any other vertex with e(x) or less stops.

Center — The center of a graph is the vertex with the lowest associated number.

Radius — The radius is the associated number of the center.

Diameter — The diameter is the maximum associated number of the vertices in the graph.

4.6.2 Attributes of the Graph

There are several important attributes which the graph formed by the geodesic dome construction has.

1. The graph is planar with degree 6. The small degree makes a hardward implementation feasible. The planarity of the graph means that even a VLSI implementation of the system may be possible (Sequin(1981)). Our system, using several hundred processors, is too large to fit onto one wafer using present-day technology. However, the density of logic-gates per chip is steadily increasing, and eventually the wafer may be able to hold a system of the magnitude of ours.
2. The diameter of our graph is approximately equal to the radius.

This is easy to see: note that for every processor, there is a processor on exactly the opposite side of the dome. Since the density of processors per unit area is constant, the radius of the graph is approximately equal to its diameter.

This radius-diameter relationship is important, because if there were a large discrepancy between these quantities, bottlenecks in the system would result at the center, or centers, of the graph.

3. The radius of the graph is proportional to the square root of the number of processors. To see this, consider the hexagon in Figure 10. Messages are passed along the path labelled x. Using this distance, we will derive an expression relating the number of processors, n, to the radius of the graph, i.e., the distance between the poles.

As an approximation, let us assume that we have n identical hexagons spread over the surface of the sphere, instead of our 12 pentagons and n-12 hexagons. Using simple trigonometry, we can show that the area "a" of each hexagon is

$$a = 2 * (3^{**}.5) * x^{**}2. \qquad (3)$$

The sum of the areas of the n hexagons must be equal to the area of a unit sphere,

$$n * a = 4 * \pi. \qquad (4)$$

Substituting Equation 3 into Equation 4 yields an expression for x,

$$x = \mathrm{sqrt}((2*\pi)/(n*3^{**}.5)). \qquad (5)$$

The largest distance which a message ever has to travel is the pole-to-pole distance π. The number of nodes which must be traversed is therefore,

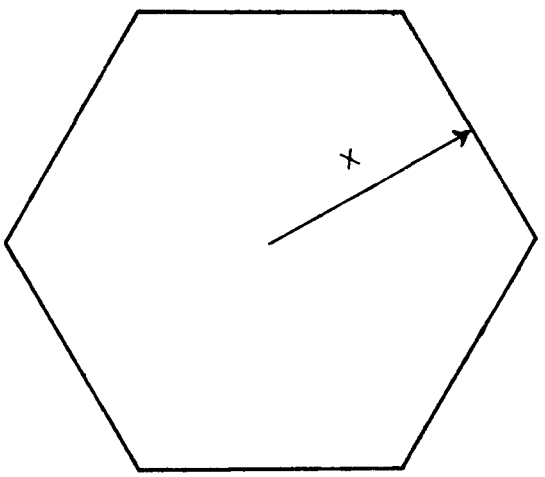


Figure 10

Hexagonal facet showing message path. Messages are communicated along the path labelled x.

$$r \simeq \pi / 2 \; x \tag{6}$$

or

$$r = \text{sqrt}(\pi*3**.5*n/8). \tag{7}$$

Numericially, this is

$$r \simeq .82 \; n**.5 \tag{8}$$

5. Realization of the Geodesic Structure

In this section, we describe the details of the geodesic structure which we proposed in the last chapter. Along with the algorithms, we will highlight the important features of a 4,000 line LISP simulation, which was written as an aid to the development of the algorithms. The simulation used 482 processors, with one facet per processor.

We will present a theoretical analysis of the performance of the system, along with some experimental results obtained from the simulation. Finally, we will show how the algorithms can be extended.

5.1 Methods

A flowchart giving an overview of the flow of control of the system is illustrated in Figure 11. The preliminary setup of the system consists primarily of determining which pixels will be held by which processor. Other system constants which can be pre-computed in this phase will be pointed out later. The object-initialization phase consists of inputting surfaces describing the object. The surfaces are sampled at the facet locations, and these samples are linked together to approximate the surface. The rotation parameters are also initialized at this time.

The operation phase involves the actual rotation and display of the object. We will show the principles by which the calculations are performed, and indicate which of the relevant parameters can be pre-computed. The display itself consists of four phases:

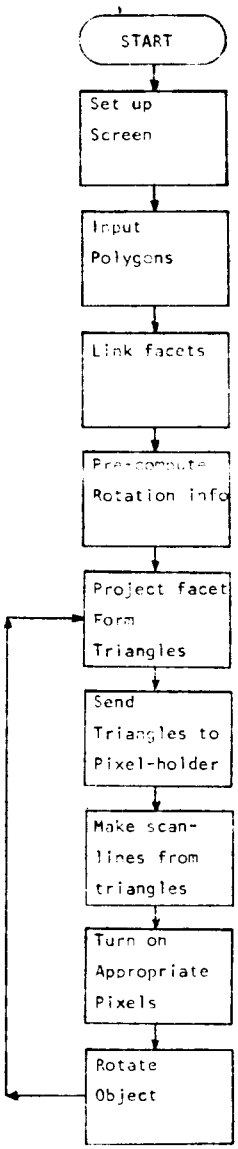


Figure 11

Flowchart showing overview of major system functions.

1. calculate the rectangular Cartesian coordinates of the facets and their intensities,
2. assemble sets of three facets to form triangular patches,
3. send the triangular patches to the processor governing the corresponding pixels, and
4. turn on the appropriate pixels.

The algorithms will be illustrated using a specific processor as an example. Each has a name, such as P31-123. The first number, 31, refers to the ϕ coordinate of the processor, rounded to the nearest integer, while the second number, 123, refers to the θ coordinate. The actual coordinate of P31-123 is

$$(1,\ 30.7278,\ 122.5887).$$

This processor can communicate with its six neighbours: P28-114, P41-123, P36-117, P20-120, P24-129 and P36-131.

5.1.1 Preliminary-Setup

At this time, the pixels on the screen are split up among the processors. Each processor is to handle an equal part of the screen, so that the work of turning the pixels on is distributed as equally as possible. Further, it is important that adjacent processors hold adjacent regions of the screen, since a triangular screen patch, consisting of facets at its vertices, can stretch over the region governed by more than one processor.

The two angular coordinates of the processors provide a convenient way of mapping processors onto the screen. The first angular component varies from 0 to 360, the second from 0 to 180. When we normalize these coordinates, each processor will have two

coordinates between 0 and 1, which then describe a unique position on the screen. Each processor holds the pixels which lie in a small region around its screen position.

The distribution of processors produced by this method is illustrated in Figure 12. Each polygon in the figure represents the part of the screen governed by one processor. The distribution is relatively even toward the middle of the screen, but thins out toward the top and bottom. This is a result of the singularities in the spherical coordinate system. For many applications, this is acceptable, since the center of the screen is more heavily used than other parts of the screen. For this reason, this scheme was used in the simulation.

For a general purpose raster-graphics device, however, this may not be acceptable. To rectify the situation, the (ϕ,θ) coordinate of the processor could be multiplied by a modulating function, which would have the effect of moving the end processors closer together, and pulling the central ones farther apart.

Each processor must "be aware of" which pixels it is holding. For each scan-line segment a processor holds on to, it maintains (see Figure 13):

1. a pointer to the processor holding the previous segment of this scan-line,
2. the x-coordinate of its starting position,
3. the y-coordinate of its starting position,
4. its length, and
5. a pointer to the processor holding the next segment of this scan-line.

