OPTIMIZING HARDWARE UTILIZATION OF PARALLEL-PROCESSING BIT-MAP GRAPHICS DISPLAYS

by

Anson Yiu Cho Lee

B.Sc. (Hon.), Simon Fraser University, 1983

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the School

of

Computing Science

SIMON FRASER UNIVERSITY

August 1987

APPROVAL

Name: Anson Y. C. Lee

Degree: Master of Science

Title of thesis: Optimizing Hardware Utilization of Parallel-Processing Bit-Map Graphics Displays

Examining Committee:

Chairman: Dr. Binay Bhattacharya

Prof. H.K. Reghbati
Senior Supervisor

Dr. Rick Hobson

Dr. Tom Calvert

Dr. Rob Cameron
External Examiner

Date Approved: 28/August/86

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

Optimizing Hardware Utilization of Parallel-Processing Bit-Map Graphics Displays

Author: _____
(signature)

Anson Yiu Cho Lee
(name)

Aug. 11, 1987
(date)

ABSTRACT

Among all the operations in the scene rendering process, scan-converting an image is perhaps the most computationally intensive task. For conventional graphics displays, the utilization of the image memory bandwidth is a key factor in determining their image rasterization performance. The symmetric memory-to-screen mapping scheme is evaluated for its effectiveness to cope with the problem.

Owing to the drop of hardware cost, parallel-processing architectures have been proposed for many advanced high-performance graphics displays. Optimizing their hardware utilization is the main theme of this thesis. To evaluate these systems, a study of their architectural structures is done. This study leads to a general model for graphics display architectures. Based on this model, a new concept called Regional-Rasterization is proposed. With this technique, a more cost-effective parallel-processing solution to the image rasterization problem can be obtained.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to Professor Hassan K. Reghbati for his guidance and supervision throughout my graduate work.

I would also like to thank Dr. Thomas W. Calvert, Dr. Richard F. Hobson, Dr. Robert D. Cameron, Dr. Binay K. Bhattacharya, and Dr. Thomas K. Poiker for their assistance.

Finally, the help of Miss. Venus Law in typing some chapters of the thesis is sincerely appreciated.

This research was supported by the Natural Sciences and Engineering Research Council of Canada under Grant No. A0743.

DEDICATION

To my parents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF GRAPHS

CHAPTER 1

INTRODUCTION

1.1 Description of the Problem

"Computer graphics" had its starts with X-Y plotters. The technology was soon extended to include calligraphic (line-drawing) CRT systems based on refresh-vector hardware and, at a later date, direct-view storage tubes followed by higher performance refresh-vector systems.

Meanwhile, a separate "image processing" technology had been following its own evolutionary path. Instead of dealing with lines and points, randomly positioned anywhere on the display surface, image processing was based on a rectangular array of picture elements or *pixels*. Digital information defining the state of each pixel was stored in a random-access memory and used to generate a television-type, raster-scan CRT display - in monochrome or full color.

The division between vectorgraphic (also known as calligraphic) "computer graphics" and raster-scan "image processing" has now been bridged by the new raster display technology. Low-cost memories have made it economically feasible to assemble high-resolution, fine-detail raster images that all but eliminate the objectionable stairstepping of vectorgraphic lines when displayed on a raster-CRT screen. The ability to fill areas with solid color (and shading) makes it useful in more

applications of different nature. The result is *raster scan graphics* combining the full-color, pixel-by-pixel control potentials of image processing and the line-drawing capabilities of vectorgraphics in a single display system.

Unfortunately, the move from calligraphic to raster displays has brought new problems. For calligraphics, only the endpoint coordinates of lines needed to be stored, so memory requirements were held to a minimum. Refresh-vector writing speed made it possible to "animate" the display and to create interactive systems which allowed the operator to control the display in "real time" through such input devices as lightpens, joysticks, and digitizer tablets. Again, however, only the endpoints needed to be recalculated with each refresh cycle, minimizing the need for high-speed computational capabilities.

The proper value at each pixel is a function of the data base (the simulated environment), the viewing position and orientation of the simulated viewer, and the location(s) of the light source(s) in the simulated environment. The environment is most often described as a set of solids in the environment (Euclidian three-space) coordinate systems. Each object is usually described by a set of planar tiles ("polygons") which form its various surfaces. Figure 1.1, from [SUTH74], shows the boundaries of a set of polygons defining the surface of a 3-D object. Other methods of object description are sometimes used -- e.g., as collections of geometric solids or curved surfaces [FUCH79]. We assume hereon that the common planar-polygon

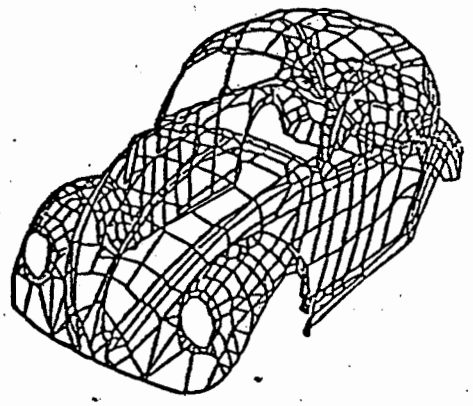


Figure 1.1 Polygon mesh model of an object (from [SUTH74]).

descriptions are used.

Each polygon is described as a list of vertex coordinates (x, y, z in 'world' coordinates) and colors (values of Red, Green, Blue that specify the intrinsic color of the vertex). A transformation engine operates on the coordinates of the vertex list for each polygon, transforming the polygon to 'eye' coordinates in response to user input from joystick, trackball, or some similar device. Next, polygons (or portions of polygons) that are outside the viewing pyramid are clipped and perspective division is performed to transform 'eye' coordinates to 'screen'

coordinates. Finally, a lighting model calculator modifies each vertex's intrinsic color according to the position and intensity of light sources. The output of the "front-end" pipeline is still a list of polygon vertices, but with vertex coordinates and colors transformed to the proper value for display (see Figure 1.2).

In advanced color graphics systems, the rasterizer performs a series of steps needed to translate a list of polygon vertices into a smooth-shaded, rendered, digital image, with hidden surfaces properly removed, and perhaps anti-aliased to reduce pixelization artifacts. In order to compute the Red, Green and Blue color (shade) values for a particular pixel, the system has to determine:

(a) which, if any, polygons map onto this pixel's area,

(b) the details about the precise part of this closest polygon which maps onto the pixel -- its assigned color (R,G,B), its angle and distance from the light sources(s), and its angle and distance to the viewer, and

(c) which one from this set is closest to the viewer (and thus is the one visible obscuring all the other polygons).

The problem to transform a scene defined in high level polygon descriptions into a raster image is called the *image rasterization problem.*

A long-standing goal of researchers in computer graphics systems has been the development of real-time three-dimensional modeling systems. These systems, which produce a realistic image

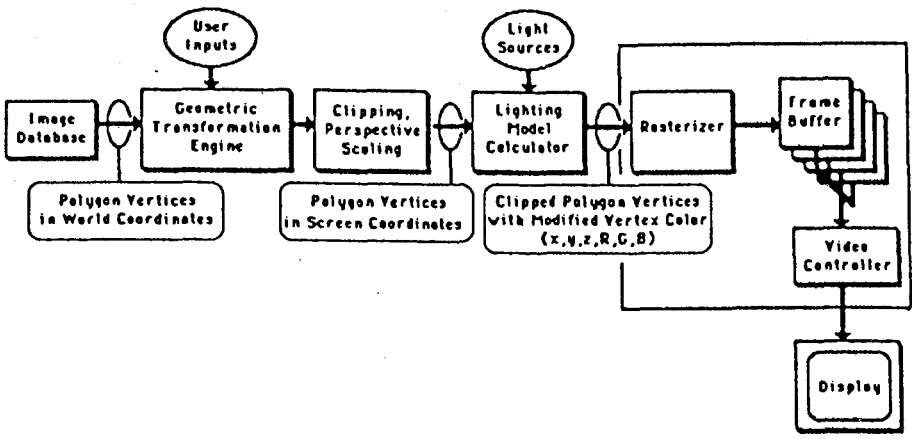


Figure 1.2 Scene rendering process pipeline

of a simulated three-dimensional environment, have a wide variety of potential uses -- from flight simulators for pilot training to interactive Computer Aided Design (CAD) systems. The most sophisticated of these systems produces, in real-time, images of startling reality. The only limitation to widespread use of these systems has been their prohibitive costs, ($500,000 and up). Thus, virtually the only uses today are those for which there is no real alternative -- e.g., simulating maneuvers in gravity-free space or training-simulators for pilots of sophisticated airplanes. If such modeling systems could be

provided at significantly lower costs, it is safe to presume that their use would become dramatically more widespread.

A short examination of the computational expense of the problem suffices to justify the complexity and expense of current systems which solve it. A video image normally consists of a matrix of pixels between 512 to 1024 rows (scan-lines) with 512 to 1024 pixels in each scan line. The image is then simply a set of some 250,000 to 1,000,000 pixels, each of which is computed by a pel-process. The problem at hand is simply executing these up to 1 million pel-processes each time the image is scanned out, usually not less than 30 times per second.

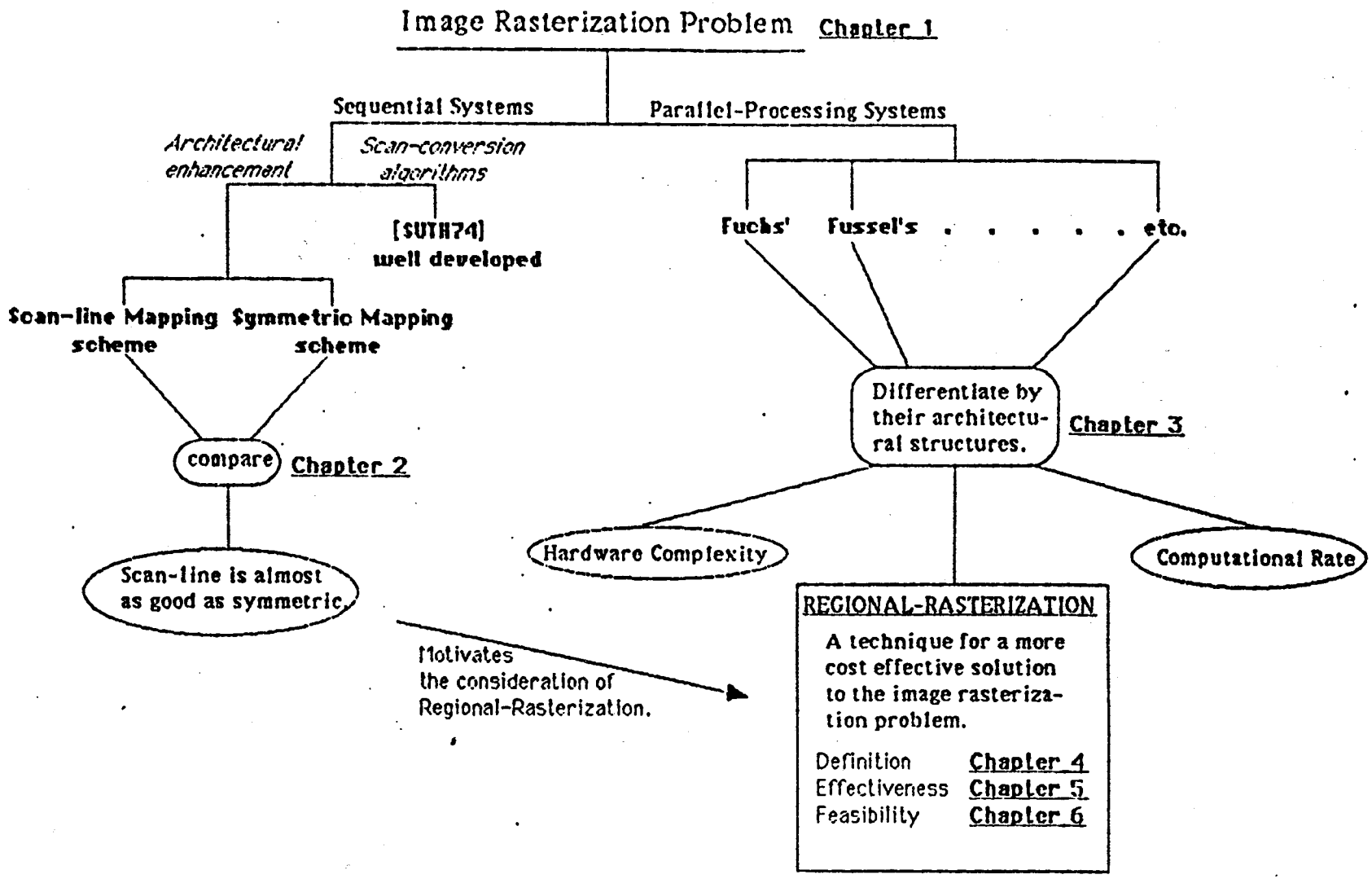
In this thesis, the impacts of various architectural characteristics of image rasterizers on their efficiency in performing image rasterization are investigated. The goal is to search for a more cost-effective alternative to the current solutions for the image rasterization problem.

1.2 Thesis Outline

In Figure 1.3, the thesis outline is schematically illustrated; and various chapters of the thesis are conceptually mapped to different issues of the image rasterization problem.

In Chapter 2, memory-to-screen mapping schemes for frame buffer based image rasterizers are investigated. A mapping scheme defines the memory access geometry of a frame buffer.

Figure 1.3 Thesis Outline



Owing to the limited bandwidth and the memory contention between the image update and video refresh activities, the frame buffer updating efficiency becomes critical to the performance of image rasterization. Although the access bandwidth is unaltered by a change in the memory-to-screen mapping scheme, the efficiency of drawing/reading image patterns into the frame buffer is affected. Therefore, with a more appropriate mapping scheme, the image rasterization process can be sped up. In Chapter 2, two dominant types of mapping schemes, the scan-line mapping scheme and the symmetric mapping scheme, are discussed. A mathematical model is built to evaluate these schemes. Three benchmark tests are conducted to compare the two schemes.

A general representation for graphics systems is described in Chapter 3. The representation identifies an image rasterizer according to the characteristics and nature of its architecture, and thus provides a means to compare different image rasterizers. With this representation, the hardware complexity and computational power of a parallel-processing graphics display can also be formulated. These two factors are crucial to the evaluation of the rasterizer's cost-effectiveness.

A novel concept for image rasterization called *Regional-Rasterization* is proposed in Chapter 4. (In the thesis, Regional-Rasterization is sometimes called "the Regional-Rasterization technique".) Motivations, functions, and overhead of this new technique are discussed in that chapter. With this novel concept, the cost-effectiveness of

parallel-processing architectures for image rasterization can be improved substantially.

Chapter 5 focuses mainly on the performance of the Regional-Rasterization technique. To evaluate the technique, a performance model is created. This model is a logical representation of the Regional-Rasterization technique. A set of simulated work loads is applied to the model. These work-loads are generated based on a new image model. Estimates of two important performance measures for the Regional-Rasterization technique are then obtained. According to these measures a gross prediction of the technique's effectiveness is obtained.

In Chapter 6, a discussion on the feasibility of implementing the Regional-Rasterization technique is given. A design of a key component, the local objects identifier, is presented. The feasibility is further verified by applying the technique to two typical parallel-processing graphics display architectures. The conclusion of the thesis can be found in Chapter 7.

CHAPTER 2

MEMORY-TO-SCREEN MAPPING SCHEMES AND IMAGE RASTERIZATION

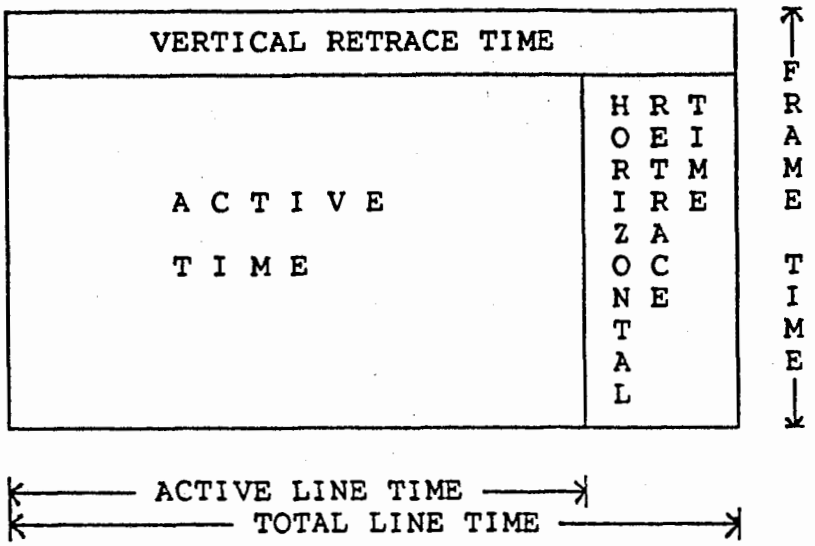
In raster-scan graphics displays, each pixel of the raster is assigned a fixed memory location in the frame buffer and can be addressed randomly. The allocation strategy of the memory cells in the frame buffer to the raster pixels is known as the *memory-to-screen mapping scheme*.

The contents of the frame buffer are updated by a device called the *display processor*. Another device, the *video signal generator*, also needs to read data from the frame buffer in a regular and predictable fashion in order to refresh the screen of the display. Owing to the limited access bandwidth of the frame buffer, an efficient way to draw patterns becomes crucial to the overall performance of a frame buffer based image rasterizer. Because of the differences in their pixel-to-memory assignment geometry, unequal number of frame buffer accesses may be required to draw a similar pattern for various mapping schemes [CHOR82, GUPT81, MATI84, SPRO83]. The two most popular mapping schemes are the *scan-line mapping scheme* and the *symmetric mapping scheme*. The major objective of this chapter is to carry out a quantitative investigation into the impacts of these mapping schemes on the frame buffer update efficiency.

2.1 Video Refresh Requirements

The *refresh rate* of a display is the number of complete images, or frames, drawn on the screen in one second. A typical refresh rate for CRT displays is 60Hz. As the electron beam sweeps from the left to the right, some finite amount of time is required to return from the right extreme back to the left. This activity is called *horizontal retrace,* and the time taken is termed as horizontal retrace time. Similarly, after a raster is completely scanned, the electron beam needs to return to the top margin. It is called the *vertical retrace* and the time required is the vertical retrace time. During retraces, the electron beam is turned off such that no new image data is displayed. The video signal is said to be blanked, and the respective times are called horizontal and vertical *blanking intervals*. When not blanked, the signal is said to be *active*.

Figure 2.1 shows how total frame time is composed of vertical retrace time, horizontal retrace time, and active time. Total line time is equal to frame time minus vertical retrace time divided by the number of visible lines per frame. Active line time is the value obtained by subtracting the horizontal retrace time from the total line time. As described earlier, a scan line consists of a certain number of pixels. The active line time is the total amount of time required to display the pixels of a scan-line. Hence, each pixel in a scan line is displayed for a period called the *pixel cycle time* which is the



$$\text{Pixel cycle time} = \frac{\dfrac{(1/\text{refresh rate}) - \text{vertical retrace}}{\text{visible lines per frame}} - \text{horizontal retrace}}{\text{pixels per line}}$$

Figure 2.1 Timing partition of a video frame.

active line time divided by the number of pixels per scan line.