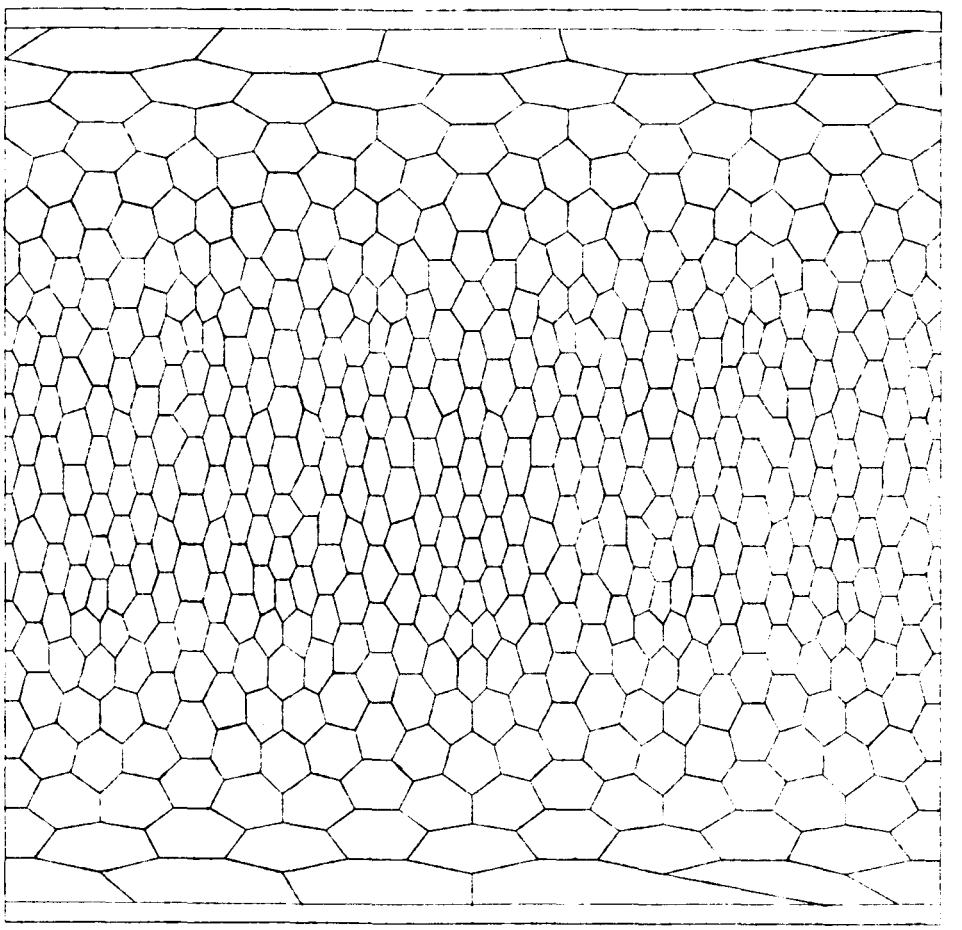


Figure 12

Distribution of Pixel-holders among screen. Each processor governs one of the polygons.

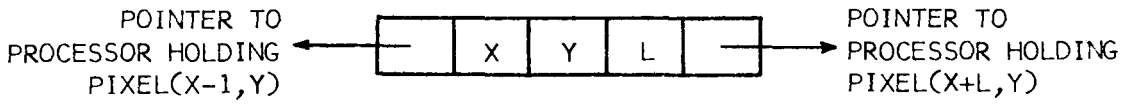


Figure 13

Sample scan-line for pixel-holder.

The first and last items in this list are required in instances where a triangle spreads over the pixels governed by several processors.

5.1.2 Object-Initialization

In this phase, the object of interest is input to the system. The user specifies a set of polygons approximating the object's surface. Each polygon is identified by a name. The vertices of the edges must lie inside a unit sphere. Processing of this information proceeds as follows.

1. For each facet, the point of intersection of each polygon with a line extended from the center of the sphere to the facet, is calculated. For example, for the processor P31-123, every plane would be examined to see whether it intersected the radial line-segment from the center of the sphere to the point

(1,30.7,122.6) (in spherical-polar coordinates). The intersections are recorded as radial distances from the center of the sphere. The surface normals of each of the planes are then calculated. The surface normal is recorded in Cartesian coordinates oriented such that the positive z-axis coincides with the radial line of the facet. the radial line. These facet-crossings and their normals are labelled so the complete surface can later be re-constructed. An example of such a label on processor P31-123 is

(F31-123 BACK)

The first part refers to the processor which initially holds the facet, and the second part to the name of the surface which was crossed.

2. The radial values at each facet are sorted in descending order.
3. The two surface normals of each plane are then re-examined. The correct one is chosen by noting that as the object is completely enclosed in the sphere, the outermost surface must face out, and subsequent normals alternate between pointing in and pointing out.

5.1.2.1 Linking Surfaces

Now that the input phase has "digitized" the surface of the object, the discrete values must be linked to record how the surface samples are related to form continuous surface patches.

Let us discuss the details in terms of a specific example. Figure 14 shows processor P31-123 surrounded by its neighbours. Notice that triangular patches are formed by P31-123 and adjacent pairs of neighbours. These patches, appropriately shaded, are

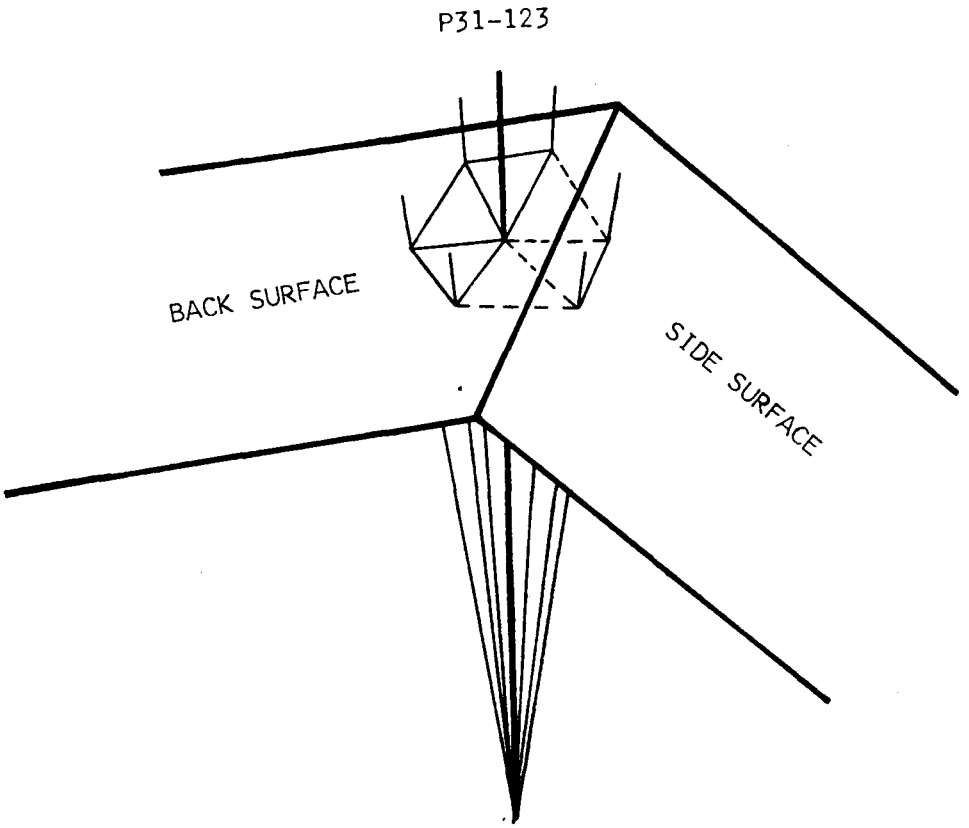


Figure 14

Example of how facets are linked to form surfaces.

destined for display on the screen. Each of these 6 patches is shared by two neighbours so every processor must handle the computation for approximately two triangles.

On the facet F31-123 we place the names of adjacent facets which the holder of F31-123 will have to assemble. In this way, we guarantee that the structure of the object is not disturbed. This is in spite of the fact that the facets themselves are handled by different processors, and that they travel independently through the network.

5.1.2.2 Setting up the Rotation

One of the design criteria was that rotations should be performed with as simple arithmetic as possible. The method used the following:

1. Given an axis of rotation, assign to each processor a "virtual-coordinate". This spherical-polar coordinate is chosen so that the axis of rotation coincides with the north-south axis of the virtual coordinate system. For each processor, coordinates in this new coordinate system are computed.
2. Later in the operation-phase, to rotate by ϕ° simply means to increment the angle of orientation of each surface patch by ϕ° in the virtual coordinate-system.

Note that the second step determines the time required for a rotation, as it is repeated for every rotation increment.

5.1.3 Calculation of Facet Positions and Intensity Values

After a rotation, the object is displayed. First, every processor examines its facet information and decides which position it corresponds to, and what intensity it should have. Consider the group of facets illustrated in Figure 15. The key to determining the (x,y) screen coordinates from a facet at (ϕ,θ) recording a surface at radius r, is the observation that the surface patch can only lie in a small cone-like region of the screen.

Each processor stores as pre-computed constants both its spherical and Cartesian coordinates. Let these be $(1,\phi,\theta)$ and (x,y,z), respectively. A surface point $\underline{P}$ held by the processor will have coordinates (r,ϕ,θ) where $0<r<=1$. The equivalent Cartesian coordinates of P are simply (rx,ry,rz) which can be calulated with one vector-multiplication.

The intensity of a surface patch is determined by the angle of orientation of the surface with respect to the light source. At each surface point, the surface normal unit vector $\underline{n}$ relative to the axis of the facet must be known. And every facet must hold a unit vector $\underline{s}$ in the direction of the light source. The angle θ between these is then given by

$$\underline{n} \cdot \underline{s} = \cos \theta \qquad (9)$$

Newman & Sproull (1979) propose that the intensity i of the the patch will then be related to the light source intensity I by the equation

$$i = I \cos \theta \qquad (10)$$

Note that if a more sophisticated shading algorithm were desired, it could easily replace this fast, albeit simple, algorithm.

Some of the surfaces of the object being modelled will be back

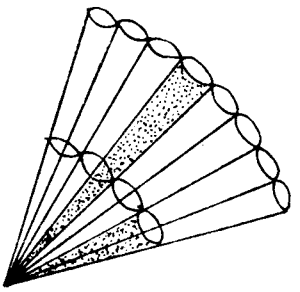


Figure 15

Subset of the facets drawn from a front view. Note that a surface which a facet sees, can only correspond to a narrow cone-like region of the screen.

surfaces, surfaces which cannot be seen from the user's viewing position. We want to avoid doing much work on these surfaces, since they are of little value. The way we can detect which facets correspond to back surfaces is to repeat the intensity calculation once more, this time operating on a unit vector in the direction of the viewer, instead of the light source. If the cosine of the angle is negative, then this facet corresponds to a surface which cannot be seen. If all three vertices of a triangle correspond to back surfaces, then the triangle is eliminated.

5.1.4 Small-angle Approximations

The calculations for facet position and intensity were based on a facet being centered exactly on the processor holding it. In general, this will not be true after a rotation. Instead, the facet will be displaced from the center by some small amount $(\delta\phi, \delta\theta)$, relative to the virtual coordinate of the processor.

The correction for the position of the facet is particularly simple. It is well known that over a small region of space, the slope of a curved surface is approximately a constant, given by its derivative. For this reason, we can pre-compute the rate of change of the coordinates of a processor as a result of a small pertubation in the ϕ direction, and a small pertubation in the θ direction. The actual change in the Cartesian coordinates can be calculated by merely scaling the pre-computed values by the appropriate amount.

A somewhat more complex technique is used to adjust the unit vector in the direction of the light source for the intensity calculation. The vector is computed at the corners of the surface patch covered by the processor, and the actual value obtained by a linear interpolation, analogous to that of Bui-Tuong (1975).

This method works everywhere except at the poles of rotation, where a processor can cover the entire range of ϕ values from 0 to 360. For this reason, the calculation near the poles involves pre-computing surface normals at increments of 20° in the ϕ direction, and linearly interpolating between the appropriate values.

5.1.5 Scaling and Translation

The points modelling the object are now ready to be scaled and translated. Since the points all lie inside the unit sphere, scaling and translation can be accomplished with the following:

Let

s be the scaling factor,

$\underline{X}$ be the vector giving the desired translation, in rectangular Cartesian coordinates, and

$\underline{P}$ be an object vector, whose endpoints are the origin and a point of interest on the object.

The scaled and translated vector $\underline{P}'$ is related to the initial object vector $\underline{P}$ by the formula

$$\underline{P}' = \underline{X} + s * \underline{P}. \qquad (11)$$

5.1.6 Perspective Transformation

According to Rogers & Adams (1976, p. 72), if the vanishing point is located at (0,0,-h) as in Figure 16, any vector $\underline{v}=(x,y,z)$ on the object becomes

$$\underline{v}' = \underline{v} * 1 / (1+z/h). \qquad (12)$$

The important feature of this equation is that the direction of the vector to any part of the object remains invariant under the perspective transformation. By this time, the scaled and translated (x,y,z) coordinates of the facet have already been determined. The scaling factor for the perspective transformation can easily be calculated from the coordinates of the point, and the transformed vector obtained by a vector-multiplication.

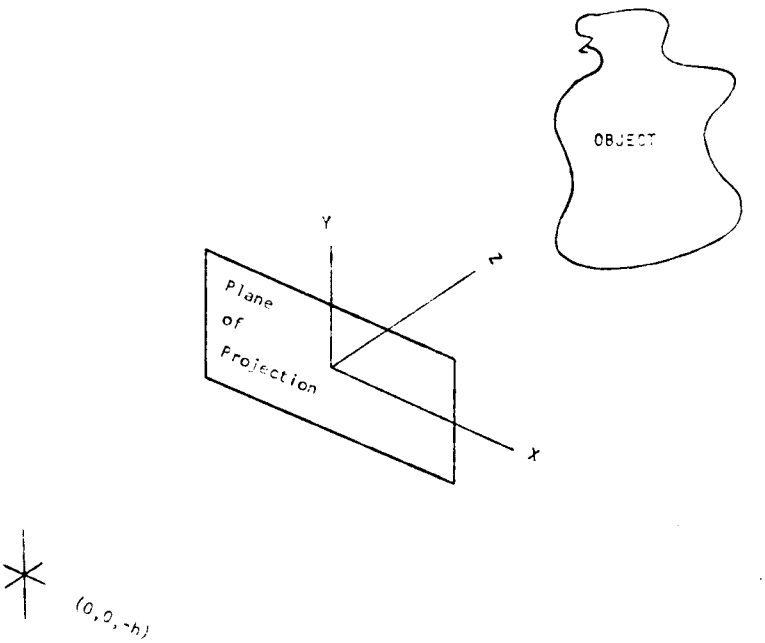


Figure 16

Construction for calculating perspective transformation. The vanishing point is located at (0,0,-h).

5.1.7 Assembling Triangles for Display

Now that the intensity and rectangular Cartesian coordinates have been determined for each facet, the facets must be assembled to form vertices of triangles. Which facets are assembled by which processors is dictated by the linking information which each facet holds. Each processor proceeds as follows:

1. Send the completed facet points (i.e. name of facet, its screen position and intensity), to each neighbour.
2. For each of the facets which the processor holds, assemble the vertices of the triangles as required.

If two facets were held by neighbouring processors in the initial input, we would expect them to remain in neighbouring positions at all times, since we are dealing with rigid objects. However, as we will see in the next section, the rotation phase does not guarantee that a processor arrives at the desired destination. A facet may in fact be removed from the correct processor by one node. This means that sometimes, a processor may not actually hold all the required vertices at the conclusion of step (2) above. In this case, a special request must be issued to all neighbours.

After the triangle has been assembled, its destination must be calculated. Recall that a processor's (ϕ,θ) coordinate served as (x,y) coordinates for the center of the screen patch which the processor governed. Now this process is reversed. Knowing where a point on the screen lies, we can determine the approximate spherical-coordinates of the processor which holds that point.

With this in mind, the vertices of the triangles which a processor has assembled are averaged, and this average value is used

as the destination point.

5.1.8 Traversing the Network

Once the facet holder has decided which positions the triangles will fill on the screen it must send the description of the triangles. We will solve the general problem of finding the least expensive route from one processor to another processor.

The cost c of sending the information is expressed by the formula

$$c = d * (c1 * s + c2) \qquad (13)$$

where

- d is the number of nodes traversed en route,
- c1 is the cost of transferring one byte of information from one processor to another,
- s is the number of bytes in the message, and
- c2 is the cost of deciding which processor is to receive an outgoing message.

The two algorithms which have been developed to decide on the exact path, illustrate that the "shortest" path (i.e. smallest d) may not be the least expensive (i.e. smallest c) path. The first algorithm, the Great-circle Algorithm with variations, follows the shortest path, but involves very expensive calculations, compared to the Small-circle Algorithm, which has simple calculations but produces a path which is usually somewhat longer.

5.1.8.1 Great-circle Algorithm

Keeping in mind that processors are spread evenly about the surface of a unit sphere, it is clear that the shortest path between any two processors follows the great circle characterized by the positions of the starting and finished processors. Our task, therefore, will be to find the chain of processors which lie closest to this shortest path.

Any plane section of a sphere is a circle. This circle is referred to as a Great-circle if the plane of intersection passes through the center of the sphere, otherwise it is referred to as a Small-circle. It is interesting to note that the problem of finding the shortest route via the Great-circle has historical overtones. Early mariners looking for the north west passage used this principle in their "Great-circle" sailing by attempting to plan their routes as far from the equator, and as close to the poles, as possible.

Definitions:

Let

f be the destination position of a message

i be the initial position of a message.