In order to satisfy the video refresh requirements, the video generator must access the frame buffer regularly. To change the contents of the frame buffer, the display processor must compete with the video generator for memory cycles. Hence the availability of *effective bandwidth* to the display processor

is always a critical factor which determines the performance of an image rasterizer. Parallel memory I/O, achieved by using multiple memory chips to implement the frame buffer, can always increase the effective bandwidth. By implementing the frame buffer with display RAMs[1], the overall memory bandwidth can almost be doubled. Although differences in memory-to-screen mapping strategies do not change the total memory bandwidth, they may alter the average image updating efficiency. In the following section, the two most popular memory-to-screen mapping schemes will be introduced.

2.2 Memory-To-Screen Mapping Schemes

In most cases, the frame buffer is implemented with RAM chips. *Words* in RAM chips are usually arranged in a two dimensional array. Each word in a chip has a unique address (r,c), where r and c are the row and column address of the word in the array respectively. For brevity, in this chapter it is assumed that the word length is one bit. In order to store the large amount of pixel values, n memory chips are assumed to be used. In the rest of the chapter, we assume n to be a perfect square such that $n=k^2$ where $k=1,2,\cdots$.

---

[1] A display RAM is a quasi-two-ported RAM such that its primary port is always available for image updating while data for video refresh can be read from its secondary serial port [MATI84].

2.2.1 The Scan-Line Mapping Scheme

Since all the chips can be addressed independently, it is possible to access *n* pixels simultaneously (*n* is the number of chips used). In the scan-line mapping scheme, memory cells from each chip with the same address are assigned to a string of adjacent pixels along a scan line (see Figure 2.2). This scheme is motivated by the high output rate required by the video signal generator. Theoretically, the video refresh controller should have a new pixel value ready for every pixel cycle which has a very short period. For example, the actual required pixel rate for a 1024x1024 60Hz non-interlaced display is approximately 1 pixel per 11.5 ns. With a typical memory cycle of 150 ns for single bit reads (e.g. the 4164 64Kx1 DRAM), more than 13 pixels must be read simultaneously in a single memory cycle to achieve the desired pixel rate. Scan-line mapping is particularly good for high output parallelism. In the example shown in Figure 2.2, an effective pixel rate of 1 pixel per 9.375 ns, is potentially achievable.

Writing an *h-by-w rectangular pattern* onto the screen is done by updating an appropriate *h-by-w grid of pixels* in the frame buffer. A 1-by-*n* grid of pixels in the frame buffer can be updated extremely fast in the scan-line mapping scheme provided that the grid is well matched with the *word boundary*. Assuming the pattern is already available in a buffer or a latch, which is called the *pattern buffer*, a single frame buffer access given an appropriate word address to all the chips is adequate.

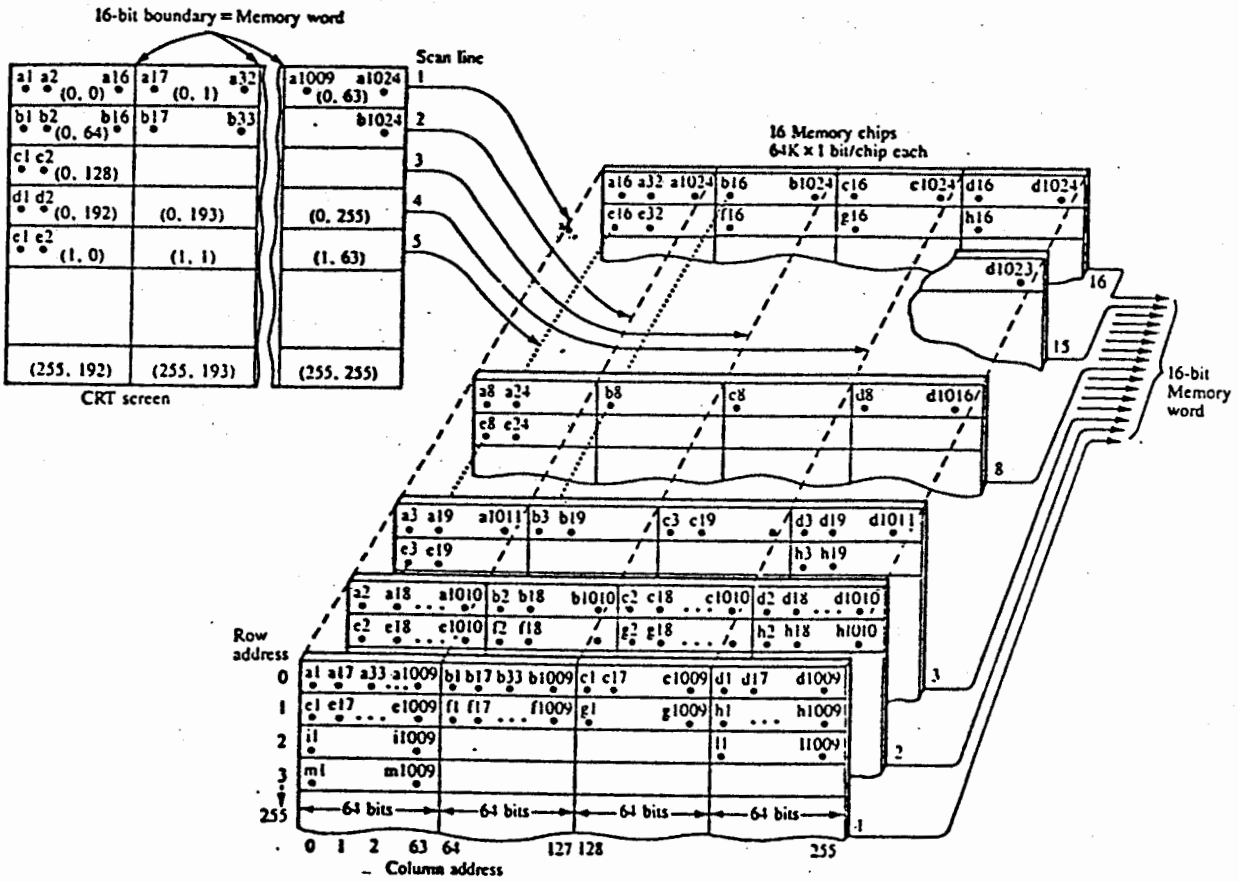


Figure 2.2 Scan-line mapping scheme for a system implemented with 16 64Kx1 RAM chips (from [MATI84]).

However, it becomes more complicated when the 1-by-*n* grid spans across a word boundary (see Figure 2.3). Addresses sent to chips which contain pixels on the right side of the word boundary must be adjusted. In order to be able to update a random 1-by-*n* grid in one memory cycle, address adjusting circuitry must be included in the frame buffer. Moreover, data alignment hardware is also required to match the data with the word boundary before

it can be correctly written into the destination grid.

Figure 2.3 describes the procedure to write a 1-by-16 pattern into an arbitrary 1-by-16 grid. The leftmost pixel of the pattern is to be written to the location (r',c') of chip 5. Similarly, the next 11 pixels are to be stored into the cells (r',c') of chips 6, 7, ..., and 16 respectively. Starting from the 13th pixel, the remaining 4 pixel values are to be stored at location (r',c'+1) for chips 1, 2, 3, and 4. The data in the pattern buffer must firstly be aligned with the word boundary. This can be done by shifting the pattern data around until the leftmost cell holds the pixel to be written into chip 1. The next step is to give appropriate addresses to all the chips. In the example, chip 1,2,3 and 4 are given the address (r',c'+1) and the others the address (r',c').

The major drawback of the scan-line mapping scheme is its poor performance in dealing with *vertical objects* such as a vertical vector. (In this chapter, a rectangular pattern whose width is larger than its height is classified as a *horizontal object*, otherwise it is a *vertical object*.) Because of the mapping strategy, all the pixels along a vertical vector are stored in the same chip [GUPT81, MATI84, SPRO83]. Therefore, pixels along the vector must be written sequentially.

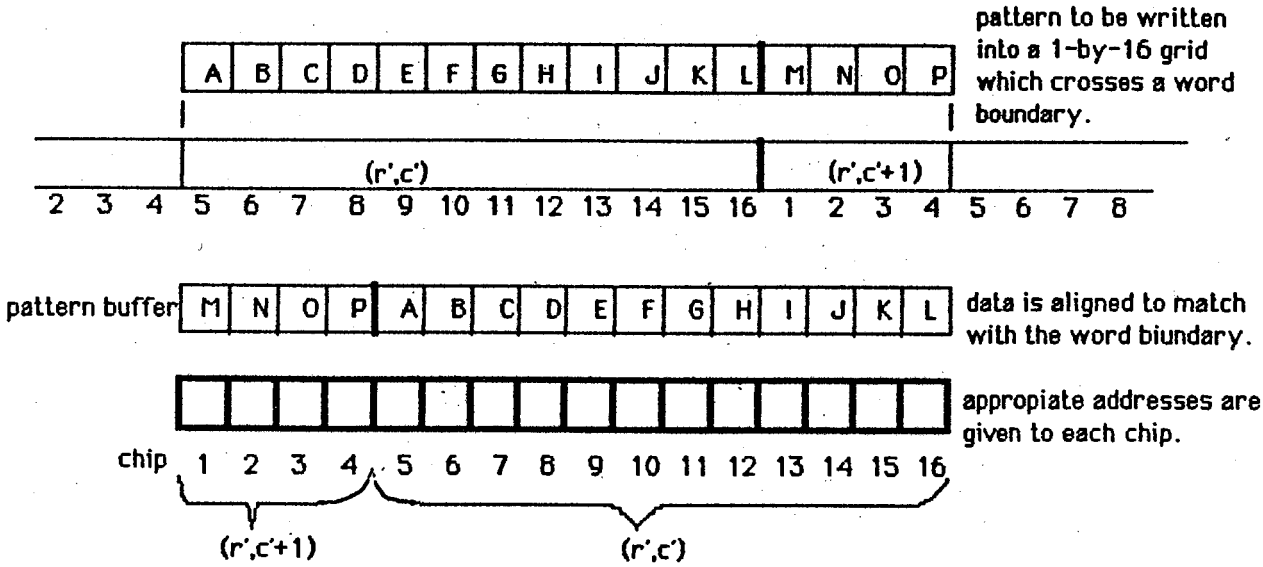


Figure 2.3 Address transformation and data alignment mechanism in the scan-line mapping scheme in a 16-chip system.

2.2.2 The Symmetric Mapping Scheme

In the scan-line mapping scheme, a 1024x1024 display may require at most 1024 memory cycles to draw a vertical vector. This rate is too slow for some real-time applications. In order to lessen the required memory cycles for writing a vertical object, the symmetric mapping scheme arranges memory chips in a symmetric square [GUPT81,SPRO83]. For example, in Figure 2.4 the memory cells with the same address are mapped to a 4-by-4 square grid on the display [MATI84]. This design gives rise to a system

of more consistent performance.

An obvious drawback of symmetric mapping is a slower update rate for horizontal objects. For applications which generate more horizontal objects, the scan-line mapping scheme is definitely better than the symmetric mapping scheme. Another disadvantage is that the number of pixels which can be provided to the video signal generator in parallel is decreased considerably. If $n=k^2$ chips are used in the scan-line mapping scheme, the n pixels read from location (r',c') of all the chips lie along a scan line. A single read from the frame buffer by the video signal generator supplies the next n pixel values of the same scan-line. In the symmetric mapping scheme, the n pixels are read as k rows of k pixels. Among them only one row of pixels belongs to the current scan-line and is immediately needed by the video signal generator. The remaining $(k-1)$ rows of pixels are either discarded or saved up in a *refresh buffer* [SPRO83].

The refresh buffer must be capable of holding $2k$ complete scan-lines [SPRO83]. It is used as two k-line sections that alternately supply data to refresh the display and are replenished from the frame buffer. During the display of one set of k lines, information for the next set of k lines is transferred from the image memory to the refresh buffer. Shift registers or FIFO memories are commonly used to build the refresh buffer, for their speed to shift out data.

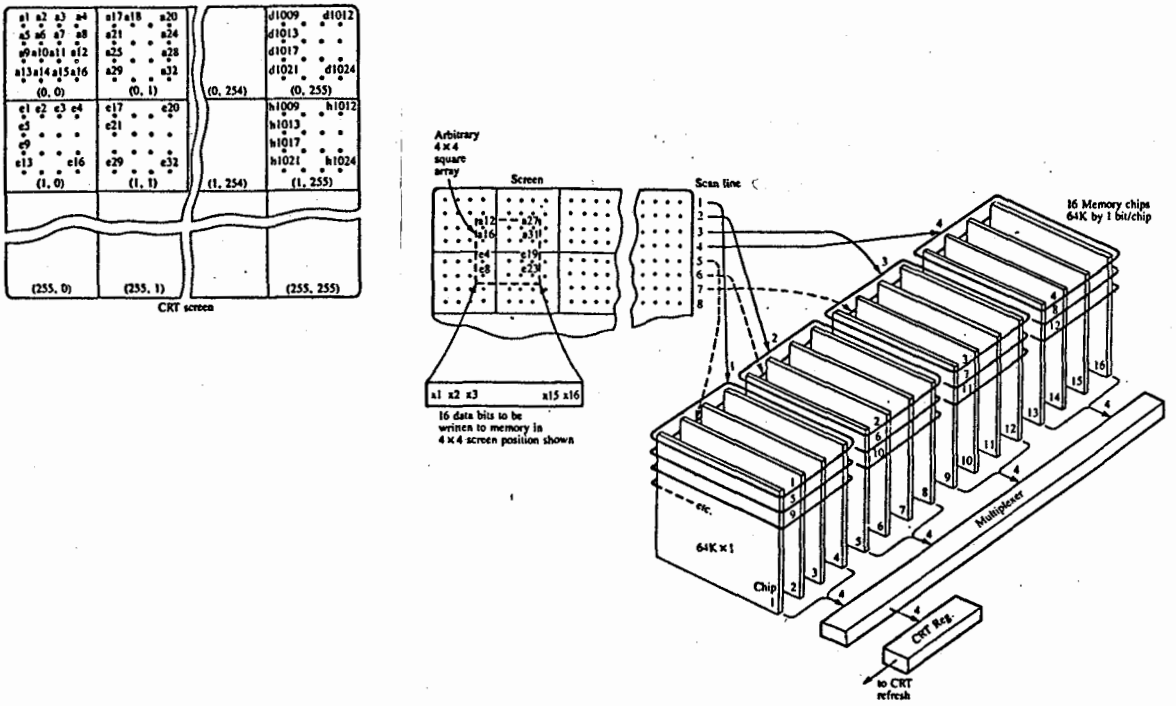


Figure 2.4 Symmetric mapping scheme for a system with 16 64Kx1 RAM chips (from [MATI84]).

With a refresh buffer, the average achievable pixel rate remains the same as that of the scan-line mapping scheme. But the extra cost involved to implement the buffer is considerable, since thousands of pixels have to be stored in expensive high speed memories.

Furthermore, a *k*-by-*k* square pattern may now mismatch to a horizontal as well as a vertical word boundary. Hence more complicated address transformation and data aligning mechanism must also be employed (see Figure 2.5). This adds even more to the cost of the system.

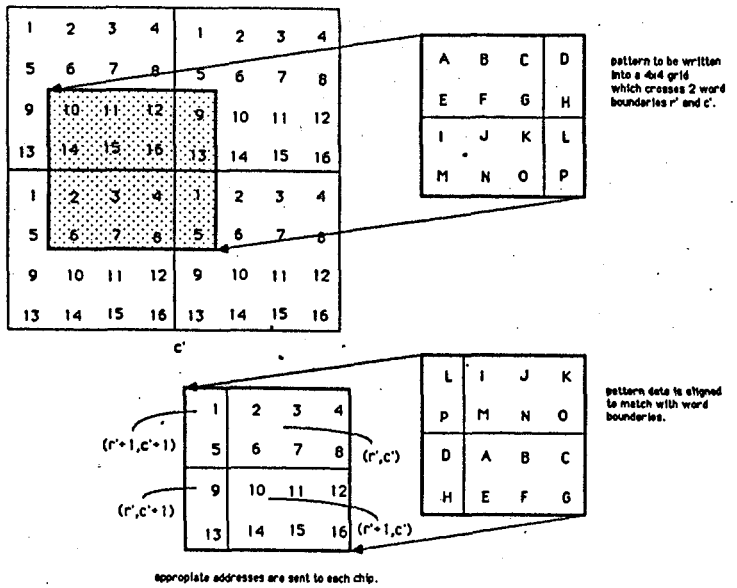


Figure 2.5 Address transformation and data alignment needed to update an arbitrary 4-by-4 grid in a 16-chip system.

To justify the cost effectiveness of a mapping scheme, a quantitative comparison is conducted. In the following section, a mathematical model is created to evaluate the expected number of accesses needed to update an arbitrary rectangular grid of pixels.

2.3 Mathematical Model for Analysis

One of the most important objectives of this chapter is to explore the average case performance of the scan-line mapping scheme and the symmetric mapping scheme. The measurement of their performance is made in terms of the number of memory cycles required to update an arbitrarty *h-by-w grid* of pixels.

The analysis is performed for an M*x*M non-interlace whose frame buffer is implemented with $n=k^2$ memory chips. There are also proper data alignment and address adjustment facilities provided. Thus, an arbitrary 1-by-n grid can be updated in a single memory cycle for the scan-line mapping scheme, and an arbitrary k-by-k grid of pixels for the symmetric mapping scheme.

In our model, only *rectilinear rectangular patterns*[2] are considered. Drawing an h-by-w rectilinear rectangular pattern always requires an update to the corresponding h-by-w grid of pixels in the frame buffer. Let $G\{x,y,h,w\}$ be the event that the h-by-w rectangular grid of pixels, whose top leftmost corner is at (x,y), is to be updated. Let P($G\{x,y,h,w\}$), the probability of the event $G\{x,y,h,w\}$, be $P\{x,y,h,w\}$. With the given screen format, h and w may vary from 1 to M. Since $(x+w)$ and $(y+h)$ do not exceed M+1, x and y may vary from 1 to $(M+1-w)$ and $(M+1-h)$ respectively. Assume graphical objects are distributed fairly

---

[2] A rectilinear rectangle is a rectangle whose edges are parallel to the x and y axes.

uniformly over the display, and thus there are no preferred positions. This means any two grids are equally likely as long as they have the same h and w measurements. In this case $G\{x,y,h,w\}$'s are independent of x and y. In the rest of the chapter, $G\{h,w\}$ will denote the event that an h-by-w grid at an arbitrary location is to be updated, and its probability will be denoted by $P\{h,w\}$. Notice that the event $G\{h,w\}$ is equivalent to the event $\mathbf{U}\ (G\{x,y,h,w\})$[3], and $P\{h,w\}$ is equal to $\Sigma\ (P\{x,y,h,w\})$. Also let $N\{x,y,h,w\}$ be the number of frame buffer accesses required to update an h-by-w grid at (x,y). With proper data alignment and address adjustment facilities (see Figures 2.3 and 2.5), it takes the same amount of memory cycles to update an h-by-w grid of pixels at any arbitrary (x,y). Therefore, $N\{x,y,h,w\}$ can be represented by $N\{h,w\}$ for all (x,y).