The algorithm we will follow can be loosely described as follows:

1. Calculate the rectangular Cartesian coordinates of f
2. Look up the rectangular Cartesian coordinates of i
3. WHILE message not arrived DO
   a. Find out which neighbour n of i lies closest to f
   b. Send the information to n
   c. Set i to n

The phrase "message not arrived" requires elaboration. Each

processor "covers" a pentagonal or hexagonal region of space. The computation to determine whether a message has actually arrived can, for this reason, be difficult. A simplification is introduced by assuming that the processor governs a rectangular region whose width and height are given by the maximum extent of the hexagon or pentagon. This rectangular space covers a larger area than the original shape, meaning that a message may be considered "arrived" whereas the actual destination was one of the neighbouring processors. This minor inexactness is tolerable, and saves the time required for an exact arrival calculation.

The non-trivial part of this algorithm is step 3a, finding the neighbour closest to the destination. Two different methods of roughly equal complexity have been developed to calculate this. Both involve calculating the Cartesian coordinates of the destination, $\underline{f}$, and looking up the Cartesian coordinates $\underline{i}$ of the current processor. The neighbouring processor closest to the destination is then the one with the minimum value of

$$|\underline{f} - \underline{i}| \tag{14}$$

Notice that this calculation, which must be repeated for each of the 5 or 6 neighbours, involves three multiplications and additions.

The second method for finding the processor closest to the destination involves selecting one of the processors which lies closest to the great circle characterized by the initial and destination points.

Let us define an additional quantity, $\underline{c}$, which is perpendicular to both $\underline{i}$ and $\underline{f}$. The equation for this vector uses the cross-product,

$$\underline{c} = \underline{i} \times \underline{f} \tag{15}$$

The equation for the plane of the great circle on which $\underline{i}$ and $\underline{f}$ lie can be expressed as

$$\underline{c} \cdot \underline{n} = 0 \qquad (16)$$

Since the processors are separated by 360/n degrees, we can find the processors which lie closest to the great circle by determining which neighbouring processors satisfy

$$\underline{c} \cdot \underline{n} < \cos\ (360/n) \qquad (17)$$

This subset, which usually has size two, can then be analyzed to find which one lies closer to the destination by simply subtracting angles.

Both of these methods involve considerable computation. The cost c of the message-passing system is the cost which increases as a function of the number of processors, and it is what causes the saturation effect illustrated earlier in Figure 4. Clearly, we need an inexpensive calculation.

5.1.8.2 Small-circle Algorithm

The second method of traversing the network is less expensive than the Great-circle Algorithm, despite the fact that it does not guarantee the shortest path. The method relies on finding the shortest way of traversing the plane of the display device, as illustrated previously in Figure 12. We will follow the Small-circle defined by the starting and destination points, whose plane is perpendicular to the axis of the sphere. We can consider the first angular component of a processor's coordinate to be a distance in the x-direction, and the second a distance in the y-direction. Clearly, the equation for the path we seek is that of a straight line,

$$y = m x + b \qquad (18)$$

where m is the slope, and b is the y-intercept. We seek to identify the set of processors which lie closest to the line.

Definitions

Let

I be the (ϕ,θ) coordinates of the starting processor, and

F be the (ϕ,θ) coordinates of the destination.

The algorithm is:

1. Set the difference D between the starting and the finishing processors to F - I.
2. WHILE D > angular width of processor (i.e. message not arrived) DO
   a. Set N to that neighbour of I which, will decrease the second coordinate of D and whose slope is the closest m.
   b. Set I to N
   c. Set D to F - I

Note that this algorithm requires only one division and several comparisons for each processor on the path. However, the shortest path is not guaranteed.

A sample comparison between the Great-circle and Small-circle Algorithms is illustrated in Figure 17. In the diagram, we are looking down at the north pole of a sphere. Imagine the initial and final positions with different longitude, but identical in latitude. The Great-circle Algorithm would send the message along the shortest path, i.e., across the pole. The Small-circle Algorithm would carry the message around the globe along the small circle of constant

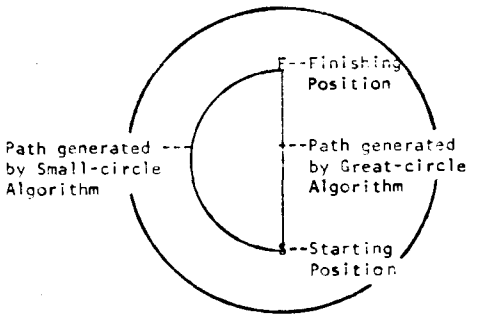


Figure 17

Comparison of Great-circle Algorithm and Small-circle Algorithm. A message must be sent from one side of the north pole to the other. The Great-circle Algorithm finds the path across the pole, while the Small-circle algorithm travels around the pole. Note that the worst-case performance of the two algorithms is identical.

longitude.

It should be clear from this discussion that the worst case for the Small-circle algorithm is equal to the worst case for the Great-circle Algorithm, i.e., the radius of the graph. The Small-circle Algorithm is much less expensive to compute, but has the disadvantage that in general messages remain in circulation longer. This, in turn, increases the chance of the message density building up somewhere in the network, resulting in a bottleneck.

5.1.9 Turning on Pixels

Once a triangle has arrived at the correct processor, the corresponding pixels must be turned on. A small complication is that

a given triangle may span the pixels held by several adjacent processors.

The procedure is first to resolve the triangle into scan-lines according to a polygon-filling algorithm similar to that of (Crow(1976)). In the simulation, Gourand shading was added to the filling algorithm (Newman & Sproull(1979)). As described earlier, each processor "knows" which pixels it is responsible for, and which neighbours handle adjacent pixels.

An iterative process for correctly distributing the pixels among the processors is:

1. Decompose the triangle into scan-lines.
2. WHILE a processor p has scan lines which don't belong to it DO
   a. Let p keep that part of the scan-line which belongs to it.
   b. Send the remaining portion of the scan line to the appropriate neighbour.

Once a process has only pixels which belong to it, the pixels are individually inserted into a z-buffer via the algorithm:

```
For each pixel

    If z-value of new pixel is < z-value of previous value
       then
          store   new z-value and
                  new intensity-value

repeat
```

5.1.10 Rotating Object

Each time the screen is refreshed, the object is rotated by a small amount. This makes the rotation smooth. To initialize the rotation about a given axis, each processor calculates its

"virtual-coordinate", which is aligned so that the axis of rotation coincides with the north-south axis of the virtual-coordinate system. Every facet holds a local-coordinate describing its position relative to the virtual-coordinate of its processor. Although the calculation of the virtual-coordinate is time-consuming as it requires numerous trigonometric calculations, the calculation is only performed once for each new axis of rotation.

The calculations required during the rotation are very simple. Rotation requires adding the angle of the rotation increment to the first angular component of the local-coordinates. Then the message is sent to its destination, in a fashion similar to that for sending the triangles from the facet-holders to the pixel-holders. The amount of time required for a rotation is proportional to the rotation's angle.

5.2 Simulation and Results

The algorithms outlined in the previous section were implemented under the MTS operating system on an IBM 4341 using LISP. The primary purpose of the simulation was to aid in the development of the algorithm and to study the message-passing system, rather than to produce high-quality images.

Processors are represented by atoms, and the information a processor "knows" is stored on its property list. A complete inventory of properties is given in Appendix A. As properties are added to the 482 simulated processors, memory is used up very quickly due to the fact that LISP uses 8 bytes for each conscell. The memory limitations became an insurmountable problem. Due to hardware restrictions, the amount of memory for LISP programs and

data is limited to 255 pages, or approximately 1 megabyte.

By removing properties when they were no longer needed, it was possible to complete all phases of the simulation, with the exception of the actual turning on of the pixels, in the manner described in Section 5.1.9. Not running this last part is not a major loss, since it is a problem which has been thoroughly studied, and is well understood. To obtain pictures, nonetheless, the polygon-filling algorithm was applied sequentially to the triangles when they arrived at the appropriate pixel-holder. The z-buffer data was handled by a FORTRAN program. Images were generated by a PL/1 program using overprinting on the line-printer to produce grey-shades.

5.2.1 Simulating Parallelism

Clearly the IBM 4341 is not capable of supporting the kind of parallelism required for the graphics system. Instead, an operation which would in reality be executed simultaneously on each processor was handled sequentially using a LISP MAPping function. The amount of CPU time required in an actual implementation could clearly not be measured using this technique. Furthermore, only an estimate of the processor utilization could be determined, since the simulation had to synchronize steps which would not necessarily have to be syncronized in reality.

Despite the problems inherent in a simulation, it was nonetheless possible to study the message-passing system. Messages were handled in "rounds". In each round, a processor passed on the first of its outgoing messages to the appropriate neighbour. During the rotation, and during the facet-to-pixel-holder transfer,

the following data were collected:

1. For each message, the number of nodes visited between its source and its destination was measured. This value should always be less than or equal to the radius of the graph.
2. At each round, the percentage utilization of the processors was measured, that is, the number of processors which had a message to send.

The minimum number of rounds in which the message-passing system can stop is clearly one greater than the length of the longest message. If messages are delayed due to bottlenecks anywhere along the way, the number of rounds required will be larger.

5.2.2 Some Examples

The first test object we will consider is a sphere. It is representative example, as its surface varies smoothly, and all possible surface orientations to the light source are represented.

First, the sphere was displayed without rotation. Figure 18a shows a graph for the number of messages versus message length for the facet-to-pixel-holder transfer. Most of the messages had to travel a distance of between 2 and 6 nodes, with a maximum distance of 8. Figure 18b shows the percentage utilization versus round. At the outset, approximately 55% of the processors had messages to send. This is reasonable, since approximately half the surfaces were entirely on the back of the sphere, and were removed since their surface normal pointed away from the viewer. The utilization decreased in a near-linear fashion as time progressed. It took 10 rounds to dispose of all outstanding messages. This is one more

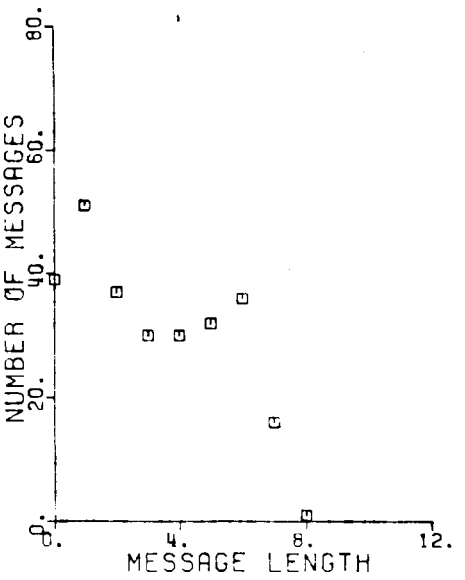


Figure 18a

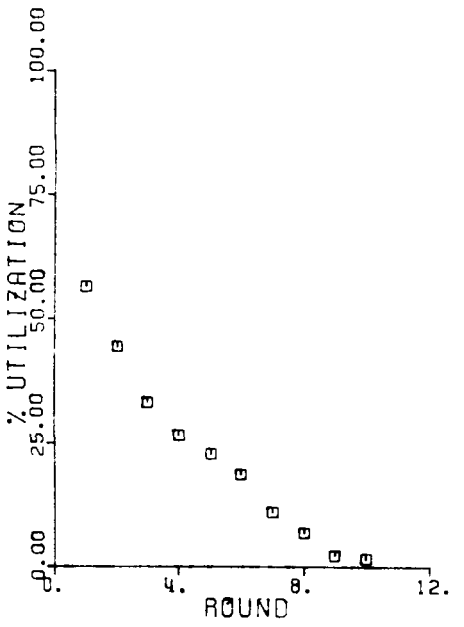


Figure 18b

Graphs of message-passing for draw-phase of unrotated sphere. Graph (a) shows number of messages versus message length. Graph (b) shows the percentage utilization of the processors versus round.

than the minimum 9 rounds. This result indicates that the information flowed through the network without undue delays.

The sphere which was produced is illustrated in Figure 19. As expected, the intensity varies smoothly from the center of the sphere outward, and the sphere appears symmetric.

Next, the sphere was rotated by 6°, which at 30 frames/second would represent a rotational speed of 1/2 a revolution per second. Figure 20 shows the graphs of the message-passing behavior during the rotation. The longest messages travelled 1 node, while it took 2 rounds until all messages had arrived. This result indicates that a slow rotation, which may be typical for many applications, is very inexpensive.

Two further rotations were tested: 30° representing a moderate speed of 3 revolutions per second, and 180°, representing the maximum speed of 15 revolutions per second. The graphs for the rotations are illustrated in Figures 21 and 22, respectively. In each case, all messages arrived at their destination in only a small number of rounds more than the minimum possible. The 30° rotation took 5 rounds to complete with a maximum message length of 3. The 180° rotation had some messages which travelled the maximum possible distance of 20 nodes, and took 26 rounds to complete.

The draw phase for the rotated spheres had results which were identical to those for the unrotated spheres, and the image of the rotated spheres was also virtually identical to the image of the unrotated spheres. For this reason, the graphs are not repeated.

The next example which was run was a cube. The image was drawn looking perpendicular to the front face. Figure 23 shows the

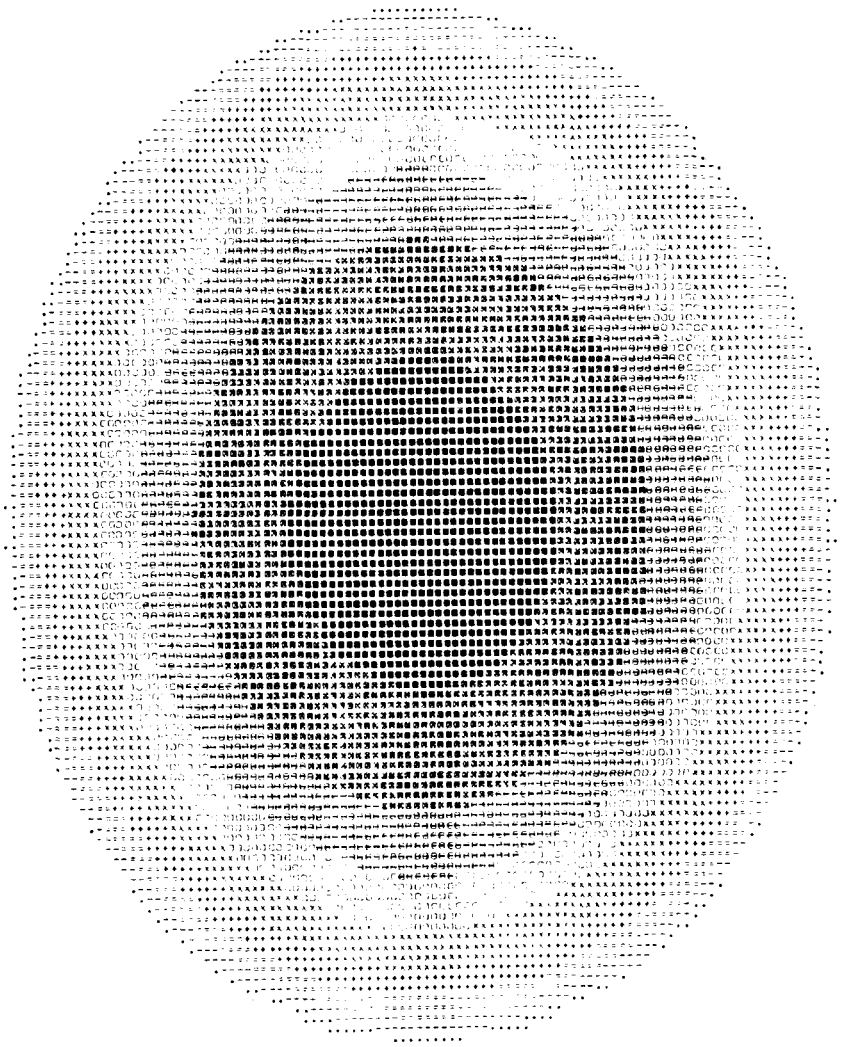


Figure 19

Image of unrotated sphere. The image is longer than it is wide as the line-printer prints 10 horizontal columns/inch but only 8 vertical lines/inch.

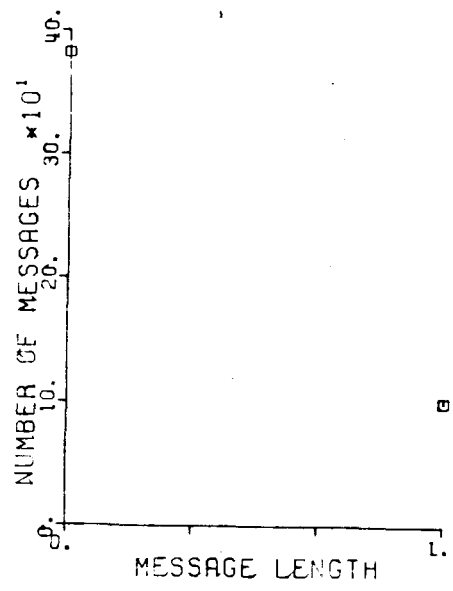


Figure 20a

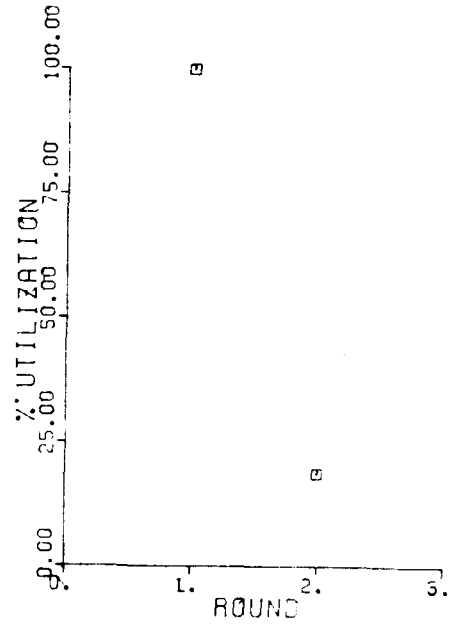


Figure 20b

Graphs of message-passing during rotation of sphere through 6°. Graph (a) shows number of messages versus message length. Graph (b) shows the percentage utilization of the processors versus round.