Since h and w may vary from 1 to M, there are M^2 different $G\{h,w\}$'s, M^2 different $P\{h,w\}$'s, and M^2 different $N\{h,w\}$'s. The expected number of frame buffer accesses required to update an arbitrary grid is:

$$E = \sum_{h=1}^{M} \sum_{w=1}^{M} P\{h,w\} * N\{h,w\}$$

---

[3] The union of a number of discrete events means the OR of all these events.

2.3.1 Memory Accesses Requirement

For scan-line mapping, any 1-by-n grid can be updated with a single frame buffer access (see Figure 2.3). When an h-by-w rectangular pattern is written into the frame buffer, the number of accesses required is equal to the number of 1-by-n grids needed to be updated. Obviously, the h-by-w grid spans over h scan-lines and invloves $\lceil w/n \rceil$ words. This implies that at least $h * \lceil w/n \rceil$ 1-by-n grids have to be updated in order to write the larger h-by-w grid. Hence, $N\{h,w\}$ for scan-line mapping is:

$$h * \lceil w/n \rceil$$

For symmetric mapping, k-by-k ($k=\sqrt{n}$) pixels can be updated in parallel. The number of accesses required to update an h-by-w grid is the minimal number of k-by-k grids that it covers, which for symmetric mapping is:

$$\lceil h/k \rceil * \lceil w/k \rceil$$

2.3.2 Probability Distribution of $P\{h,w\}$

$P\{h,w\}$ defines the probability of updating an arbitrary rectilinear h-by-w grid. In the random sample space, there are M*M possible events because both h and w can take any integer from 1 to M. Hence, there are M*M different possible ordered pairs (h,w).

There are two major considerations affecting the distribution of $P\{h,w\}$. They are (1) the statistical behavior of

the shape of the graphics objects, and (2) the statistical behavior of the size of the graphics objects. Unfortunately, there does not exist a general statistical structure to describe a graphics environment. For simplicity, let us assume that h and w are two identical and independent random variables whose distribution are defined by the probability density function δ. Then $P\{h,w\}$ will be equal to $\delta(h)*\delta(w)$. By denoting the E's for scan-line and symmetric mapping schemes with E_{scan} and E_{sym}, the following expressions are obtained:

$$E_{scan} = \sum_{h=1}^{M} \sum_{w=1}^{M} \delta(h)*\delta(w)*h*\lceil w/n \rceil \qquad (2.1)$$

$$E_{sym} = \sum_{h=1}^{M} \sum_{w=1}^{M} \delta(h)*\delta(w)*\lceil h/k \rceil * \lceil w/k \rceil \qquad (2.2)$$

2.3.3 Validity of the Mathematical Model

In general, an image may be composed of any number of arbitrary patterns. Let the set of displayable geometric patterns of an application be $S=\{G_1,G_2,\ldots,G_w\}$. Assume the probability of occurrence of G_i is $P(G_i)=P_i$, and the number of frame buffer accesses required to draw G_i is N_{1i} for the scan-line mapping scheme, and N_{2i} for the symmetric mapping scheme, respectively. By elementary statistics, the expected number of frame buffer accesses required to draw an image for the scan-line and symmetric mapping schemes are $E_{scan}=\Sigma\ P_i*N_{1i}$ and $E_{sym}=\Sigma\ P_i*N_{2i}$, respectively. The immediate concern of the analysis is to determine the P's, N_1's, and N_2's. Unfortunately, it is infeasible to obtain the exact P's. The probability of occurrence of a certain geometric pattern is highly dependent on

the nature of the application. For example, when a graphics system is used to display business data, alphanumeric characters might be the dominant geometric objects. In a CAD system, rectilinear rectangles are quite common. Therefore, it is difficult to obtain the underlying probability distributions of the geometric patterns without prior knowledge about the application. For simplicity, in our mathematical model, only rectilinear rectangles are considered.

Let the set of rectangles be $R=\{R_i \mid i=1,2,...j\}$ and $N(R_i)$ be the number of memory cycles required to draw the R_i, R_i R. If the relative frequency of R_i (which can be expressed as the probability $P(R_i)$) can be obtained, then the expression $\sum_i P(R_i)N(R_i)$ will be the expected number of memory cycles required to draw an image. Therefore, if the $P(R_i)$ and $N(R_i)$ for all R_i R can be obtained, the model should be valid.

Unfortunately, the relative frequencies of the R_i's are unknown. Hopefully, better estimations can be obtained by using statistical inferences, such as statistical regression [KLEN78]. Nevertheless, to conduct statistical estimation may require a large volume of sample data. In addition, there is still no guarantee on the accuracy of the estimates derived from such a large amount of sample data. Therefore, no attempt was made to carry out these kinds of estimation processes.

In the following section, the mathematical model is applied to three benchmark cases. The p.d.f. for R used in the three

benchmark cases are created on a purely intuitive basis. They are chosen because of their simplicity, reasonableness, and understandability. Although none of them is applicable to the general case, they are believed to be good approximations to many real-life situations.

2.4 Benchmark Cases

In this section three benchmark cases are presented. In these cases, the resolution of the display is assumed to be 1024x1024.

2.4.1 Uniform Distribution

The first probability density function (p.d.f.) being considered is very simple. Here δ is assumed to be uniform. That means:

$\delta(1) = \delta(2) = \delta(3) = \ldots\ldots\ldots = \delta(1024)$ and

$\delta(1)+\delta(2)+\delta(3)+\ldots\ldots\ldots+\delta(1024) = 1.$

Hence $\delta(h) = 1/1024.$

$$P\{h,w\} = \delta(h)\delta(w)$$
$$= 1/(1024*1024) = 1/1048576.$$

Notice that here $P\{h,w\}$ is independent of h and w. For example, updating a 10-by-10 grid is as probable as updating any 100-by-100 grid, or any 19-by-157 grid. When there is a lack of

prior knowledge about the height and the width of patterns that a system will deal with, this uniform p.d.f. for P is very appropriate. Consequently, from (2.1) and (2.2) the expected performance for the scan-line mapping scheme and the symmetric mapping scheme are

$$E_{scan} = 1/1048576 \sum_{h=1}^{1024} \sum_{w=1}^{1024} h * \lceil w/n \rceil \text{ and}$$

$$E_{sym} = 1/1048576 \sum_{h=1}^{1024} \sum_{w=1}^{1024} \lceil h/k \rceil * \lceil w/k \rceil \text{ respectively.}$$

Table 2.1 shows the E_{scan}'s and E_{sym}'s evaluated for different k's (see (2.1) and (2.2)). Note that for consecutive rows the total number of chips used is quadrupled.

$k=\sqrt{n}$	E_{scan}	E_{sym}	$(E_{sym}/E_{scan})*100\%$
2	65856.2500	65792.2500	99.9028
4	16656.2500	16512.2500	99.1355
8	4356.2500	4160.2500	95.5007
16	1281.2500	1056.2500	82.4390
32	512.5000	272.2500	53.1220

Table 2.1. Expected performance for both mapping schemes (case 1).

From this table, it can be observed that an increase in the number of chips used for the frame buffer, n, reduces the expected number of frame buffer accesses, E. This can easily be explained by the fact that when more chips are being used, more pixels can be read in parallel, hence fewer accesses are required to update a grid of pixels. Ideally, if n is doubled,

any updates to the frame buffer which required z frame buffer accesses should now require only $z/2$ accesses. In order to investigate the relation between the increase in the number of chips and the enhancement in performance, the *improvement function*, Θ is introduced.

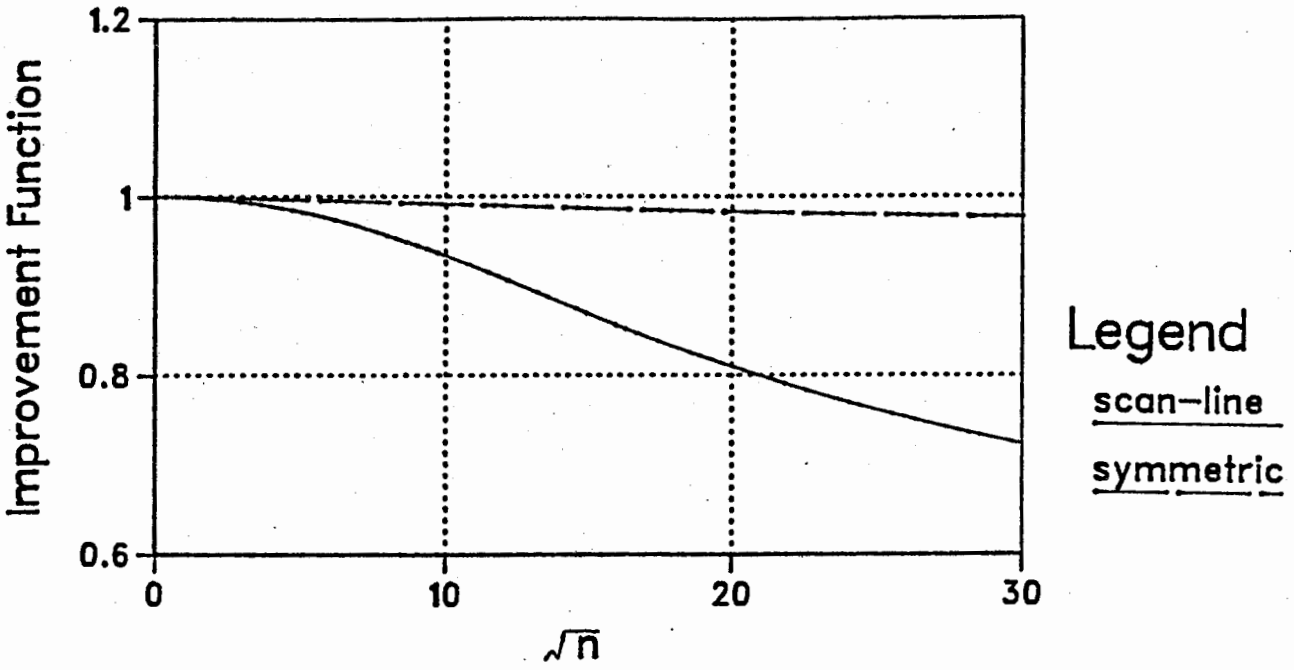
When n is increased from n_1 to n_2, the corresponding E decreases from E_1 to E_2. The improvement function at $n=n_2$ is expressed as:

$$\Theta(n_1,n_2) = \frac{E_1/E_2}{n_2/n_1} .$$

We will assume that $n_2=4n_1$ (see Table 2.1). The Θ's for the scan-line and the symmetric mapping schemes, with uniformly distributed P, are plotted against $\sqrt{n_2}$ in Graph 2.1. Notice that k is equal to $\sqrt{n}$. As shown in the graph, the symmetric mapping scheme is quite close to the ideal case.

2.4.2 Non-Uniform Linear Distribution

In the previous case, $G\{h,w\}$ was assumed to be independent of h and w, and thus $P\{h,w\}$ is the same for all h and w. This implies the events $\mathbf{U}\ G\{x,y,h,w\}$ for various h's and w's are equally likely. Since there are fewer $G\{x,y,h,w\}$ for large h and w, the probability of $G\{x,y,h,w\}$ will be larger if h and w are larger. However, in many cases, a larger grid of pixels is not updated as often as a smaller one. Intuitively, a screen can hold more small patterns than larger ones. This encourages the



Graph 2.1 Plot of improvement functions vs $\sqrt{n_2}$ (from Table 2.1).

use of little objects to construct an image. In addition, if the scene is modelled with smaller rectangles (see Figure 2.6), more details are preserved and hence a better approximation can be obtained. In this section, event $G\{x,y,h,w\}$ for any x, y, h, and w are assumed to be equally probable. That is, $P\{x,y,h,w\}$ are the same for any combination of x, y, h, and w.

For a particular h and w, there are $(1024-h+1)*(1024-w+1)$ distinct h-by-w grids. Therefore $P\{h,w\}$ will be large if h and w are small. The total number of distinct events is $\sum_{i=1}^{1024}\sum_{j=1}^{1024}(1024-i+1)*(1024-j+1)$. Hence, the probability of each event is $1/(\sum_{i=1}^{1024}\sum_{j=1}^{1024}(1024-i+1)*(1024-j+1))$ and thus $P\{h,w\}$ can be

expressed as :

$$P\{h,w\} = \frac{(1024-h+1)*(1024-w+1)}{\sum_{i=1}^{1024}\sum_{j=1}^{1024}(1024-i+1)*(1024-j+1)} = \frac{(1025-h)*(1025-w)}{\sum_{i=1}^{1024}\sum_{j=1}^{1024}(1025-i)*(1025-j)}. \qquad (2.3)$$

With the $P\{h,w\}$ given in (2.3), the expected number of accesses are evaluated (see (2.1) and (2.2)) and shown in Table 2.2. The improvement function obtained from this table is plotted against $\sqrt{n_2}$ in Graph 2.2.

$k=\sqrt{n}$	E_{scan}	E_{sym}	$(E_{sym}/E_{scan})*100\%$
2	29369.4585	29326.6460	99.8542
4	7471.4488	7374.6204	98.7040
8	1999.4488	1865.2708	93.2892
16	641.4585	477.2626	74.4027
32	342.0000	124.9270	36.5284

Table 2.2 Expected performance for both mapping schemes (case 2).

Similar to the uniform p.d.f. case, the Θ for symmetric mapping scheme is shown to be closer to the ideal . However the Θ's for both mapping schemes drop even more rapidly. This suggests that Θ is somewhat correlated to $P\{h,w\}$.

2.4.3 Chi-Square Distribution

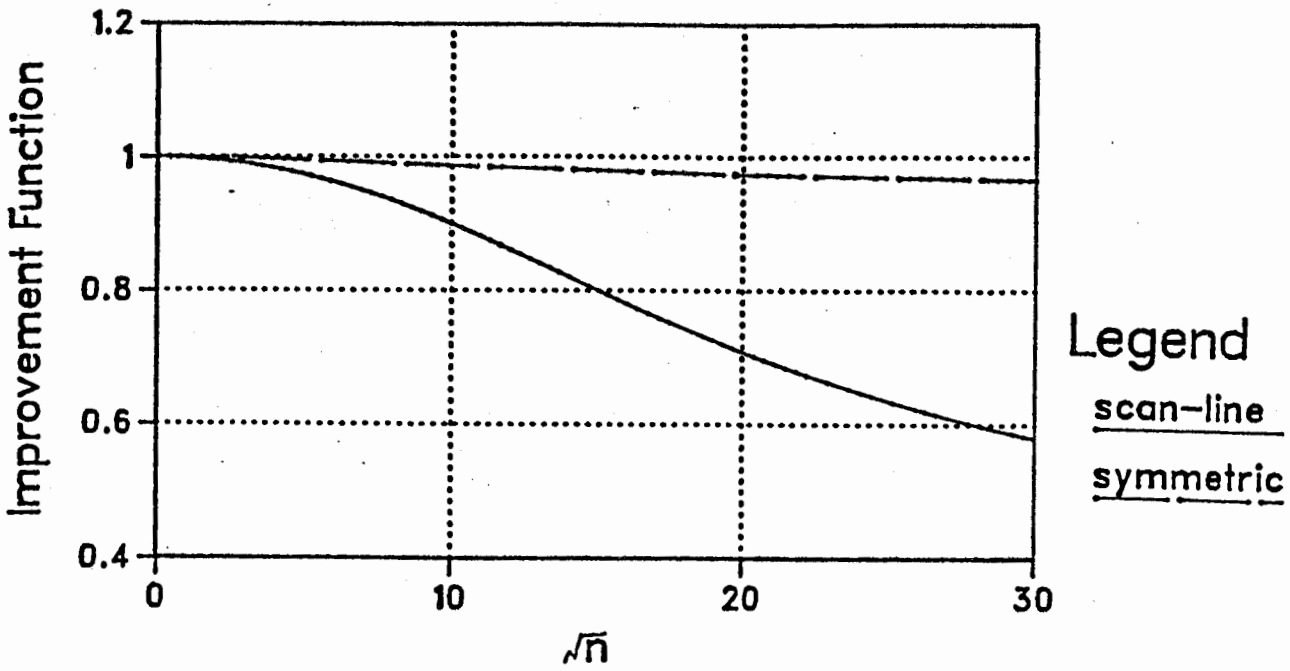
Graphics systems which approximate three dimensional surfaces with polygons or fractal faces use many polygons or triangles to compose an image. Moreover, these polygons or triangles are of moderate size. Relatively bigger or smaller geometric objects are less populated, and patterns with extreme sizes are almost non-existent.

The p.d.f. of h and w for this kind of systems can best be described by the following characteristics:

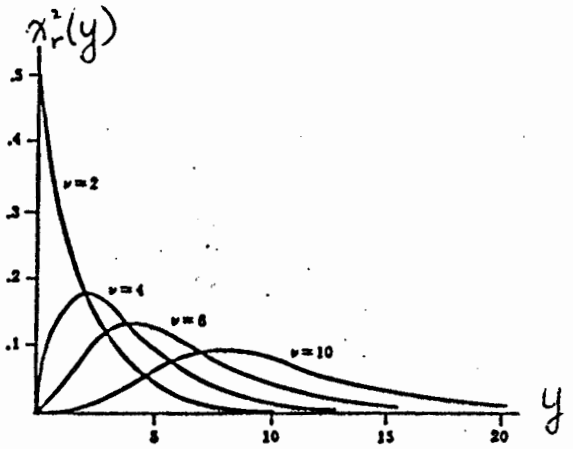
1. There exists an *optimal* x_0 such that $\delta(x_0)>\delta(x)$ for all x such that $1\leq x\leq 1024$ and $x\neq x_0$.
2. The farther x is away from x_0, the more is $\delta(x)$ less than $\delta(x_0)$.
3. $\delta(x)$ approaches 0 as x approaches 1 or 1024.

This statistical structure can appropriately be approximated by the well known *chi-square* p.d.f. The shape of a particular chi-square p.d.f. depends on its degree of freedom, denoted by ν. Graph 2.3 shows the shape of different chi-square distributions with various ν's [SPIE61].

Let S_x and S_y be the width and the height of the screen respectively. For a square screen, such as the one used in our model, S_x and S_y are identical and denoted by S. Assume the optimal x_0 of δ is about S/10. Also assume it is about 75% confident that x falls in the range between S/20 and S/2, that is, $\int \delta(x)dx\simeq 0.75$. The chi-square p.d.f. of 3 degrees of



Graph 2.2 Plot of improvement functions vs $\sqrt{n_2}$ (from Table 2.2).



Graph 2.3 Plots of chi-square p.d.f's of various ν's.

freedom, denoted by $\chi_3^2(y)$, has the global maximum at $y=1$, and its integral from 0.5 to 5 is approximately equal to 0.75. Hence it is very suitable for approximating δ. However, there are two problems prohibiting direct substitution of δ by χ_3^2:

(a) The 75% confidence interval is intended to be (51,512). However, χ_3^2's 75% confidence interval is (0.5,5).

(b) The function δ is discrete, whereas χ_3^2 is continuous.

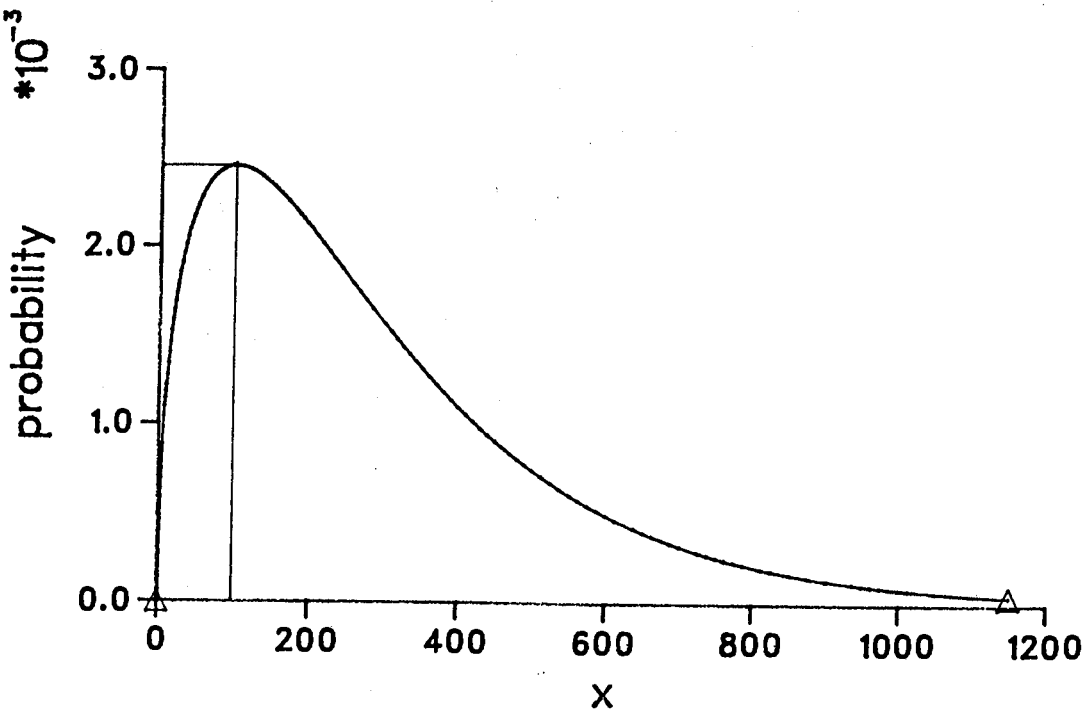
Fortunately, these two problems can be solved easily. Firstly, in order to use $\chi_3^2(y)$ on the domain (1,...,1024), y is substituted by $x/100=0.01x$. Therefore the global maximum of $\chi_3^2(0.01x)$ occurs at $x=100$, and (50,500) is an $\simeq$75% confident interval. The $\chi_3^2(0.01x)$ is

$$\frac{1}{2^{3/2}\Gamma(3/2)}(0.01x)^{1/2}\mathrm{EXP}(-0.01x/2)\ ,$$

where $\Gamma(\tau) = (\tau-1)\Gamma(\tau-1)$ and $\Gamma(1/2)=\sqrt{\pi}$.

The shape of this p.d.f. is displayed in Graph 2.4.

Secondly, $\delta(a)$ is approximated by $\int_\beta^\gamma \chi_3^2(0.01x)dx$, where $\beta=(a-0.5)/100$ and $\gamma=(a+0.5)/100$. Quantitatively, this integral is equal to the area under the curve $\chi_3^2(0.01x)$ between $x=a-0.5$ and $x=a+0.5$. This integral is further approximated by the area $1*\chi_3^2(0.01(a-0.5+a+0.5)/2) = \chi_3^2(0.01a)$. Note that this is the area of the bar with unit width and height equal to $\chi_3^2(0.01a)$. Hence $\delta(x)$ can be represented by $\chi_3^2(0.01x)$. To ensure $\sum_{1\le h,w\le 1024}^{\forall h,w} \chi_3^2(0.01h)\chi_3^2(0.01w)$ is equal to 1, an adjustment factor $J=1/(\sum_{1\le h,w\le 1024}^{\forall h,w} \chi_3^2(0.01h)\chi_3^2(0.01w))$ is multiplied to $\chi_3^2(0.01h)\chi_3^2(0.01w)$.



Graph 2.4 A plot of the chi-square distribution on $x/100$ with $\nu=3$.

Therefore, $P\{h,w\}$ can be formulated as:

$$J*\chi_3^2(0.01h)*\chi_3^2(0.01w)$$

$$= \frac{J}{2^3(\Gamma(3/2))^2}\,(0.01h)^{1/2}(0.01w)^{1/2}\ \mathrm{EXP}(-0.01(h+w)/2)\ .$$

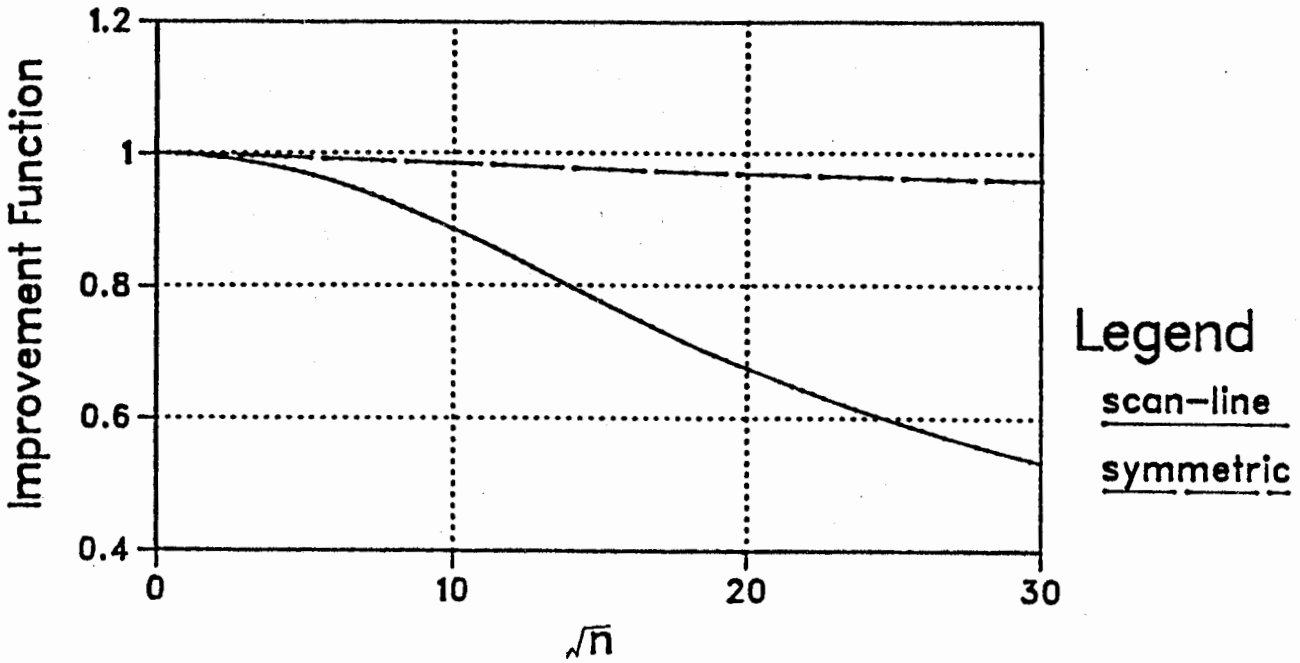
The evaluated E's (see (2.1) and (2.2)) for various values of n are tabulated in Table 2.3. The behavior of Θ for both mapping schemes are shown in the plot (Graph 2.5) of Θ against $\sqrt{n_2}$.

$k=\sqrt{n}$	E_{scan}	E_{sym}	$(E_{sym}/E_{scan})*100\%$
2	20356.2089	20320.6894	99.8255
4	5195.8748	5115.8765	98.4604
8	1406.6181	1296.9197	92.2013
16	467.2054	333.3064	71.3404
32	286.5982	87.9703	30.9104

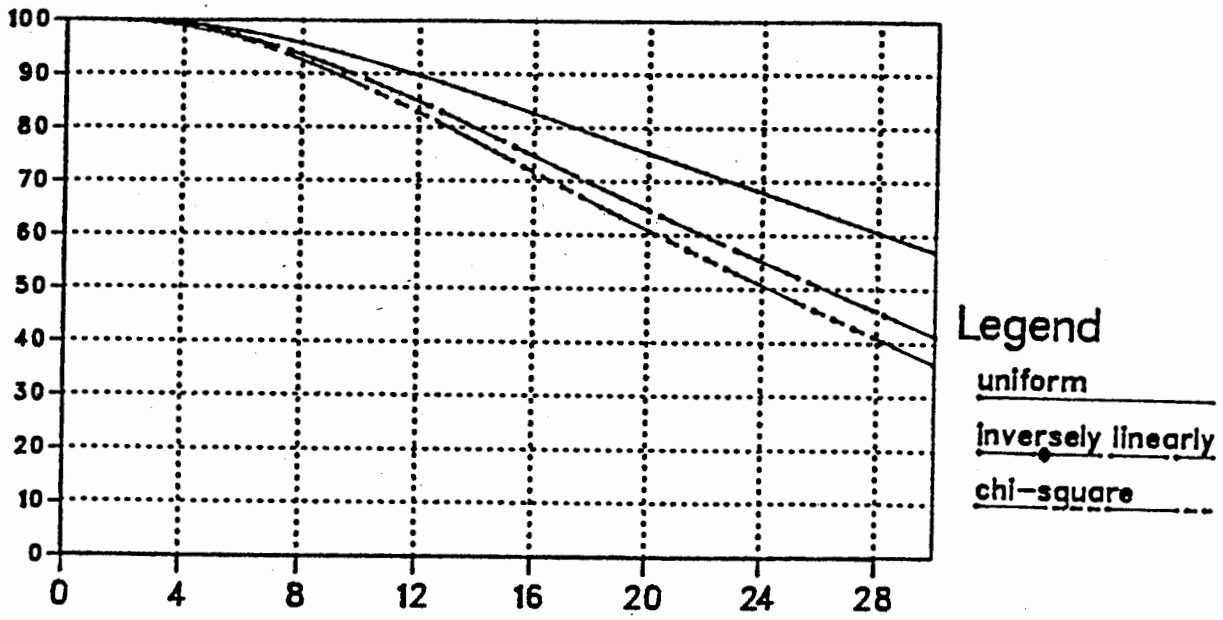
Table 2.3 Expected performance for both mapping schemes (case 3).

2.5 Choosing a Mapping Scheme

In the previous section we noticed that the expected number of accesses for the symmetric mapping scheme is always less than that of the scan-line mapping scheme. As illustrated in Graph 2.6, E_{sym}/E_{scan} always gives a value less than 1. In addition to this, as n increases, symmetric mapping tends to be more and more effective than the scan-line mapping scheme. This is caused by the early degradation of the improvement function for scan-line mapping. Consequently, when more chips are required to implement the frame buffer, the design tends to favor symmetric mapping. The overhead for implementing symmetric mapping, which includes hardware for output buffering, address transformation, and two dimensional data alignment, becomes less significant in the sense of performance per unit cost.



Graph 2.5 Plot of improvement functions vs $\sqrt{n_2}$ (from Table 2.3).



Graph 2.6 Plot of $(E_{sym}/E_{scan})*100\%$ vs $k=\sqrt{n}$ for the 3 cases.

However, since the performance of the symmetric scheme is not much superior to the scan-line mapping scheme when n is small, it is natural to question whether the extra cost spent in building an image memory system using the symmetric mapping scheme is worthwhile. As illustrated in Table 2.1, the scan-line mapping scheme is compatible with the symmetric mapping scheme when k is ≤ 4 (i.e., $n \leq 16$). Consequently, if 16 or less chips are used, the scan-line mapping scheme is superior because of its simpler implementation and compatible performance.

There are two factors which favor the use of 16 or less chips for the frame buffer. With today's memory technology, 64K,

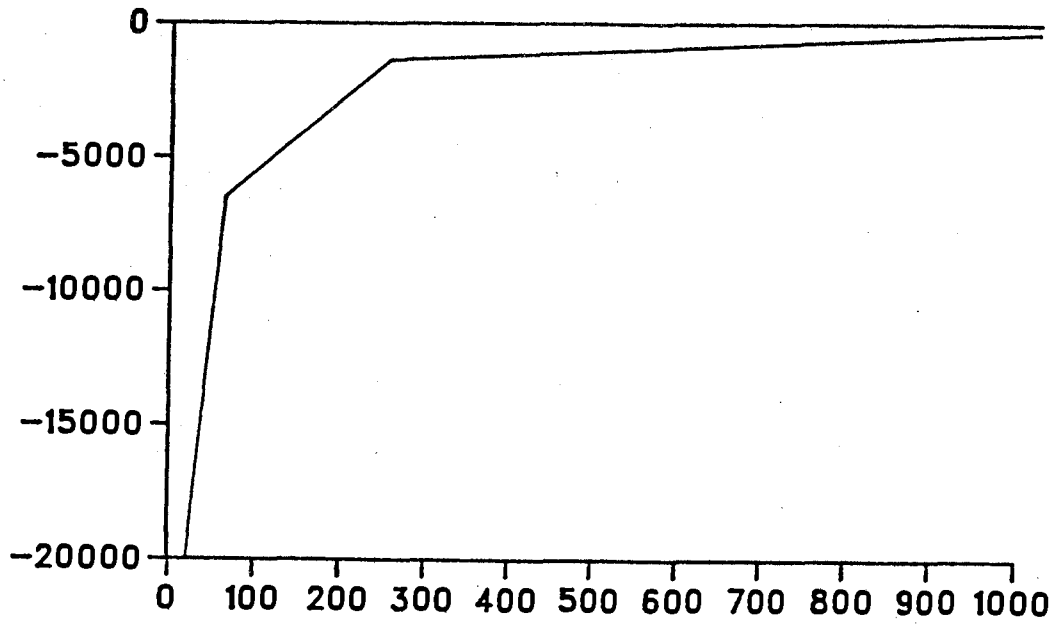
128K and even 256K memory chips (4164, 41128, and 41256) are very popular in the market. With these high density memory chips, a frame buffer can easily be built with less than 16 chips. These high density memory chips ease the implementation of frame buffer and supporting circuit+. In addition, the typical memory cycle time of these chips is ≤150 ns (especially for those having fast access modes such as page mode, extended page mode, or ripple mode [FALL84, FINK83, WHIT84]). With this speed an effective pixel access rate of ≥1 pixel for every 10(37 ns) can easily be achieved if 16(4) chips are used. Consequently, it is feasible to implement an 1024x1024 (512x512) display with 16(4) RAM chips.

Another factor which prevents the designer from using too many chips is the consideration of the actual achievable reduction in the number of frame buffer accesses per chip added. The best way to look at the problem is to study $\Delta E/\Delta n$. Assume when n is increased from n_1 to n_2, E decreases from E_1 to E_2. Then $\Delta E/\Delta n$ is expressed as:

$$\frac{E_2-E_1}{n_2-n_1} .$$

The $\Delta E/\Delta n$ for E_{sym}'s from Table 2.2 is plotted against n_2 in Graph 2.7. The $\Delta E/\Delta n$ for other cases are very similar to the one shown in Graph 2.7.

This graph shows that $\Delta E/\Delta n$ converges to zero quickly as n increases. Thus, a significant improvement can only occur at



Graph 2.7 Plot of $\Delta E/\Delta n$ vs n_2 for E_{sym} (from Table 2.2).

small n. This suggests that using too many chips to improve the overall system performance is not cost effective. Upgrading the system by using faster memories, dual-port RAMs, or display RAMs, would be more appropriate.

Differences in memory-to-screen mapping strategies may result in deviations on the effectiveness of scan conversion algorithms. The underlying memory organization of the k-by-k symmetric mapping scheme provides a model of computation where updates to the frame buffer can take place on any k-by-k square of the display. This two dimensional model of computation should be able to exploit the two dimensional nature of almost all graphics applications, and hence provides efficient algorithms

for them. However, faster scan conversion algorithms cannot improve the frame buffer updating efficiency. Therefore, the effectiveness of scan conversion algorithms is not a major deciding factor in choosing a mapping scheme unless more than enough frame buffer bandwidth is available. This becomes less probable as scenes become more complex. However, variations in scan conversion algorithms may favor different implementations (for example, different microcode for the display processor), and thus could potentially affect the overall cost of the implementation.

CHAPTER 3

PARALLEL-PROCESSING ARCHITECTURES FOR GRAPHICS DISPLAYS

In a conventional single processor frame buffer raster-scan graphics system, the frame buffer is a very important component. Its versatility provides an extremely flexible medium for image manipulation. In fact, many image processing systems rely on the frame buffer in which many image processing activities become feasible. However, the frame buffer does not offer the most efficient way of representing an image. The large amount of memory it uses makes it expensive. Its limited effective bandwidth creates a bottleneck for image updating. In most cases, the performance of a raster scan graphics system is bounded by the effective bandwidth of its frame buffer. Although many attempts have been made to increase the memory bandwidth [FINK83, MATI84, OSTA84, SPRO83], conventional frame buffer architecture is still too slow for high quality real-time graphics.

Real-time raster image generation requires a new frame to be generated for every screen refresh in order to give the illusion of motion and dynamic behavior. The problem with the basic frame buffer system is that generating a typical frame containing a few thousand polygons will take several seconds which is much longer than one frame refresh period, even if the symmetric mapping scheme is employed (see Chapter 2). Given present circuit speeds, massively parallel processing and pipelining

techniques are mandatory so as to generate a new frame during every refresh cycle. Researchers have proposed several high performance system architectures for achieving the high computation rate [CLAR80, DEME85, FUCH79, FUSS82, GHAR85, LOCA79, WEIN81, WHEL82]. These systems demonstrate promising results in handling today's sophisticated graphics applications.

3.1 Architectural Structures

Two major architectural structures have affected the design of most multi-processor graphics systems. These architectural types are :

(a) partitioned object-space structure, and

(b) partitioned image-space structure.

These two structures are closely related to two important domains of graphics representations, namely the *object space* and the *image space*. They are defined as follows:

Object Space :

The object space of an image is the set of graphics primitives, such as lines, polygons, circles, ... etc., which compose the scene to be depicted. (In this thesis, "graphics objects" is always used in place of "graphics primitives".) In many applications, the graphics objects are constrained to be of a single type, such as triangles, and the number of objects is bounded above by a constant (n). If this is the case, the object space may be considered as a set of n triangles.