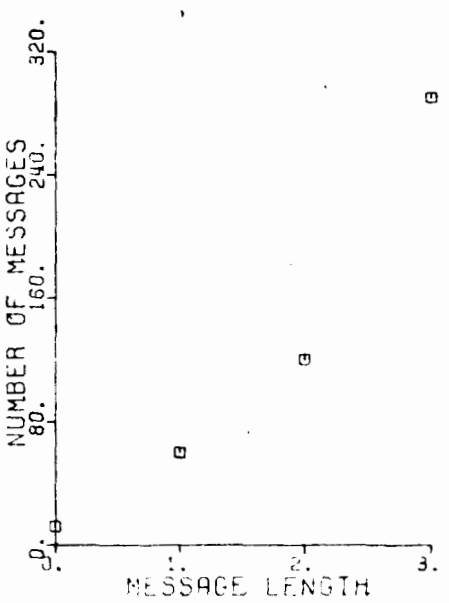


Figure 21a

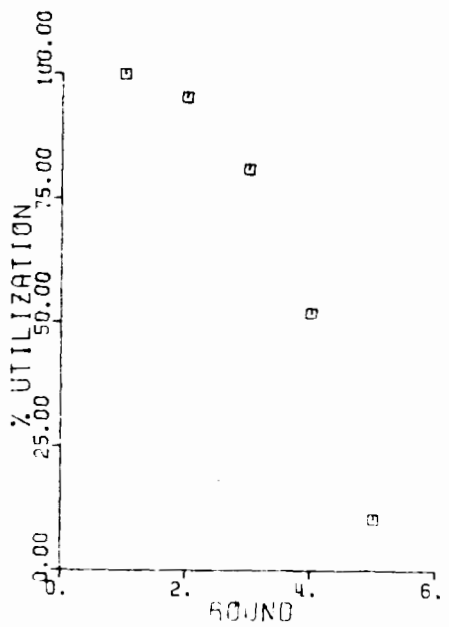


Figure 21b

Graphs of message-passing during rotation of sphere through 30°. Graph (a) shows number of messages versus message length. Graph (b) shows the percentage utilization of the processors versus round.

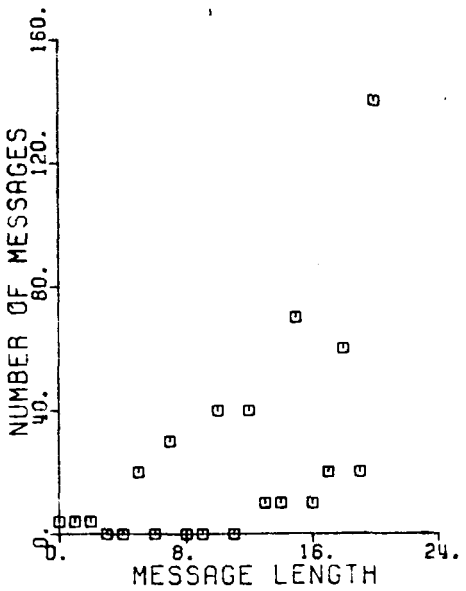


Figure 22a

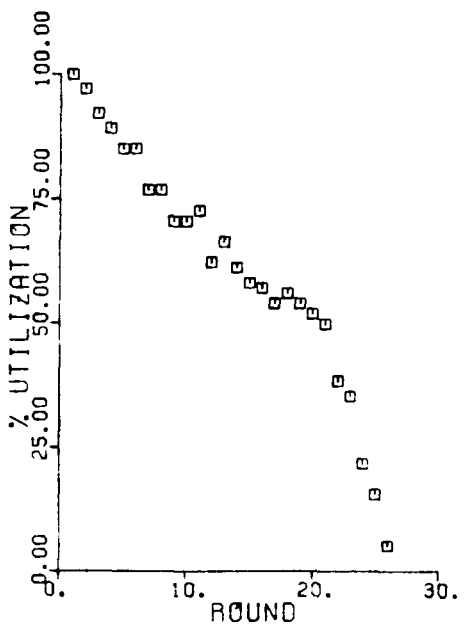


Figure 22b

Graphs of message-passing during rotation of sphere through 180°. Graph (a) shows number of messages versus message length. Graph (b) shows the percentage utilization of the processors versus round.

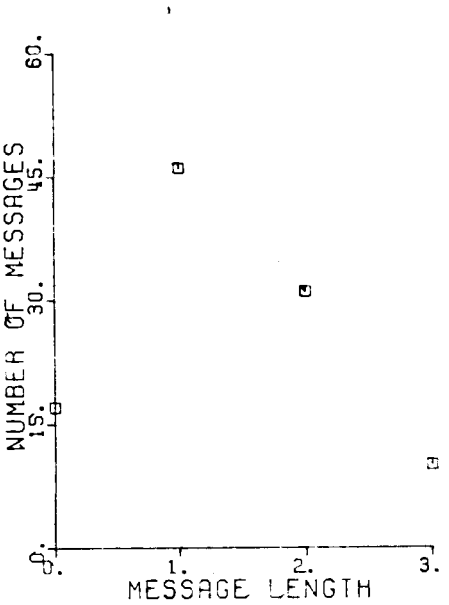


Figure 23a

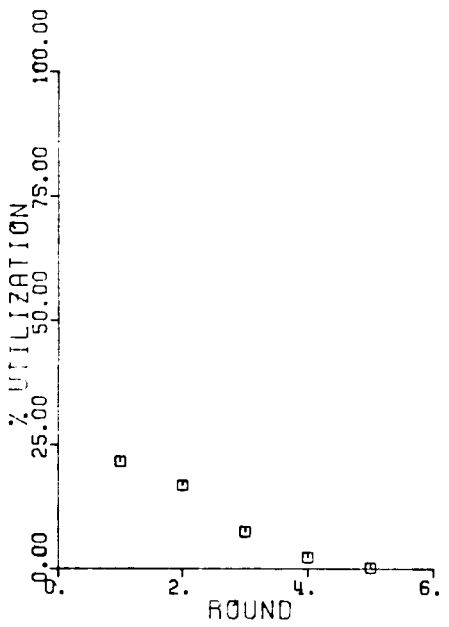


Figure 23b

Graphs of message-passing for draw-phase of unrotated cube. Graph (a) shows number of messages versus message length. Graph (b) shows the percentage utilization of the processors versus round.

graph of the number of messages versus message propagation length. The longest message had a length of only 3, compared to 6 for the sphere. This is because the processors toward the middle of the screen were also the ones which held the facets modelling that part of the cube. Figure 23b shows the graph of utilization versus round. All messages arrived by the 5th round, meaning that messages were not seriously delayed by bottlenecks.

The image which was produced is illustrated in Figure 24. The intensity is uniform everywhere except at the edges. In the lower half of the image, the intensity drops off smoothly to the white background. This is because the triangles which cover that part of the screen have one vertex on the front of the object, and two on the side or bottom. The interpolation algorithm makes the intensity vary smoothly over the triangles.