Image Space:

The image space of a scene is the set of raster elements (pixels) of the display. It contains a fixed number (m) of elements which equals to the resolution of the display. Notice that a graphics object in the object space is defined by a high level parametric description which is usually a mathematical expression of its shape, shade, and orientation. The graphics objects must be rendered into their corresponding raster images in the image space, and then drawn on a CRT.

To render a scene, all graphics objects in the object space must be processed; and for each object, all elements of the image space must be considered (if no coherence property is assumed). "> Recall that a pixel evaluation process is called a *pel-process*. To render a graphics object, it may be required to execute more than one million pel-processes for a 1024x1024 display. If a scene contains thousands of graphics objects, the rendering activity may require the execution of a tremendous number of pel-processes per each frame of image generated. In addition, a pel-process can be broken down into up to three sub-tasks. The first sub-task is a containment test (*con-test*) which determines whether the pixel in question is inside the graphics object or not. The second sub-task is the computation of the shade and depth of the pixel according to the description of the object (*pel-evaluation*). The third one is a visibility test (*vis-test*). Although the first two sub-tasks can be carried

out simultaneously, the third one requires the depth value of the pixel and thus must be performed after the completion of the pel-evaluation task.

3.2 Partitioned Object-Space Architecture

In a partitioned object space architecture, the graphics objects in the object space are divided into a number of *groups* such that the union of these groups forms the object space itself. In addition, these groups are not necessarily mutually exclusive. Objects from the same group are handled concurrently. In most cases, an individual processor is assigned to each of the objects. Different groups of objects share the same processors and hence must be processed sequentially. This definition of "group" will be used throughout the rest of the thesis.

Examples of partitioned object space systems include Fussel's [FUSS82], Locanthi's [LOCA79] and Weinberg's[WEIN81] systems. In their systems, the entire object space is treated as a single group. Each object in the object space is assigned to an individual object processor and rendered simultaneously. A sequence of pixel values is generated by each object processor to synthesize the image of its associated object. The pixel generation activities of the *object processors* are synchronized such that they are all working on the same raster pixel at any instance. When a particular raster pixel is being considered,

each object processor will perform a con-test and pel-evaluation to determine the depth and the shade value of the pixel according to the object it is associated with. The values from all the processors are then passed down to a comparator which conducts the vis-tests.

3.3 Partitioned Image-Space Architecture

In this type of architecture, the image space (raster) is divided into a number of *partitions* of pixels. Pixels from the same partition are independent of each other and hence can be processed simultaneously. Nevertheless, manipulations of different partitions are carried out sequentially. Except in a few image processing applications, all of the pixels in the image space are mutually independent of each other. Therefore in most cases, there is only one partition which contains all the pixels in the raster considered. This architecture can be exemplified by the Pixel-Planes system [FUCH81, FUCH82, POUL85]. Whelan's rectangle area filling system is also of this type [WHEL82].

A Pixel-Plane is a special smart-memory chip for raster-scan graphics. The frame buffer is built with one or more Pixel-Planes. In the Pixel-Planes system, the entire image space is grouped into a single partition. Since all pixels of the partition are to be processed simultaneously, there is an individual processor associated with each image memory cell

which holds a pixel. In this way all the pixels in the raster (image space) can be processed in parallel. The objects in the object space are rendered sequentially. According to the high-level descriptions of the geometric objects, each pixel-processor executes a pel-process to determine the value of its corresponding pixel.

Since graphics objects are processed sequentially, the total time required to synthesize a scene is proportional to the total number of objects in the scene. Therefore, it will take a relatively long time to process a scene with many objects.

3.4 A Generalization of the Two Architectures

Almost all raster graphics displays employ either one or both partitioning structures. In this section, a general architectural representation for image rasterizers is presented. The representation is very useful in characterizing an architecture with its strategy to distribute processing tasks among the processors. To discuss this representation, two definitions are needed:

Object-Space Partitioning Scheme (OP-Scheme):

In a parallel processing architecture, the object space can be divided into a certain number of groups such that the objects in a subset can be processed in parallel. The object-space partitioning scheme defines the way graphics objects are grouped. An *α-β-OP scheme* is an object-space

partitioning scheme which groups the graphics objects in the object space into α groups and the maximum cardinality of the α groups is β.

Image-Space Partitioning Scheme (IP-Scheme):

The image-space partitioning scheme determines the way that the image of a display is partitioned. A γ-λ-*IP scheme* is an image-space partitioning scheme which divides the image space into γ partitions. The maximum cardinality of the partitions is λ. To rasterize a graphics object, the system will consider the γ partitions, each having at most λ pixels. Notice that the IP-scheme defines the way that a graphics object is rasterized. Therefore, it is theoretically possible to have different IP-schemes for different graphics objects. However, this is not a common practice in raster graphics displays and thus is not considered in this thesis.

From now on, let us assume that the maximum number of graphics objects in the object space is n and the number of raster pixels in the image space is m. In general, any image rasterizer can be identified by the way that its object space and image space are partitioned. For example, the Pixel-Planes system [POUL85] is constructed based on an n-1-OP scheme and an 1-m-IP scheme. In this system, the object space which contains at most n graphics objects is partitioned into n subsets, each of which contains at most 1 object. However, the raster image of each object, which is made up of m pixels, is formed by a single

image space partition with exactly m pixels. Assume the total time required to execute a pel-process is T_{pel} units, then the Pixel-Planes system is able to obtain the entire raster image of an object in T_{pel} units of time. However, the objects in the object space must be processed sequentially.

At the other extreme, the Fussel's real-time scan conversion engine [FUSS82] utilizes a 1-n-OP scheme and an m-1-IP scheme. That means it is capable of handling all the graphics objects simultaneously. However, only one pixel can be computed per T_{pel} units of time.

We call a system which employs an α-β-OP scheme and a γ-λ-IP scheme to be an α-β-γ-λ-*system*. The hardware complexity and the computational power of an α-β-γ-λ-system will be discussed in the following sections.

3.5 Hardware Complexity of an α-β-γ-λ-System

The amount of hardware required to implement an α-β-γ-λ-system can be determined by the way that the object-space and the image-space are partitioned. For a system which uses an α-β-OP scheme, its object-space is divided into α groups of graphics objects. The maximum number of objects in the groups is β. Therefore, in order to process all the objects of a group simultaneously, at least β processing units are required.

For each object processor, a γ-λ-IP scheme is implemented. The image of an object is composed by γ partitions of λ pixels. All the pixels from the same partition are determined simultaneously. This costs the system λ individual pixel processors to be associated with each object processor. The total number of pixel-processors required to implement a rasterizer employing an α-β-OP scheme and a γ-λ-IP scheme is $\beta\lambda$. In the rest of the thesis, the value $\beta\lambda$ is regarded as an indicator for the *hardware complexity* of an α-β-γ-λ-system.

3.6 Processing-time Complexity of an α-β-γ-λ-System

In the process of rasterizing a scene each graphics object must be redefined with its corresponding raster image in the image space. This process involves the translation of the high-level description of the object into a set of relevant pixel values. The image size of an object can be as large as the entire raster. Therefore, in the worst case, every pixel in the raster must be processed in order to obtain the corresponding raster image of a graphics object. For an α-β-γ-λ-system, the task of finding the raster image of a single object will take γT_{pel} units of time since the raster is divided into γ partitions which are processed sequentially. Furthermore, objects in the scene are grouped into α groups and the processing of these groups are carried out sequentially. The total time needed to handle all objects will be proportional to α. Consequently, the total time required to render a scene into

its corresponding raster image, defined in the image space, will be $\alpha\gamma T_{pel}$ units. That is, the throughput of the system will be n objects per $\alpha\gamma T_{pel}$ units of time. In the rest of the thesis, the value $\alpha\gamma$ is defined as the *processing-time complexity* of the rasterizer which can be used to indicate the computational power of an α-β-γ-λ-system. Notice that a small $\alpha\gamma$ indicates that the system can rasterize a scene of upto $\alpha\beta$ objects in a short period.

3.7 Classification of Image Rasterizer Architectures

In Table 3.1, various recently proposed architectures for image rasterizers are classified according to the α-β-γ-λ notation. In a conventional single processor sequential system, the graphics objects are rasterized sequentially, and the raster pixels are generated sequentially as well. That is, the system is employing an n-1-OP scheme and an m-1-IP scheme. Therefore, its hardware complexity is 1 and its processing-time complexity is nm. However, in most cases, some kind of graphics coherence properties are utilized in these systems. As a result, the actual number of pixels required to be considered per object is less than m. Consequently, the processing-time complexity of the system is probably less than nm.

Clark's system has 64 (as described in [CLAR80b]) IMP processors running in parallel. Each processor is responsible for a set of $m/64$ pixels. Thus, it is classified as an

Table 3.1 Classification of image rasterizers with α-β-γ-λ notation.

	α-β-γ-λ notation	Hardware Complexity $\beta\lambda$	Processing-time Complexity $\alpha\gamma$
Sequential System	n-1-m-1	1	nm
Clark's [CLAR80b]	n-1-(m/64)-64	64	nm/64
Demetrescu's [DEME85]	n-1-64-(m/64)	m/64	64n
Fuchs' [FUCH80]	n-1-1-m	m	n
Fussel's [FUSS82]	1-n-m-1	n	m
Gharachorloo's [GHAR85]	$\sqrt{m}$-$\sqrt{m}$-1-$\sqrt{m}$	$\sqrt{m}$	m
Lachanthi's [LOCA79]	1-n-m-1	n	m
Weinberg's [WEIN81]	1-n-m-1	n	m
Whelan's [WHEL82]	n-1-1-m	m	n

n-1-(*m*/64)-64-system. Although the processing-time complexity of this system is *nm*/64, it may require less time to render a scene; since, again, graphics coherence can be employed in this system.

Fuchs' [FUCH81] and Whelan's [WHEL82] system are two typical *n*-1-1-*m*-systems. Fussel's[FUSS82], Lochanthi's [LOCA79], and Weinberg's [WEIN81] systems are typical 1-*n*-*m*-1-systems. Their characteristics have been described in the previous sections.

Gharachorloo's system employs a systolic array of 512 processor cells (in [GHAR85], a 512x512 display is considered). Each processor cell handles one column of raster pixels. Graphics objects in the object space form 512 groups each associated with a scan-line. An object belongs to a group if and only if its raster image intersects with the scan-line that the group is associated with. (In [GHAR85], Gharachorloo assumes that at most 512 objects are in a group.) The image rasterization process is completed by evaluating the 512 scan-lines sequentially. To generate a scan-line, the group of objects associating with that scan-line is "pumped" through the systolic array. As the objects propagate through the processor array, each processor cell will execute a pel-process for each object passing through. Since this activity is carried out in a pipelined fashion, the rasterization of up to 512 objects can be overlapped. This group-wise rasterization of objects makes a group of objects look like a single graphics object to the system. Therefore, we can consider the objects space of

Gharachorloo's system as 512 groups of objects each of which contains only a single (combined) object. The α-β-γ-λ notation for this system is thus $\sqrt{m}$-1-$\sqrt{m}$-$\sqrt{m}$ (in [GHAR85] $\sqrt{m}$ is 512).

Demetrescu's system employs a number of SLAM modules [DEME85]. A SLAM module is in fact a smart image memory module which contains a matrix of 64 rows by 256 columns of storage cells. There is also processing logic built inside the SLAM modules such that a row (256) of pixels can be generated in parallel. Therefore, we can consider that there are 256 pixel-processors in a SLAM. To build a rasterizer for a $\sqrt{m}$x$\sqrt{m}$ diplay, a $\sqrt{m}/64$ by $\sqrt{m}/256$ array of SLAM's must be used. In this case, $(\sqrt{m}/64)*(\sqrt{m}/256)*256=m/64$ pixels can be updated simultaneously. With this configuration, the image space can be regarded as being partitioned with a 64-$(m/64)$-IP scheme. Therefore, Demetrescu's system can be classified as an n-1-64-$(m/64)$-system.

CHAPTER 4

REGIONAL-RASTERIZATION

To make real-time image generation possible, many high performance raster scan graphics displays employ parallel-processing techniques. Owing to the nature of the scan conversion problem, the low-cost "loosely-coupled" processor network structure is very appropriate for graphics displays. Unfortunately, because of the poor inter-processor communication ability of loosely-coupled parallel processor, most of the inherent graphics coherence properties are lost [KAPL79]. As a consequence, many efficient scan conversion algorithms commonly used in sequential systems, which make use of various forms of graphics coherence properties, cannot be applied in parallel-processing graphics displays.

Regional-Rasterization is an architectural enhancement to parallel-processing raster-scan graphics displays. Conceptually the technique makes use of the "face" coherence property of graphics objects[1] complemented with the "even distribution of graphics objects" argument[2]. With Regional-Rasterization the entire image space is partitioned into a number of (equal) partitions. The generation of the entire scene image is accomplished by processing these image space partitions

---

[1] Face coherence : The faces are generally small compared to the size of the screen [SUTH74].

[2] Graphical objects are distributed fairly *uniformly* over the display and thus there are no "preferred positions" for objects [SPRO81].

sequentially, or simultaneously. For each image partition, only those graphics objects whose projection is contained entirely or partially by the partition are rendered.

4.1 Concepts Behind the Regional-Rasterization Scheme

">

Most n-1-1-m and 1-n-m-1 systems, including the Pixel-Planes and Fussel's systems, are not cost-effective in terms of processor utilization. Consider the scene displayed in Figure 1.1. The average size of the planar tiles (polygons) which are used to model the various surfaces of the 'beetle' is very small compared to the area of the entire screen. By observation, the mean size of the polygons is only about one-hundredth of the area of the entire display. Thus, the average number of pixels that are covered by a graphics object is $0.01m$. Therefore, theoretically, a graphics object can be rendered with an average of $0.01m$ pel-processes. As a result, the optimal number of pel-processes required to render the scene is $0.01nm$.

Based on the above observation, the total number of pel-processes required to render a scene, can be optimized by ignoring irrelevant pixels. The technique employed to achieve this goal is called *Regional-Rasterization*. By using the Regional-Rasterization technique, the cost-effectiveness of an image rasterizer can be improved by either: (1) boosting the

system throughput and hence overall performance with acceptably small hardware overhead; or (2) cut down a large amount of hardware, required to implement a system, with little penalty in system throughput. Notice that a substantial improvement in hardware utilization can be achieved through either one of these architectural enhancements. The term "Regional-Rasterization" denotes the idea of maximizing processor utilization by considering only relevant pixels as a graphics object is rasterized.

4.1.1 Utilization of Coherence Information

When rendering an object, if all raster pixels are considered, we say that the object is *rasterized exhaustively*. In contrast, if only relevant pixels are considered, we say that the object is *rasterized non-exhaustively*. The basic idea of Regional-Rasterization is, by using a coherence property, to avoid rasterizing graphics objects exhaustively. A similar approach has long been adopted in sequential systems. Although the idea is straightforward, no attempts similar to Regional-Rasterization have yet been applied to parallel processing systems. This may be due to the following reasons:

(a) To reduce the work involved in the scene rendering process, various forms of coherence, such as object, area, or scan-line, can be incorporated. In fact, the use of a priori knowledge in the form of image or object coherence has been crucial to the efficiency of most sequential systems. In the case of parallel processor implementations, much of this

coherence information will be lost when independent parallel calculation tasks are distributed across a network of processors unless a sufficiently large amount of data can be exchanged among processors [KAPL79]. Nevertheless, in order to reduce the high cost of interconnection and the problems associated with high data transfer among processors, processing units in most multi-processor graphics displays are "loosely-coupled". This type of configuration makes employment of coherence in rendering very difficult, or even impossible.

(b) Employment of coherence properties in scene rendering activity is advantageous only in optimistic cases. In worst case situations, the overhead introduced will degrade the performance of the system. Therefore, most designs avoid the employment of any coherence. They are usually designed to allow computational redundancy and provide "room" for worst cases. Although this may result in a less cost-effective design, it guarantees uniformity and reliability. This is particularly true in the case of multi-processor VLSI systems where the cost of individual processors is almost negligible.

Therefore, unless the approach is very simple, easy to implement, and requires very little overhead, using coherence to achieve better average performance is not of much value to these systems. In the following sections, we will examine the possibility of incorporating face-coherence property of graphics objects into the scene rasterization process.

4.1.2 Average Size of Graphics Objects

In the classic paper of Sutherland, Sproull and Schumacker [SUTH74], a set of statistical measures of the complexity of rendering were presented. These environment statistics are listed in Table 4.1 and Table 4.2. In the same paper, a useful relationship was also given :

$$D_e = F_r * ((H_f/a) * (H_f/b)) \qquad (4.1)$$

From the above expression, one can see that for constant D_e, F_r is inversely proportional to H_f. They also stated that most environments of any great degree of complexity appear to be nearly isotropic[3]. The only possible explanation for this characteristic is that graphics objects are distributed fairly uniformly in the environment. Consequently, it is reasonable to assume that $D_e << F_r$. That is, the average number of objects overlapping at a pixel is much smaller than the total number of objects in the environment. In fact, an estimate of 3 is used for D_e in [SUTH74]. Therefore, by expression (4.1), H_f/a and H_f/b are relatively small. Since H_f/a and H_f/b represent the average size of faces expressed as a fraction of the displayed height and width respectively, an average graphics object occupies only a small fraction of the display area when F_r is large and is much larger than D_e.

---

[3] An environment is said to be isotropic if the depth complexity is independent of the viewing direction. That is, the expected number of faces penetrated by any randomly chosen line is independent of the direction of the line.

Table 4.1 (from [SUTH74]).

ENVIRONMENT STATISTICS

F_t	Total number of faces in the environment.
F_r	Number of relevant faces in the environment.
D_c	Depth complexity of the environment (average).
C_t	Total number of clusters in the environment.
C_r	Number of relevant clusters in the environment.
F_c	Number of faces per cluster (average).
E_t	Total number of edges in the environment.
E_r	Number of relevant edges in the environment.
E_s	Number of relevant edges if sharing is allowed.
E_c	Number of contour edges in the environment.
X_r	Total number of edge crossings in the viewing plane.
X_v	Number of intersections of visible edges.
X_f	Number of face intersections.
H_f	Height of a face in resolution units (average).
S_r	Total number of segments, visible or not.
S_t	Number of segments on a scan line, visible or not (average).
S_v	Number of visible segments on a scan line (average).
L_v	Total length of visible edges (measured in resolution units).
n	Vertical resolution of screen (number of scan lines).
m	Horizontal resolution of screen.

Tabel 4.2 (from [SUTH74]).

STATISTICS FOR THREE ENVIRONMENTS

Statistic	*Rule of Thumb*	*Roberts' House* (1/25)	*Harbor* (1)	*Big Harbor* (25)
n	given	500	500	500
m	given	500	500	500
F_r	given	100	2500	60000
F_c	given	10	25	200
D_c	given	3	3	3
F_t	$2F_r$	200	5000	120000
C_t	F_t/F_c	20	200	600
E_t	$4F_t$	800	20000	480000
E_r	$E_t/2$	400	10000	240000
E_c	$E_r/(F_c/2)^{1/2}$	180	2800	24000
E_s	$(E_r-E_c)/2+E_c$	290	6400	130000
X_r	$(D_c-1)E_r/4$	200	5000	120000
X_v	X_r/D_c	70	1700	40000
H_f	$(nmD_c/F_r)^{1/2}$	86	17	4
S_t	$(D_cF_rm/n)^{1/2}$	17	87	420
S_v	S_t/D_c	5	29	140
L_v	$2nS_v$	5000	29000	140000

From the above discussion, a graphics object in a complex scene should occupy only a small fraction of the entire display area on the average. Therefore, a substantial amount of computation effort can be saved by a non-exhaustive rasterization algorithm. Consequently, Regional-Rasterization technique will be beneficial in most cases.

4.2 A Functional Model

Following the arguments given in the previous section, it is clear that it is unnecessary to consider all the raster pixels while rendering a graphics object. If one can divide the object space and the image space into object sub-spaces $\{O_1, O_2, \ldots, O_p\}$ and image sub-spaces $\{I_1, I_2, \ldots, I_p\}$ respectively, such that objects in O_i cover only the pixels in I_i $(1 \geq i \geq p)$, then we need to consider only the pixels of I_i for rendering the objects of O_i. In this case, objects in O_i are said to be *local* to the partition I_i; or the objects are local objects of I_i[4].

Consider an IP-scheme which divides the raster into several mutually exclusive equal partitions. By the evenly distributed argument (mentioned in the previous section), there will be roughly the same number of objects occupying each partition. When a graphics object is rendered, the only pixels to be considered are those in the partition(s) that the object is

---

[4] Note that an object can be local to more than one partition.

local to. If the partitions are large enough, most graphics objects will be enclosed entirely by a single partition (i.e., very few objects will span over more than one partition). Therefore, the average number of pel-processes required to execute in order to rasterize an object will be approximately equal to the number of pixels in each partition. This behavior is depicted in Figure 4.1 which shows that the scene from Figure 1.1 when divided into four quarters. The unshaded polygons are enclosed entirely in a quarter. As demonstrated, the unshaded polygons represent a majority of the entire object space. In addition, most of the shaded polygons are local to only two partitions. For these polygons, only pixels in two adjacent partitions should be considered. Therefore, the number of relevant pixels is only 1/2 of the total number of raster pixels in the image space.

The way to form the object groups is determined based on the IP-scheme. Firstly, the object space is divided into p groups where p equals the number of raster partitions as described by the IP-scheme. Secondly, a one-to-one correspondence is constructed between the object space groups and the image space partitions such that an object space group contains those and only those graphics objects which are local to the partition. This process is called the *local object identification* process. For example, the four object space groups as defined by the IP-scheme in Figure 4.1 will look like those shown in Figure 4.2. Notice that the total number of objects in the four

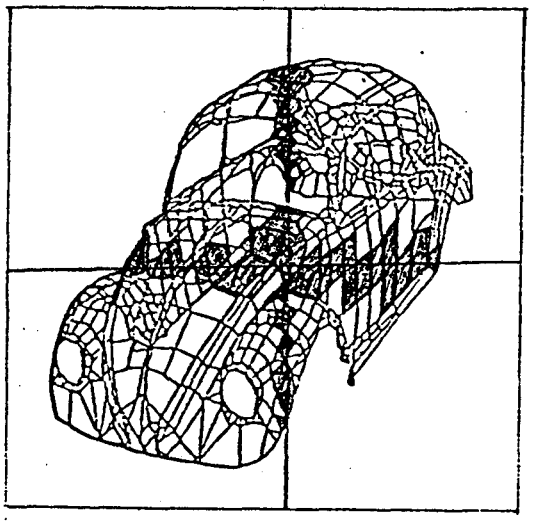


Figure 4.1 A sample IP-scheme employed by Regional-Rasterization.

groups may be larger than the actual number of objects in the object space.

4.2.1 Effectiveness of Regional-Rasterization

As mentioned, I_i is the sole image partition associated with O_i. We can regard the rasterization of objects in O_i into image sub-space I_i as an independent image rasterization problem P_i.

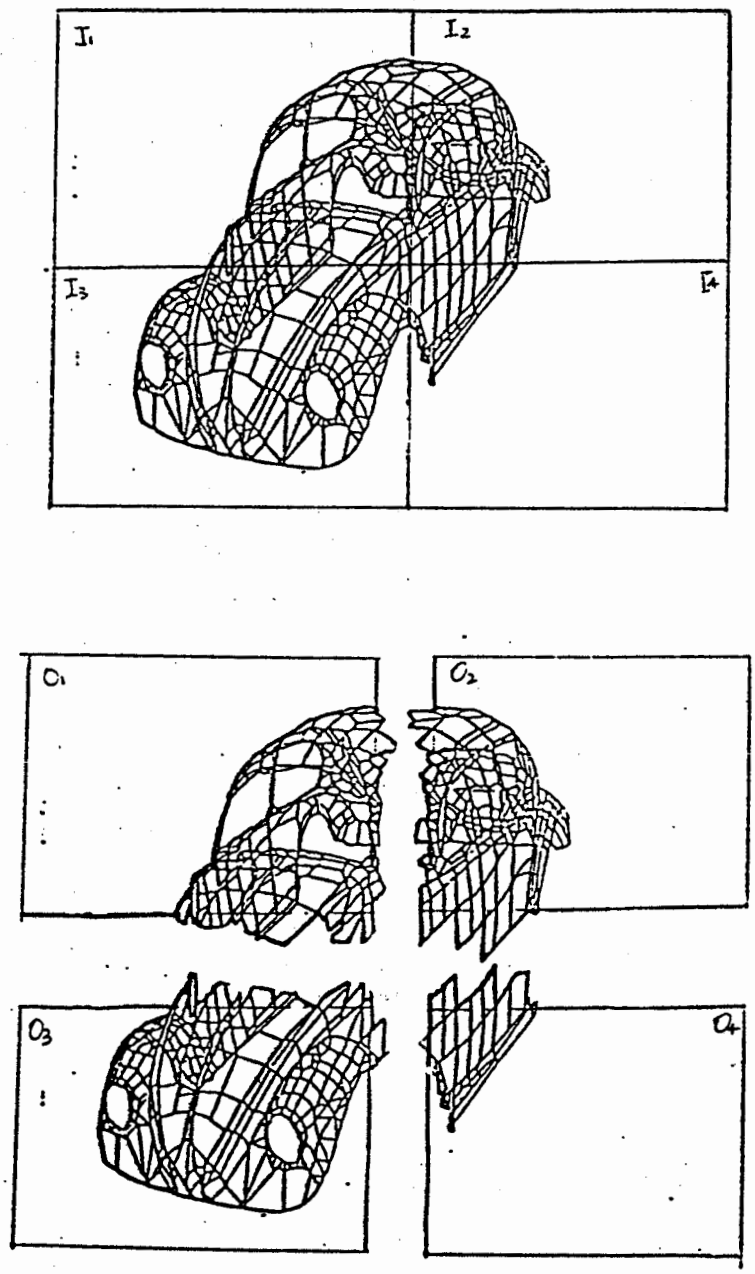


Figure 4.2 The Corresponding OP-Scheme for the sample IP-scheme from Figure 4.1.

Therefore, the original scene rasterization problem, which can be considered as the *parent* problem, can be broken down into p independent sub-problems ($P_1, P_2, \ldots, P_p$). These sub-problems can be regarded as the *child* problems. A solution for the parent problem can then be obtained by resolving the p child problems individually. Let H_0 and T_0 be the number of processors and the processing time required to solve the parent problem respectively. Since all P_i's are independent, they can be carried out sequentially or in parallel. An implementation of the Regional-Rasterization technique to an image rasterizer, which tackles a parent problem by sequentially resolving all the child problems is called a *sequential implementation*. Similarly, if an implementation solves all the child problems in parallel, it is called a *parallel implementation*.

Since all the I_i's are of the same size and mutually exclusive, the cardinality of I_i is m/p. By the evenly distributed argument, it can be assumed that the cardinalities of all O_i's are close to n/p. Hence, in comparison with the parent problem, each child problem P_i can be completed in about T_0/p units of time (since only m/p pixels are concerned) with only H_0/p processors (since there are only n/p objects to worry about). If the p P_i's are to be performed sequentially, the total processing time will be close to T_0 while the number of processors required remains H_0/p. If the p P_i's are carried out in parallel, the required number of processing units will be H_0 and the processing time will be approximately T_0/p. This rough

estimate shows that substantial improvement in cost-effectiveness can be achieved by using the Regional-Rasterization technique.

In Section 6.1, more discussion on this issue is presented.

4.2.2 Required Overhead

The major overhead of Regional-Rasterization comes from the extra local object identification process which must be performed for every graphics object. When the image space is partitioned into more sub-units, the process becomes more complex in the sense that more partitions must be considered per graphics object. The overhead required to implement the local object identification process will be discussed in Chapter 6.

Another major computation overhead, in an implementation of the Regional-Rasterization scheme, is the repeated processing for a certain number of graphics objects. As discussed earlier, a graphics object must be rendered once for every image space partition that it is local to. Therefore, the average number of times that an object must be processed is very likely to be more than 1. In the worst case, each graphics object spans over all the image space partitions and hence must be rasterized once for all partitions. If this happens, the Regional-Rasterization technique looses all of its advantages. Fortunately, the chance of this occurring is very small (see Chapter 5).

CHAPTER 5

PERFORMANCE ANALYSIS FOR THE REGIONAL-RASTERIZATION TECHNIQUE

The Regional-Rasterization technique hypothesizes that (1) the number of graphics objects projected onto each partition are almost the same, and (2) most graphics objects are local to only one partition. The first hypothesis is true if the projections of graphics objects are uniformly distributed in the image space. Furthermore, if the probability of an object projection spanning more than one partition is low, the second hypothesis is also true.

Assume the image space is divided into four quarters. It is easy to recognize that the total number of graphics objects to be rasterized will remain (almost) unchanged, but the number of pixels required to be considered for each graphics object will be quartered. As a result, an optimistic estimate of up to seventy five percent reduction in computational effort is reasonably acceptable. This computational saving can be utilized in the design of a more cost-effective graphics displays. (Similar approaches have long been utilized in sequential systems for improving average case performance).

Although its potential and applicability are unclouded, a more detailed quantitative analysis on the performance of Regional-Rasterization technique is a definite necessity for its refinement and/or further development. In this chapter, an investigation on the expected performance of the

Regional-Rasterization technique, when applied to various graphics environments, is conducted.

5.1 Performance Measures

One of the key problems in both the evaluation and design of graphics systems is the definition of performance. A commonly used definition is the throughput of the system. However, we consider the effectiveness of the technique as a measure of the performance. Here "effectiveness" means the achievable saving in computation effort for rasterizing a scene as a result of applying the Regional-Rasterization technique to a graphics display. Two parameters N_{ipa}, the average number of partitions that a graphics object is local to, and SD_{op}, the standard deviation of the object groups' cardinalities, are crucial to the effectiveness of the Regional-Rasterization technique. In this section, characteristics of these two parameters are discussed.

In implementing Regional-Rasterization, the image space is partitioned into p regions (I_1, I_2, $\cdots$, I_p). According to the definition of the Regional-Rasterization technique, the object space is also divided into p groups (O_1, O_2, $\cdots$, O_p). Notice that the image space I is the union of all the I_i's and the object space O is the union of all the O_i's. That is, $I=\mathbf{U}\ I_i$ and $O=\mathbf{U}\ O_i$. The *cardinality* of a set S is the number of elements in S, which is denoted by $|S|$. Therefore, $|I|$, $|O|$, $|I_i|$, and $|O_i|$

denote the cardinality of I, O, I_i, and O_i respectively. For simplicity, we assume that all the image space partitions are equal and mutually exclusive. Hence $|I|=p|I_1|$.

To render a scene with the object space O, the display with the above implementation of the Regional-Rasterization technique will need to execute $\sum_i |O_i||I_i|$ pel-processes. Since all the $|I_i|$'s are the same, the above summation can be reduced to $|I_1|\sum_i |O_i|$. Let the value $(\Sigma|O_i|)/|O|$ be N_{ipa}. The computation complexity of the above graphics display for the given object space O and image space I becomes:

$$N_{ipa}*|O|*|I_1| \qquad (5.1)$$

Notice that N_{ipa} can be defined as the *average number of image space partitions that a graphics object is local to.* Since a graphics object covers at least 1 and at most p image space partitions, N_{ipa} may vary from 1 to p. Thus, the following inequality is obtained:

$$1 \leq N_{ipa} \leq p \qquad (5.2)$$

The N_{ipa} is equal to 1 when all the graphics objects fall entirely in a unique image space partition; and it is equal to p if all the objects span over all the p image space partitions. Denote $\frac{1}{p}\Sigma|O_i|$, the mean of the $|O_i|$'s, by $|\overline{O_i}|$. N_{ipa} can be expressed as:

$$N_{ipa} = (p|\overline{O_i}|)/|O| \qquad (5.3)$$

To study the effectiveness of Regional-Rasterization, N_{ipa} is a very important parameter to estimate. With N_{ipa}, the computation overhead due to the repeated processing of graphics objects that are local to more than one partitions can be calculated. This overhead is directly proportional to $(N_{ipa}-1)*|O|$ pel processes.

If a parallel implementation is employed, object groups are processed simultaneously. To rasterize the group with more objects will require to execute more pel-processes. As a result, it will take more time or need more processors to rasterize the group. Thus, the scene rasterization speed is determined by the maximium of all the object space cardinalities, $(MAX(|O_i|))$, for all i, $1 \le i \le p$). Therefore, another major factor affecting the performance of the Regional-Rasterization technique is the evenness of graphics objects distribution among the image space partitions. This can be estimated from the *standard deviation of the object groups cardinalities.*

By elementary statistics, the sample variance of a set of p samples $(x_1, x_2, \cdots, x_p)$ of a random variable X is

$$\sqrt{\frac{\sum_{i=1}^{p}(x_i-\bar{x})^2}{p-1}} \quad . \tag{5.4}$$

The standard deviation of the O_i's, SD_{op}, can be obtained by substituting x_i with $|O_i|$ and $\bar{x}$ with $|\overline{O_i}|$ in (5.4). However, the value $SD_{op}/|\overline{O_i}|$ is more meaningful for performance estimation

purposes (see Section 5.6.2).

In the rest of this chapter, estimates of N_{ipa} and SD_{op} are found. In order to obtain reliable estimates, real-life data should be used in the process of estimation. However, obtaining a large set of real-life samples is very difficult. Furthermore, the reliability of the derived estimates of N_{ipa} and SD_{op} is highly dependent on the sampling techniques used to collect data [SCHE79]. To avoid unnecessary complication, we borrow the concepts of *performance modeling*. In the following sections, a performance model is created on which the Regional-Rasterization technique is evaluated.

5.2 Performance Modeling

The *performance modeling* techniques are used to define some quantitative performance measures. In this particular case, the measures are the two parameters N_{ipa} and SD_{op} mentioned in the previous section. Performance modeling forms a basis for the solution to many system design and evaluation problems. Performance models can be used to develop some comparison criteria for the Regional-Rasterization technique that illustrate quantitatively the differences in performance between various implementations. Basically, the performance modeling techniques consist of three parts [CARL84]:

(a) The characterization of an application or class of applications, that is, the characterization of a work-load.

(Performance can be discussed only in the context of a particular application or, perhaps a class of applications.)

(b) The definition of the parameters that characterize the operation of the evaluated system; in this thesis, the Regional-Rasterization technique.

(c) Procedures for calculating performance measures.

The application of performance modeling techniques result in a *performance model* for the Regional-Rasterization scheme.

The second and third parts are readily available. In Chapter 4, a functional model for Regional-Rasterization is given. This functional model describes operation of the technique and thus can satisfy the requirements of (b). For (c), the two expressions, (5.3) and (5.4), described in the previous section, define the procedures for calculating the performance measures. However, the requirements of (a) are not easy to fulfill. The major problem the lack of a general characterization of graphics environments. Therefore, it is very difficult to obtain an exact work-load definition for graphics displays. To get a reasonably close approximation for the work-load, *image modeling* techniques are considered. Through the use of an *image model*, a class of images is defined. The rasterization of these images play the role of work-loads for a graphics display to which the Regional-Rasterization technique is applied. Therefore, the image model must exhibit most major characteristics of images from real-life applications.

5.2.1 Image Modeling

The conceptual validity of the Regional-Rasterization technique is critically tied to the characteristics of the graphics environments that it deals with. For example, an environment in which the graphics objects are so large that most of them cover more than half of the area of the display, the Regional-Rasterization is totally fruitless. Therefore, the two major hypotheses assumed by Regional-Rasterization must be true in order to make it worthwhile to be considered. In this section, we will look into the area of image modeling in order to reveal the general characteristics (if any) of real-life graphics environments.

The role of an image model is to provide a precise description of the image characteristics necessary for an efficient design of image operations. In [AHUJ81] image models are divided into two groups:

(a) Pixel-based models :

These models view individual pixels as the primitives of an image. Specification of the characteristics of the spatial distribution of pixel properties [HAWK70, MUER70] constitutes the image description.

(b) Region-based models :

These models view an image as a set of subpatterns placed according to a given set of rules. Both the subpatterns and their arrangements may be defined statistically, and these subpatterns themselves may be hierarchically composed of

smaller patterns.

In our analysis, the region-based models are more crucial since they can be used to characterize polygon networks, the major geometrical structures in the three dimensional graphics environments. In fact, two dimensional polygon networks are perhaps the most important geometrical structure for representing the projections of visible surface-areas of three dimensional solids.

5.2.2 Region-Based Model

Region-based models are defined using regions, instead of pixels, as primitives. A given model specifies the shapes of the regions and gives the rules for their placements in the plane, thereby allowing increased control over some pattern characteristics. Both the shapes and the distribution rules may be specified statistically.

Over the years, region-based models have received far less attention than pixel-based models in computer graphics and image processing. But recently these models have been investigated for texture analysis and synthesis. In the paper by Tamminen [TAMM81], two dimensional polygon networks generated by the Dirichlet and Poisson-line tessellation models (two important types of region-based models) have been used to model a computer graphics environment in which his EXCELL technique is tested.