The top half of the image is more interesting. The sides are uniformly dark, since it just so happened that facets were positioned very close to the edges of the cube. At the top, the triangles used alternated between having one vertex on the front and two on the sides, and the other way around. The image produced illustrates that the method of object representation is most effective with objects whose surface varies smoothly. Sharp edges are rounded off, particularly with a resolution of only 482 facets. Two methods of solving this problem have emerged. The first is simply to increase the number of facets. This increases the resolution, but at the expense of processing time. The second solution is to adjust some facet positions so they lie close to the edges of the polygons. During the initial input of the polygons, a facet lying near an edge

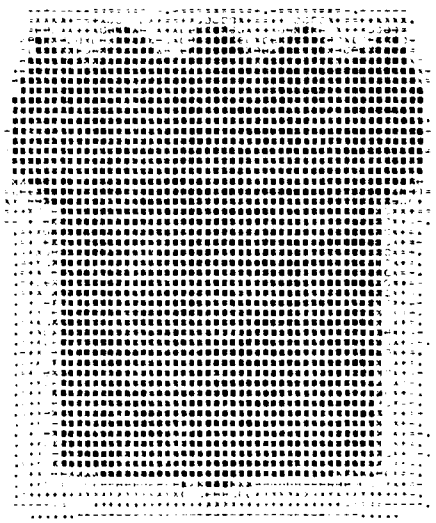


Figure 24

Image of unrotated cube. Blemishes at the edges are due to the facet positions being far from the edges of the cube.

could be moved by simply giving it a local displacement coordinate relative to the processor. This solution requires little more processing time.

5.3 Performance Analysis

The task of estimating the performance which can be expected from a multi-processor system is complicated by the fact that the performance is often dependent on the input data itself. The analysis for the graphics system is plagued by similar problems. How many twists and caves are there in the object? How much of the screen does the image cover? For these and related questions, we will assume a set of "reasonable" input data for the analysis itself. Following this, we will see what happens to the equations as less "reasonable" data is used.

Throughout the analysis, we will assume that operations such as additions, subtractions and comparisons require one time unit, u, and that multiplications and divisions require 10 units. In this way, the analysis is independent of the cycle time of the processor used. Typical values for u might range from 1 to 5 micro-seconds. Further, it will be assumed that data can be transfered between adjacent processors at the rate of 1 byte per m time units u.

The timing analysis will be treated in four sections: (1) rotation, (2) facet position and intensity calculation, (3) facet-to-pixel-holder transfer, and (4) polygon filling.

5.3.1 Rotation

Clearly the cost of rotation is proportional to the number of nodes a message visits en route to its destination. This, in turn, is dependant on the angle of the rotation, ϕ, and the radius of the graph, hence the number of processors.

Let us define the function

$$R(\phi,p) \tag{19}$$

which gives the number of rounds elapsed before the longest message arrives for a rotation through $\phi°$ with p processors. The results of the simulation indicate that the number of rounds required is only marginally higher than the radius of the graph. If we estimate the delay in the longest message to be 20%, we can use equation 8 to arrive at

$$R(\phi,p) = (\mathrm{mod}(\phi,180)/180)*1.2*.82*p**0.5 \tag{20}$$

which reduces to

$$R(\phi,p) = (\mathrm{mod}(\phi,180)/180)*p**0.5 \tag{21}$$

The number of time units required for each round will be m for each byte in the message, plus the cost of deciding which processor should be the recipient (1 division, 3 subtractions and 3 comparisions = 16u using Small-circle Algorithm). Therefore, the cost Cr of doing a rotation of b bytes through ϕ degrees using p processors is

$$Cr = (16 + b*m)*(\mathrm{mod}(\phi,180)/180)*p**0.5\ u \tag{22}$$

5.3.2 Initial Position and Intensity Calculation

Listed in Appendix B is an overview of the operations required to perform the initial position and intensity calculations. At the conclusion of this phase, the triangles for display will be ready to be

sent to the holder of the corresponding pixels. According to the estimates, 600u will be required for each crossing observed by each facet. Thus, the cost Ci for this phase is

Ci = (500+48*m) * (number of crossings observed) u (23)

5.3.3 Facet-to-pixel-holder Transfer

The cost of this phase can be estimated in a manner similar to that of the rotation. We will assume that there exists at least one message which must travel the maximum possible distance, and that the message-passing requires 20% more time to complete than the minimum possible.

With each triangle description using 48 bytes, and 16u being required for the decision about which neighbour to send an outgoing message to, the cost Ct for this phase becomes

Ct = (48*m+16)*p**0.5 u (24)

Note that if the object is centered on the screen, then no message will have to be passed more than 1/2 of the maximum distance so that the value of Ct from Equation 24 can be halved.

5.3.4 Turning on Pixels

The problem of estimating the performance of uni-processor scan-line algorithms has been thoroughly studied. Parke (1980) has the most recent results. His analysis produced the following timing estimates:

a. 9u per pixel,

b. 61u per scan-line; we will use a value of (61+20*m)u since scan-line messages may have to be sent to neighbours (the

calculation for content of these messages are assumed to be part of the scan-line algorithm itself),

c. 62u per edge, and

d. 71u per polygon.

Since we are always dealing with triangles, (c) and (d) can be combined to give 257u per triangle. Note however, that these results are valid only for triangles which have an area of at least several pixels. For very small triangles, no interpolation is required. Estimates indicate that the cost for such very small triangles is approximately 4u per triangle.

At this point, we will have to apply restrictions on the object we are dealing with. These restrictions simplify the subsequent algebra, but as will be discussed later, do not significantly alter the outcome. Let us assume:

1. The object uses some fraction x of the screen, where a typical value of x might be x=0.5.
2. The triangles have equal sizes and are distributed evenly about the screen, hence spread evenly about one x'th of the processors.

Let us assume that the number of triangles is t and that we are dealing with a 256x256 raster graphics device. The number of triangles Pt per processor is then

$$Pt = (t/x)/p \tag{25}$$

and taking right-angled triangles as representative, the average number of pixels Tp per triangle is

$$Tp = (256**2)*(x/t) \tag{26}$$

and the average number of scan-lines Ts per triangle is

$$Ts = (2*Tp)**0.5 \tag{27}$$

Now the number of pixels Pp per processor is

$$Pp = Pt*Tp \tag{28}$$

or

$$Pp = (256**2)/p \tag{29}$$

The number of scan-lines per processor is

$$Ps = Pt*Ts. \tag{30}$$

By substitution,

$$Ps = (362/p)*(t/x)**0.5 \tag{31}$$

The total time for scan-conversion, Cs, per processor, is therefore

$$Cs = 9*Pp + (61+20*m)*Ps + 257*Pt \tag{32}$$

or

$$Cs = 9*(256**2)/p + (61+20*m)*(362/p)*(t/x)**0.5 + 257*t/(p*x) \tag{33}$$

Now that we have an expression for the performance, let us go back and see what happens when we relax the restrictions made previously. Suppose that x were small, i.e., the entire object were concentrated in a small region of the screen. The scan-conversion algorithm becomes very inexpensive. Because x is small, the triangles comprising the object are very small, hence the cost Cs' is

$$Cs' = 4 * t \tag{34}$$

which is clearly less than Cs from Equation 33 for realistic combinations values for t, x and p.

The small-triangle problem can also be treated at its source. Before sending triangles to the pixel-holders, processors holding facets could collapse groups of small triangles into fewer large ones. This has the added advantage that less data flows through the

network in the facet-to-pixel-holder transfer.

The other restriction we stated is that the triangles have equal sizes. Often this does not hold, since a surface which is not perpendicular to the viewer will be composed of small triangles. Here again, the small triangles can be handled with the inexpensive scan-conversion algorithm, resulting in an overall cost which is even less than that of Equation 33.

5.3.5 Overall Cost

The overall cost in terms of CPU time is

$$T = Cr + Ci + Ct + Cs \qquad (35)$$

As an example, let us use the following parameters:

1. 482 processors,
2. m, the number of cycles required to transmit one byte of information equals 1u,
3. one facet per processor
4. a maximum of two radial values per facet,
5. a rotation through 6 degrees,
6. image centerred and using 1/2 of the screen,
7. 500 triangles.

The costs for these conditions are:

Cr = 66u

Ci = 1096u

Ct = 800u

Cs = 3680u

Therefore, T = 5642u. For u=5 micro-seconds, this result is still within the limit of 1/30 of a second required for real-time processing.

However, it is important to note that the foregoing calculation should be treated only as an order-of-magnitude estimate of the cost.

The performance of the modelled system improves in a near-linear fashion with the number of processors. The reason for this can easily be visualized by noting that the more processors we have, the smaller the part of the screen each processor must govern, hence the less pixels each must handle. Further, the more processors we have, the smaller are the hexagonal patches on the surface of the geodesic dome which each processor governs. Only the message-sending costs increase with the number of processors. However, this cost contributed only 15% to the total cost in the above example, and this cost increases only as the square root of the number of processors.

6. Conclusions

The results of the simulation indicate that the messages flowed through the network in an orderly fashion. Both the rotation phase, and the facet-to-pixel-holder phase completed the message-passing in only marginally more than the minimum possible time. It is noteworthy that the utilization of the processors during the message-passing began at between 50% and 100%, and decreased in a near-linear fashion with time. It is important that the percentage utilization vs. round (time) graphs do not have any long tails. They would indicate bottlenecks which would significantly slow the system down.

It is interesting to compare the system presented in this thesis with other attempts at producing images quickly. Clarke's system is able to transform polygons in parallel, but these polygons had to be input sequentially to a system such as Parke's multiple-processor z-buffer. Parke's z-buffer, in turn, can make efficient use of only 16 to 64 processors to display the polygons. The system presented here combines these two phases, object manipulation and object display. The number of processors can be increased to improve the performance as long as the time required for message-sending is less than the time required for object manipulation and display.

The sample calculation for the theoretical performance analysis showed that the message-sending due to the rotation and the facet-to-pixel-holder transfer contributed only 15% to the overall display time. This is significant, since this parallel processing system must compete with uni-processors which do not require message-sending. In the system presented here, the number of

processors can be increased to improve the performance as long as the total time required for the message-sending is less than the time required for object manipulation and display.

The simulation provides experimental evidence in support of the viability message-passing scheme where theoretical results would be difficult to obtain. But we must treat the results with guarded optimism. There are still hardware design problems which must be worked out before a prototype can be built. In particular, problems concerning communication protocols must be solved. How exactly will message be handled by the hardware? How will a processor detect the arrival of a message?

Another problem area is the handling of the pixel-data. Real-time raster graphics requires that an enormous amount of data be transferred between the processors holding the z-buffer and the screen, in a short space of time. Can this data be read by the video-generator without affecting the current status of the processor handling the pixel-data?

This thesis has laid a foundation for a graphics system using parallel processing for the rotation and display of three dimensional objects. A simulation has demonstrated the viability of the method, and an order-of-magnitude estimate has indicated that it may be possible to produce images in real time. The next step is the development of the details of the hardware necessary to support an environment similar to that of the simulation. The goal of real-time display of complex images may then be realized.

Appendix A: Summary of Processor Information Content

Every processor must hold certain information to be able to interact with the other processors in the system. Presented below is an inventory of the properties each processor must have. The properties are organized by the name of the system-phase in which they are required. Included is an estimate of the storage requirement (B=Bytes, KB=1000B), and an explanation of the property.

General

Name	2B	Name of processor.
Type	1B	Gives processor type: either pentagon or hexagon.
Coordinate	4B	Angular (ϕ,θ) coordinate of processor.
Neighbours	12B	List of 5 or 6 neighbouring processors.

Object Description

Cell	21B	Gives information on surfaces crossed between processor and center of sphere. Cell size varies: 21 bytes per crossing observed (2 byte radial value, 6 byte normal, 3 byte name, 12 bytes linking information).
Triangles	6B	Which pairs of neighbours join this processor to form triangles used in display of object.

Drawing

P-Screen	6B	Gives projection of this processor onto screen.
Light	6B	Unit vector in direction of light-source, relative to axis of processor.
Viewport	6B	Unit vector in direction of view-port, relative to axis of processor.

Z-buffer

Screen	80B	Description of scan-line-segments held by this processor.
Pixels	1KB	Pixel-values and z-buffer.

Rotation

Virtual-coordinates	6B	Angular (ϕ,θ) coordinate of processor aligned so that the axis of rotation coincides with the north-south axis in the virtual-coordinate system.
V-Angularwidth	4B	Gives width of processor in terms of (ϕ,θ) in virtual-coordinate system.
V-Differences	48B	Gives differences in (ϕ,θ) between processor and each of its neighbours (in virtual-coordinates), as well as the ratio $\delta\phi/\delta\theta$.

Facet to Pixel-holder Transfer

Angularwidth	4B	Gives width of processor in terms of (ϕ,θ) in virtual-coordinate system.
Differences	48B	Gives differences in (ϕ,θ) between processor and each of its neighbours, as well as the ratio $\delta\phi/\delta\theta$

Appendix B: Timing Estimates for Facet Calculations

Presented below is a summary of the operations which must be performed for each frame. The estimates are given in units "u", where operations such as addition and subtraction are assumed to take 1u of CPU time, and multiplications and divisions are assumed to require 10u. It is assumed that to send a message, m time units are required for each byte in the message.

Initial Position	1 vector multiplication	30u
Scaling	1 vector multiplication	30u
Translation	1 vector addition	3u
Small-Angle	2 multiplications 4 additions	24u
Perspective	4 divisions 4 additions	44u
Intensity	13 multiplications/divisions 16 additions	146u
Back-surface	same as Intensity	146u
Neighbour Messages	48 byte length	48*m u
Assemble Triangles	20 comparisions 5 multiplications 2 additions	72u

Total (495 + 48*m)u

Approximately (500 + 48*m)u

List of References

Anderson, G. A. & E. Douglas Jensen (1975). "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples", Comput. Surv. 7, 4 (Dec. 1975), pp. 197-213.

Bechtolsheim, A. & F. Baskett, (1980). "High-Performance Raster Graphics for Microcomputer Systems", SIGGRAPH '80, Aug., 1980, pp. 43-47.

Bezier, P. (1972). Numerical Control--Mathematics and Applications, A. R. Forrest (trans.), London, Whiley.

Brown, C. M. (1978). "Representing the Orientation of Dendritic Fields with Geodesic Tesselations", Univ. of Rochester, TR-13.

Brown, C. M. (1979). "Fast Display of Well-Tesselated Surfaces", Comput. & Graphics 4, pp. 77-85.

Bui-Tuong, Phong (1975). "Illumination for Computer Generated Pictures", Comm. ACM 18, 6 (June, 1975), pp. 311-317.

Calvert, T. W., J. Chapman, & A. Patla (1980). "The Integration of Subjective and Objective Data in the Animation of Human Movement", SIGGRAPH '80, Aug., 1980, pp. 198-203.

Chu, Wesley W., Leslie J. Holloway, Min-Tsung Lan & Kamal Efe (1980). "Task Allocation in Distributed Data Processing", Computer 13, 11 (Nov. 1980), pp. 57-69.

Clarke, James (1980). "A VLSI Geometry Processor for Graphics", Computer 13, 7 (July, 1980), pp. 59-68.

Clarke, James H. (1976). "Hierarchial Geometric Models for Visible Surface Algorithms", Comm. ACM 19, 10 (Oct. 1976), pp. 547-554.

Crow, F. C. (1976). "The Aliasing Problem in Computer Synthesizes Shaded Images", Comm. ACM 19, 11 (Nov. 1977), pp. 799-805.

Crow, Franklin (1980). "Toward More Complicated Computer Imagery", Comput. & Graphics 5, pp. 61-69.

DeBoor, C. (1972). "On Calculating with B-Splines", J. Approx. Theory 6, pp. 50-62.

Feldman, Jerome A. (1979). "High Level Programming for Distributed Computing", Comm. ACM 22, 6 (June, 1979), pp. 353-368.

Funt, Brian V. (1981). "Multi-processor Rotation and Comparison of Objects", to be presented at the 7th Int. Joint Conf. Art. Int., Vancouver, Canada, Aug., 1981.

Harris, William F. (1977). "Disclinations", Sc. Am. 237, 6 (Dec. 1977), pp. 130-145.

Heath, Thomas L. (1956) The Thirteen Books of Euclid's Elements, New York, Dover, Vol. 1-3.

Herman, G. T. (1979). "Representation of 3D surfaces by a large number of simple surface elements", MIPG26, SUNY at Buffalo.

Jones, Anita K. & Peter Schwartz (1980). "Experience Using Microprocessor Systems--A Status Report", Comput. Surv. 12, 2 (June, 1980), pp. 121-165.

Newman, William M. & Robert F. Sproull (1979). Principles of Interactive Graphics, 2nd. Ed., New York, McGraw-Hill.

Parke, Frederic I. (1980). "Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems", SIGGRAPH '80, Aug., 1980, pp. 43-56.

Requicha, Aristides A. G. (1980). "Representations for Rigid Solids: Theory, Methods, and Systems", Comput. Surv. 12, 4 (Dec. 1980), pp. 437-464.

Rogers, David F. & Alan J. Adams (1979). Mathematical Elements for Computer Graphics, New York, McGraw-Hill.

Rubin, S. & Whitted, T. (1980). "A 3-Dimensional Representation for Fast Rendering of Complex Scenes", SIGGRAPH '80, Aug. 1980, pp. 110-116.

Sequin, Carlo H. (1981). "Doubley Twisted Torus Networks for VLSI Processor Arrays", Proc. 8th. Ann. Symp. on Comput. Arch., pp. 471-480.

Wittie, Larry D. (1980). "Communication Structures for Large Networks of Microcomputers", SUNY at Buffalo, preprint submitted for publication, 27 pp.