One important class of study in region-based modeling is that of *mosaic* models. These models view an image as a mosaic,

constructed by *tessellating* the plane into cells. Each cell corresponds to an individual graphics object. tessellations commonly used include regular triangular, square, and hexagonal tessellations, and random tessellations such as the following:

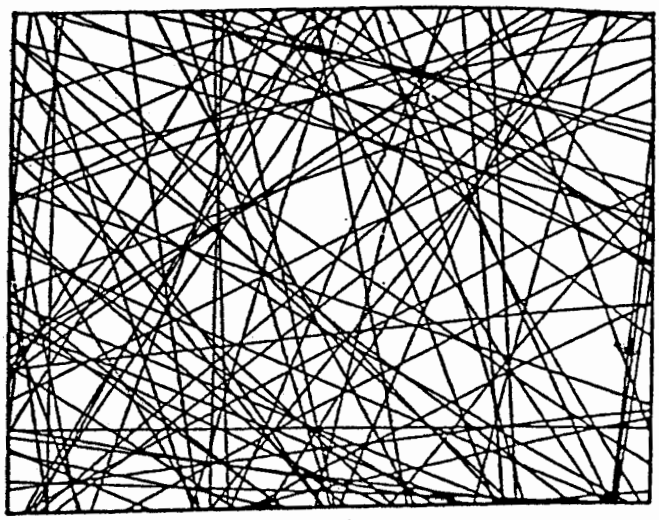
(a) Poisson-line (Random-line) :

A Poisson process chooses pairs (ρ,θ), $0\leq\theta\leq\pi$, $-\infty<\rho<\infty$. The lines $x\cos\theta + y\sin\theta = \rho$, define a tessellation of the plane (see Figure 5.1(a)). A Poisson line model is also known as a random line model.

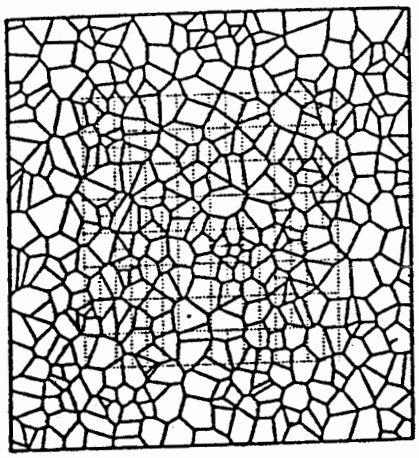
(b) Voronoi : A Poisson process chooses points (nuclei) in the plane. Each nucleus defines a "Dirichlet cell" consisting of all the points in the plane nearer to it than to any other nucleus (see Figure 5.1(b)).

(c) Delaunay : All pairs of nuclei whose Dirichlet cells are adjacent are joined by straight line segments to define the tessellation. Figure 5.1(c) displays an example of Delaunay models.

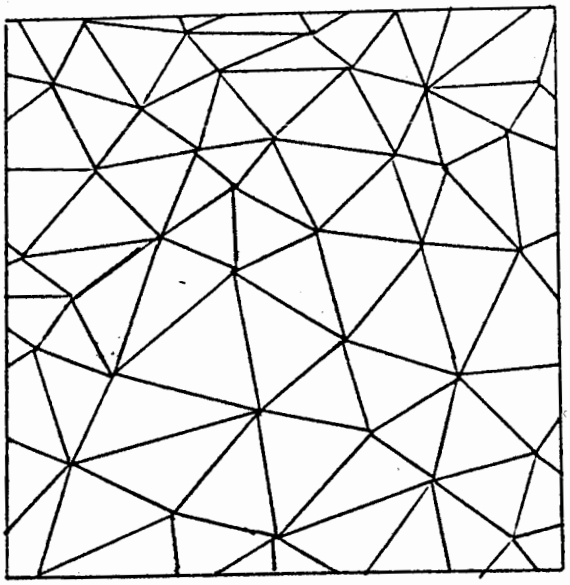
Another major class of region-based models consists of the *coverage* (or *"bombing"*) models. These models view an image as a random arrangement of a given set of shapes over a uniform background. Once again, the choice of shapes and placement rules specifies a particular model. Circles have often been used for the shapes, placed at locations chosen by a Poisson process. Figure 5.2 shows an image generated by a bombing model using circles as the coverage patterns. In [FRAN80,81], Franklin used bombing model images to represent projections of a three



(a)



(b)



(c)

Figure 5.1 (a) Poisson Line (Random Line) Model. (b) Voronoi Model. (c) Delaunay Model.

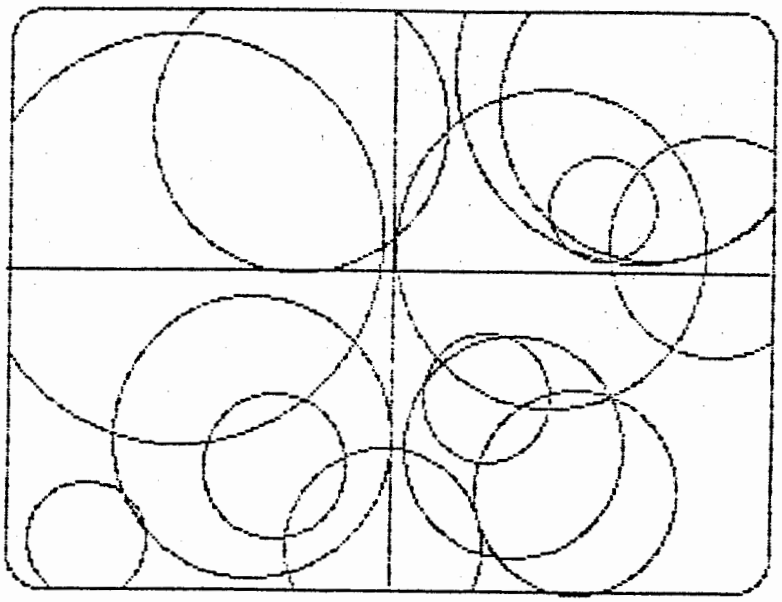


Figure 5.2 Coverage (Bombing) Model.

dimensional graphics environment in which thousands of non-penetrated spheres were packed. His linear-time hidden line removal algorithm is evaluated on these bombing model images.

Region-based models of the above types, mosaic and coverage, have been popular in many disciplines, including geology, forestry, biology, ecology, astronomy, crystallography, and statistics. Several properties of mosaics and coverage patterns have been investigated by researchers in related fields [CRAI76, MILE69,71,80, SWIT65,67,69, SANT72]. Unfortunately, these models are not very useful for polygon network environments. In the

next section, we will discuss the incompatibilities of these models with the polygon network environments. A "customized" image model will then be defined. The performance of Regional-Rasterization on this image model will be investigated later in this chapter.

5.3 Bomosaic Image Model

There exists no concrete standard data generation models for polygon networks, nor is there any standard definition for graphics environments. To estimate the performance of the Regional-Rasterization technique in three dimensional graphics environments, a suitable image model must be defined first. In fact, only for images of a specified type can an evaluation and comparison of algorithms be carried out [AHUJ81].

Basically, an image model for three-dimensional graphics environments is a general way to represent the two-dimensional projection of the environment on a plane (usually the screen). Therefore, an image model must demonstrate some important characteristics of the graphics environments under consideration. However, the chosen model must also be trivial enough to be constructed and studied. In this section, a search of some general characteristics of the graphics environments is carried out. Also, a new image model will be proposed.

5.3.1 Floating-Solid Environments

The types of applications that we are interested in are those which require a large number of polygons for scene modeling and have the ability to generate the image at real-time rate. Just to name some, interactive computer-aided design, movie-quality real-time animation and flight simulation are graphics applications of these types. These types of environments are classified as *floating-solid* environments.

A floating-solid environment can be portrayed by the following characteristics :

(a) The environment contains only three dimensional solids. Since a concave solid can easily be represented by a group of closely related convex solids, we assume that all the solids in the environment are convex.

(b) Statistical attributes of the solids, such as their sizes, shapes, surface curvatures, and placements in the 3-D space, are specific to an application.

(c) The surface of a solid is represented by a polygon mesh.

(d) The number of polygons used to construct the surfaces of the solids is determined by the area and curvature of the surfaces.

(e) The surfaces of the solids in the environment projected onto the screen form networks of 2-D polygons.

(f) The statistical attributes of the 2-D polygons are influenced by the viewing parameters. Even if the 3-D scene was highly correlated, its projection could be much less

correlated [FRAN80].

Notice that the above mentioned statistical characteristics are not only related to the nature of the application but are also time varying. To obtain a precise model for these environments, sufficient a priori knowledge must be available. Unfortunately, this is obviously impossible. In order to simplify the problem and make further analysis feasible, a more restricted form of floating-solid environment will be considered later in this thesis. Meanwhile, let us investigate some important properties of a 2-D projection of a floating-solid environment.

5.3.2 Projection of a Floating-Solid Environment

When a floating-solid environment is projected onto a 2-D plane, such as the screen of the display, few properties of the projection are observed.

(a) The projections of the polygon meshes representing the surfaces of the solids in a floating-solid environment form *clusters* of 2-D polygon networks on the screen. (In this thesis, the screen is considered as the only relevant 2-D projection plane).

(b) A cluster may *overlap* others.

(c) The interior of a cluster is formed by a two dimensional tessellation. Each tessellation cell can be considered as a polygon.

(d) Polygons tend to be smaller as they get closer to the edge

of the cluster.

These properties are depicted in Figure 5.3.

Notice that neither the mosaic model nor the coverage model can precisely define the projection of a floating-solid environment which possesses the above properties. However, a hybrid type of image model, which is a combination of the mosaic and the coverage model, is very appropriate for the current purpose. The new model is given the name *bombing mosaic*, or

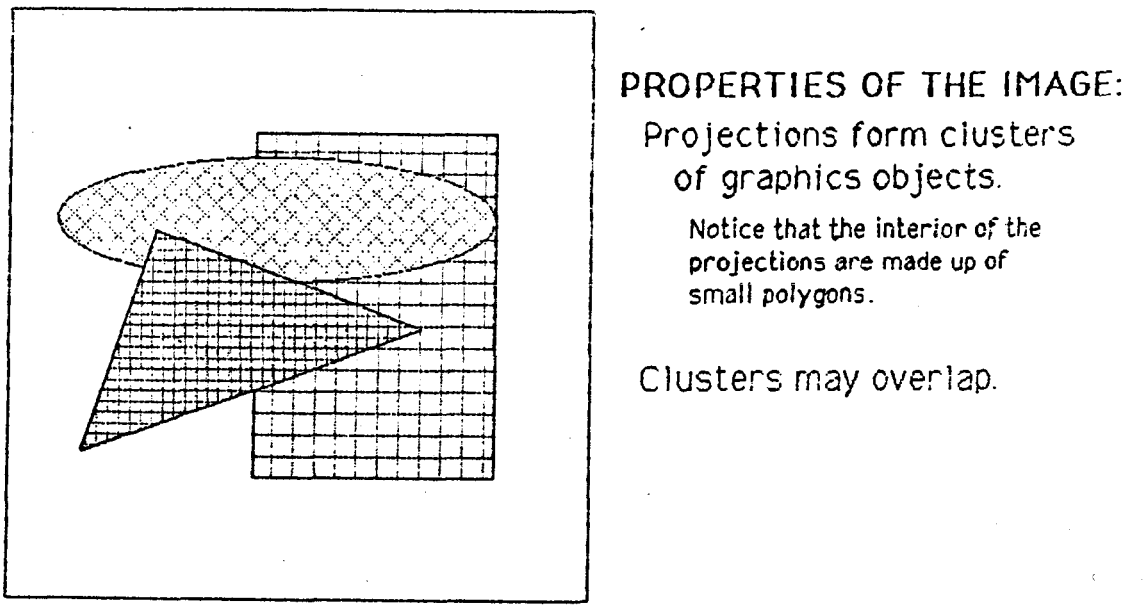


Figure 5.3 Projection of a Floating-Solid Environment.

bomosaic model.

5.3.3 Bomosaic Model

Bomosaic models portray the two-dimensional projection of three-dimensional floating-solid scenes. These models view an image as a random arrangement of patterns. Each pattern is tessellated into a network of polygons. Clearly, this model is a good representation of clusters of polygon network found in the projection of a floating-solid environment.

One weakness of the bomosaic model is its incapability of showing the polygon mesh on the "back" face of solids. However, in most cases, these "back" polygons are eliminated in an early stage of the scene rendering process pipeline, and hence never have to be considered by an image rasterizer.

5.4 C-Bomosaic Image Model

The performance of the Regional-Rasterization technique is to be evaluated based on the bomosaic model. However, the bomosaic model is still too arduous to work with.

The major difficulty in using a general bomosaic image model comes from the unpredictability of the bombing pattern shapes. This novelty of the pattern shapes is a consequence of the uncontrolled solid shapes of the floating-solid environments. To obtain a workable image model, we must restrict ourselves to a more constrained floating-solid environment, namely the

floating-sphere environment. Since it is also a type of floating-solids environment, it retains many of its important characteristics.

A floating-sphere environment is in fact a floating-solid environment which accommodates only spheres. By the symmetry argument of 2-D graphics objects: graphical objects are no more likely to be short and wide than to be tall and thin [SPRO83], circle is perhaps the most appropriate "bomb" pattern to be used in the new bomosaic model. If the symmetry argument is extended to the 3-D case, sphere is the most rational type of solid to be assumed. Furthermore, projections of spheres always form circles, the geometric type which is consistent with the symmetric argument in the 2-D case.

5.4.1 Definiton of the C-Bomosaic Model

Based on the floating-sphere environment and the bomosaic concept for image modeling, a new type of image model, called the *C-bomosaic model* is defined. The C-bomosaic model can be defined by the following characteristics :

(a) The model consists of randomly arranged circles which are called the *bomb circles*. Overlapping of bomb circles is permitted.

(b) The radius of the bomb circles are random within a range.

(c) The interior of a bomb circle is defined by the Delaunay tessellation mechanism.

(d) The number of tessellation cells in a bomb circle is random.

(e) The total number of tessellation cells is limited by a constant.

Figure 5.4 shows an image defined by the C-bomosaic model.

5.5 Generation of Work-Load

The principal overhead of Regional-Rasterization comes from the repeated processing of graphics objects which fall in more than one image space partition (see Chapter 4). Intuitively, if the graphics objects are small, and distributed uniformly, the

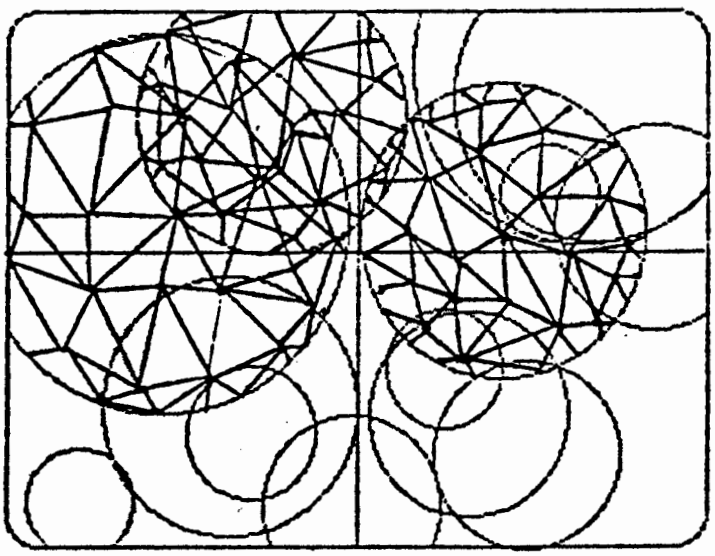


Figure 5.4 A C-Bomosaic Image.

chance of re-processing is low. In this case, we may expect Regional-Rasterization to perform reasonably well. However, owing to the clustering behavior of the graphics objects, performance analysis of Regional-Rasterization is non-trivial. In this section, behavior the of Regional-Rasterization technique on images defined by the C-bomosaic model is studied.

5.5.1 Number of Graphics Objects

The number of polygons that can be used to "tile" the solids is limited. (This number is especially important to many parallel-processing systems such as Fussel's real-time scan conversion system [FUSS82] in which an upper bound on the number of objects is set when the system is built.) This number must be reasonably large for most scenes and should be consistent with the physical limitation of most systems. "> In this thesis, the *maximum number of graphics objects* to be considered is denoted by N_p. Since our major interest is in graphics environments which consist of a large number of graphics objects, we will consider cases where N_p is equal to 500, 5,000, or 50,000.

5.5.2 Tessellation of the Bomb Circles

There are two questions associated with the tessellation of bomb circles: 1) What kind of tessellation is to be used (Poisson line, Voronoi, or Delaunay)? and 2) How many tessellation cells should there be in a bomb circle?

To answer the first question, we present the following arguments: 1) Both the Voronoi and Delaunay models give a similar tessellation pattern. However, the Delaunay tessellation method is more similar to networks produced by many surface modeling algorithms [BARN83]. 2) A Poisson line (random line) tessellation exhibits extreme variation of polygon size. The polygon network formed is also unrealistic for a projection of a 3-D polygon mesh. By these two arguments, the Delaunay tessellation appears to be more appropriate for tessellating bomb circles.

There are two major factors affecting the size of polygon tiles on the surface of a solid : 1) its surface area, and 2) the rate of change of the surface curvature. When performing the perspective transformation, solids at farther distances give smaller projected images on the screen. Therefore, even a small projected image may contain a large number of polygons. Thus, the number of tessellation cells to be found in a bomb pattern should be independent of the pattern's area. However, it is more realistic to impose a lower and an upper bound on the number of tessellation cells per bomb circle. Hence, this number is allowed to vary only within a certain range. In order to avoid the situations in which a very small bomb pattern contains a very large number of tessellation cells, the range is also partially determined by the area of the bomb circle. If the area of a bomb circle is small, it has a smaller bound for the tessellation cell number.

5.5.3 Vertices for the Delaunay Triangle Network

The first step in the tessellation process is to obtain a set of vertices. These vertices correspond to the nodes (Delaunay vertices) of a polygon network which represents the surface of a sphere whose projection forms a bomb circle. In Figure 5.5, the surface of a sphere of radius R is defined by a function f which maps (R,θ,ϕ) to a unique point on the surface.

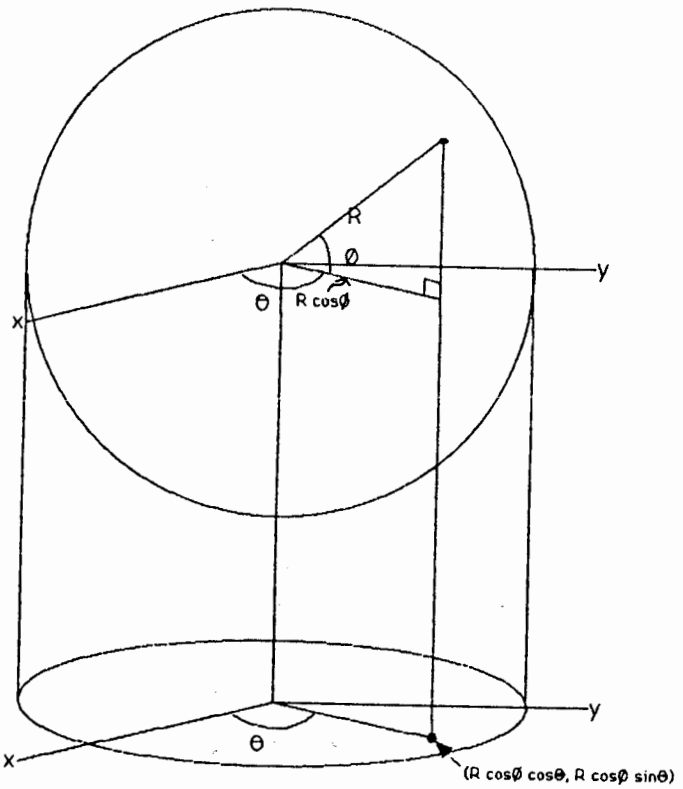


Figure 5.5 Projection of a Surface Point from a Sphere to a Plane.

By varying θ and ϕ from 0 to 2π (exclusively), the function f defines all the points on the surface of the sphere. As shown in Figure 5.5, the point $f(R,\theta,\phi)$ projects onto the point $(R\cos\phi\cos\theta, R\cos\phi\sin\theta)$ on a two dimensional Cartesian plane. Since the nodes of the polygon network are actually points on the surface of the sphere, each of them can be denoted by the (R,θ,ϕ) notation and its projection is expressed as $(R\cos\phi\cos\theta, R\cos\phi\sin\theta)$. Since the "back" polygons in the polygon network are ignored, the angle ϕ is allowed to vary uniformly between $-\pi/2$ and $\pi/2$ only. However, θ can vary uniformly between $-\pi$ and π.

The algorithm GenVL is used to generate the Delaunay vertices of a bomb circle given its location (x,y), radius R,

and number of vertices n:

```
Comment : RANDOM(type,lower,upper) is a random
          value generator which returns a value
          of data type <type> within the range
          {<lower>,<upper>}.

Comment : VL is a list of ordered pairs which is
          initially empty.

Comment : Vx, Vy are local variables.

Routine GenVL (x,y,R,n) return (VL);

1 Repeat n times;

   1.1  θ := RANDOM(real,-π,π);

   1.2  φ := RANDOM(real,-π/2,π/2);

   1.3  Vx := x + R*cosφ*cosθ;

   1.4  Vy := y + R*cosφ*sinθ;

   1.5  Append (Vx,Vy) to VL;

2 Return VL;
```

Figure 5.6 displays a set of vertices generated by the GenVL routine. The VL can be fed into any Delaunay triangulation algorithm. In this thesis, the divide-and-conquer algorithm proposed in [LEED80] is used.

5.5.4 Generation of a tessellated Bomb Circle

There are three main parameters governing the appearance of a tessellated bomb circle: (1) its location (x,y), (2) its radius R, and (3) the number of tessellation cells in the bomb circle N_{tc}. The location of a bomb circle (x,y) is assumed to be random on a 1024x1024 Cartesian plane with integral x and y

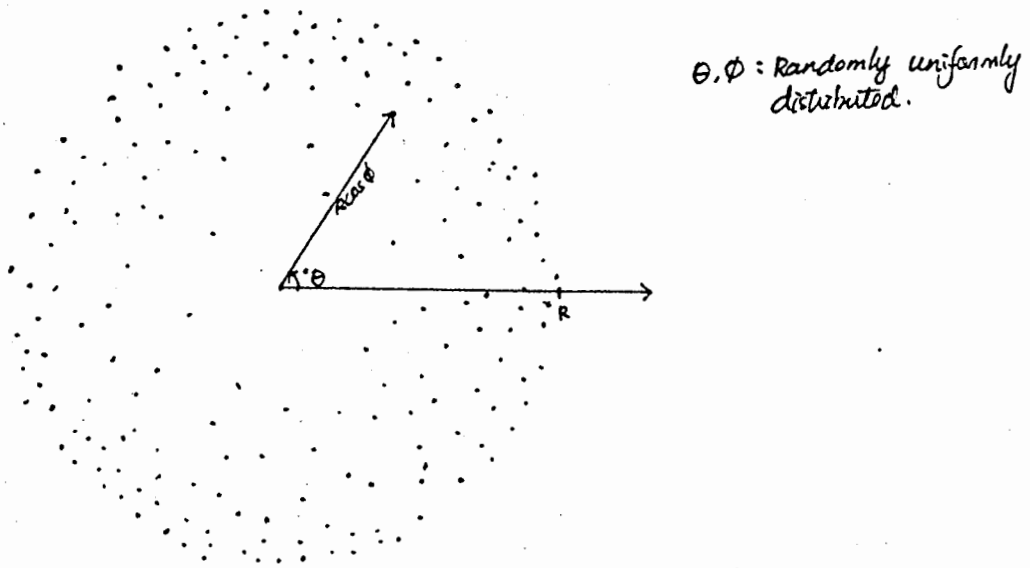


Figure 5.6 A Set of Vertices Generated with the Algorithm **GenVL**

coordinates. The radius R of a bomb circle is also random. However, R is only allowed to vary from 36 to 365. In Figure 5.7, the size of the largest and smallest bomb circle are depicted. In determining the number of tessellation cells, we set a rule which prevents too many cells to be formed in a small circle. The rule states that the maximum number of tessellation cells in a circle is directly proportional to the radius of the bomb circle. Also, to constrain the number of bomb circles in an image to be within a reasonable range, the number of tessellation cells in a bomb circle cannot be too small nor too

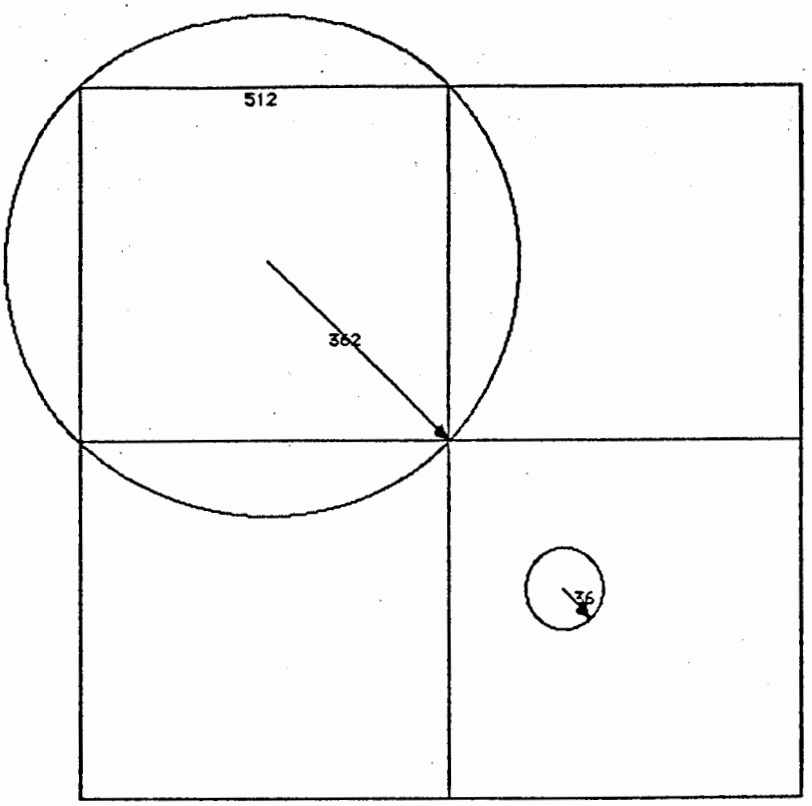


Figure 5.7 The Biggest and Smallest bomb circle.

large. In this thesis, the number of tessellation cells is allowed to vary from $N_p/100$ to $((N_p/2)/(R/365)) = R*N_p/730$. With this upper bound, even the largest allowable bomb circle contains at most one half of the maximum number of graphics objects in the scene. Consequently, there may be at least 2 and at most 100 bomb circles in the image.

The objective of using Delaunay triangulation is to obtain a polygon network such that the spatial characteristics of the polygons can be studied. Any Delaunay tessellation algorithm can be used to generate the desired triangle network. The measurement of the spatial characteristics of the Delaunay triangles can be performed after the tessellation has been completed. However, we found it more effective if the spatial statistics of the triangles can be collected at the same time as triangles are generated. With this in mind, Lee and Schachter's divide-and-conquer algorithm is used [LEED80].

The basic idea of Lee and Schachter's algorithm is to merge two sub-Delaunay triangle networks into a larger one [LEED80]. In the process of merging two sub-networks, new edges are inserted and a new Delaunay triangle is formed for each edge added. At this moment, it is very easy to identify the three vertices of the newly formed triangle; and hence identify the image space partition(s) that it is local to. With a minor modification, the triangulation algorithm can only be used to reveal the amount of local triangles of a given set of partitions.

The modified algorithm (MDT) is given an array (CNT) of p entries where p is the number of image space partitions. Each element of the array serves as a counter for local triangles of an image space partition when a new triangle is generated. MDT will test it against all the partitions. Then, the counters in CNT will be updated accordingly. This array of counts, along

with the actual number of triangles generated (N_t), are returned after the tessellation is completed.

5.5.5 Spatial Statistics Sampling

A C-bomosaic image is composed of a number of circles each of which is tessellated by the MDT algorithm. The total number of Delaunay triangles found in a C-bomosaic image is limited by a pre-determined upper bound N_p. To keep track of the total number of Delaunay triangles in a generated C-bomosaic image, the number of the tessellation cells in each bomb circle must be known. Unfortunately, the exact number of cells generated by the MDT algorithm is very difficult to control (we can only limit the number of vertices). Therefore, generating a C-bomosaic image with exactly N_p triangles is laborious. To overcome this problem, we make use of the observation that the number of Delaunay triangles generated on a set of vertices is approximately equal to the number of vertices in the set.

In the C-bomosaic routine given below, statement 3.4 restricts the sum of the number of Delaunay triangles already generated (<total> in 3.7) and the number of vertices in the new bomb circle (V_n in 3.4, 3.5) to be not greater than N_p. Since the number of tessellation cells in the new bomb circle will approximately equal to the vertex number, the value (<total> + $<T_n>$ (in 3.7, 3.8)) will be close to (<total> + $<V_n>$).

The spatial statistics N_{ipa} and SD_{op} are obtained in statements 5 and 6 according to expressions (5.2) and (5.3)

found in Section 5.1.

```
Comment : IP is an array of structures
          of p elements. Each structure holds the
          boundary of an image space partition.

Comment : RANDOM(type,lower,upper) is a random
          value generator which returns a value
          of data type <type> within the range
          {<lower>,<upper>}.

Comment : VL is a list of ordered pairs which is
          initially empty.

          MaxT is the maximum number of remaining
          cells.

          V_n is the number of tessellation
          vertices for the next bomb circle.

          T_n is the total number of tessellation
          cells just generated in the bomb circle.

          CNT is an array of p elements which holds
          the objects count for each image space
          partition.

Routine C-Bomosaic (N_p,IP,p);

1 MaxT := N_p;

2 CNT(i) := 0 for all i, 1≤i≤p;

3 Repeat;

   3.1  x := RANDOM(int,0,1023);

   3.2  y := RANDOM(int,0,1023);

   3.3  R := RANDOM(int,36,365);

   3.4  V_n := MIN(MaxT,RANDOM(int,(N_p/100),(R*N_p/730));

   3.5  VL := GenVL(x,y,R,V_n);

   3.6  (CNT,T_n) := MDT(VL,IP);

   3.7  total := total+T_n;

   3.8  MaxT := MaxT-T_n;

   3.9  if MaxT<(N_p/100), then exit Loop 3;
```

4 mean := Σ CNT(i)/p;

5 N_{ipa} := p*mean/total;

6 SD_{op} := $\sqrt{((\Sigma\ (CNT(i)-mean)^2)/(p-1))}$;

7 Return (N_{ipa}, SDop/mean);

5.6 Estimation of Performance Measures

To understand the effectiveness of the Regional-Rasterization technique in graphics environments of various complexities, the N_{ipa} and SD_{op} are estimated with various N_p's. Three values of N_p's are considered: 500, 5000 and 50,000. These N_p's correspond to the complexity of a low, medium and high quality image. For each N_p the N_{ipa} and SD_{op} are estimated for 9 different implementations of the Regional-Rasterization technique. These implementations have image space partitions of equal size which are formed by slicing the image space into an equal number of columns and rows. Therefore, the image space partitions are all squares and have the same area. For each particular scene complexity and implementation, 50 C-bomosaic images are generated. For each image, the N_{ipa} and SD_{op} corresponding to the Regional-Rasterization implementation are computed. As a result, 27 sets of 50 N_{ipa} samples and 27 sets of 50 SD_{op} samples are generated. The entries found in Table 5.1 and Table 5.2 are the means of the 50 N_{ipa} and SD_{op} samples in these sets.

# Partitions	Low (N_p=500)	Medium (N_p=5,000)	High (N_p=50,000)
1	1	1	1
4	1.307	1.113	1.040
9	1.589	1.201	1.075
16	1.911	1.346	1.117
25	2.163	1.387	1.143
36	2.467	1.519	1.183
49	2.824	1.634	1.219
64	3.133	1.728	1.250
81	3.445	1.847	1.287
100	4.260	1.952	1.316

Table 5.1. Estimate of expected N_{ipa}'s for various scene complexities. Number of image-space partitions varies from 1 to 100.

# Partitions	Low (N_p=500)	Medium (N_p=5,000)	High (N_p=50,000)
1	0	0	0
4	0.625	0.578	0.520
9	0.637	0.610	0.548
16	0.731	0.657	0.581
25	0.789	0.718	0.648
36	0.903	0.771	0.673
49	0.934	0.823	0.719
64	0.967	0.877	0.766
81	0.981	0.913	0.819
100	1.034	0.937	0.826

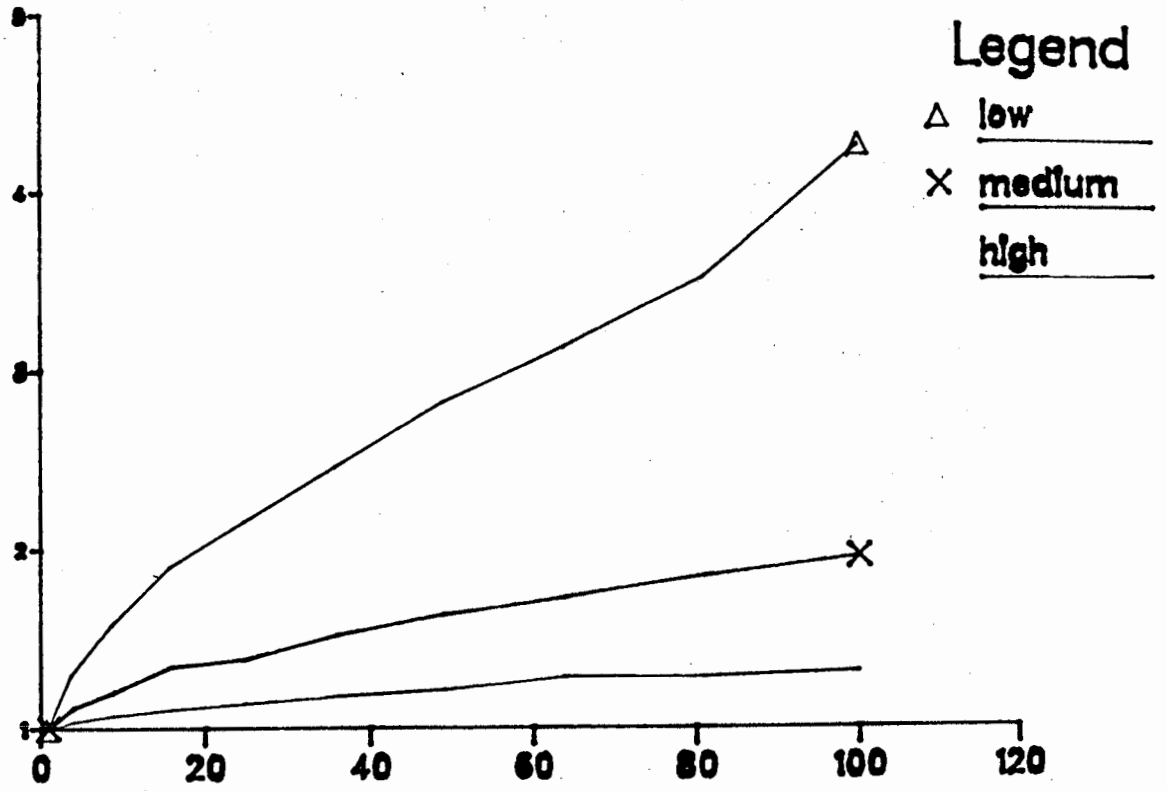
Table 5.2. Estimate of expected $(SD_{op}/|O_i|)$'s for various scene complexities. Number of image-space partitions varies from 1 to 100.

5.6.1 Analysis of the N_{ipa} Estimates

From the estimates displayed in Tables 5.1 and 5.2, we conclude that the Regional-Rasterization technique is very effective in improving the performance of graphics displays. In Table 5.1 there are three columns and nine rows of entries. The three columns correspond to three preset scene complexities. The first one is for graphics environments which contain approximately 500 objects, the second one is for environments with 5000 objects and the third is for very complex scenes which have 50,000 graphics objects. The leftmost number of each row is the number of image space partitions used in a Regional-Rasterization implementation. To analyze the results, Graph 5.1 is plotted based on the information from the table.

In Graph 5.1, three curves are plotted in a plane whose x and y represent the number of image space partitions and the estimated N_{ipa}, respectively. As illustrated in the graph, the estimated N_{ipa} increases as the number of image space partitions increases. This can be explained by the fact that the chance for a graphics object to cross a partition boundary increases as the area of a partition decreases. (This can also be shown more rigorously, using the concepts of geometric probability [SANT76].) Nevertheless, if the number of graphics objects in the environment is larger, the N_{ipa} is closer to its lower bound, namely one. To justify this result, we provide the following arguments:

(a) The average area of graphics objects tends to be smaller as



Graph 5.1 Estimate of N_{ipa}'s plotted vs number of partition.

the total number of graphics objects increases. In the C-bomosaic model defined in this thesis, this has been assumed implicitly.

(b) The expected number of graphics objects required to

constitute a polygon network of a fixed area increases linearly with the average object size.

(c) The average *breadth*[1] of a graphics object decreases at a rate much slower than that of the area of the object. For example, the diameter of a circle can be expressed as $\sqrt{(A/\pi)}$.

(d) The graphics objects in a polygon network are non-overlapping. Therefore, the expected number of objects in a network that a line of fixed length may intersect is inversely proportional to the average breadth of the objects. By argument (c) the rate of increase in the number of graphics objects intersecting a partition boundary is much slower than the decrease rate of the average object area. By arguments (a) and (b), the number of objects in a graphics environment increases more rapidly than the resulting increase in the number of graphics objects intersecting a partition boundary.

The first conclusion that is derived from Graph 5.1 is that Regional-Rasterization is most effective in complex graphics environments. Also, the computation overhead resulting from repeated rendering of a graphics object is small if the number of partitions is small. However, although more overhead is required for larger number of image space partitions (up to a certain number), Regional-Rasterization may still be favorable.

---

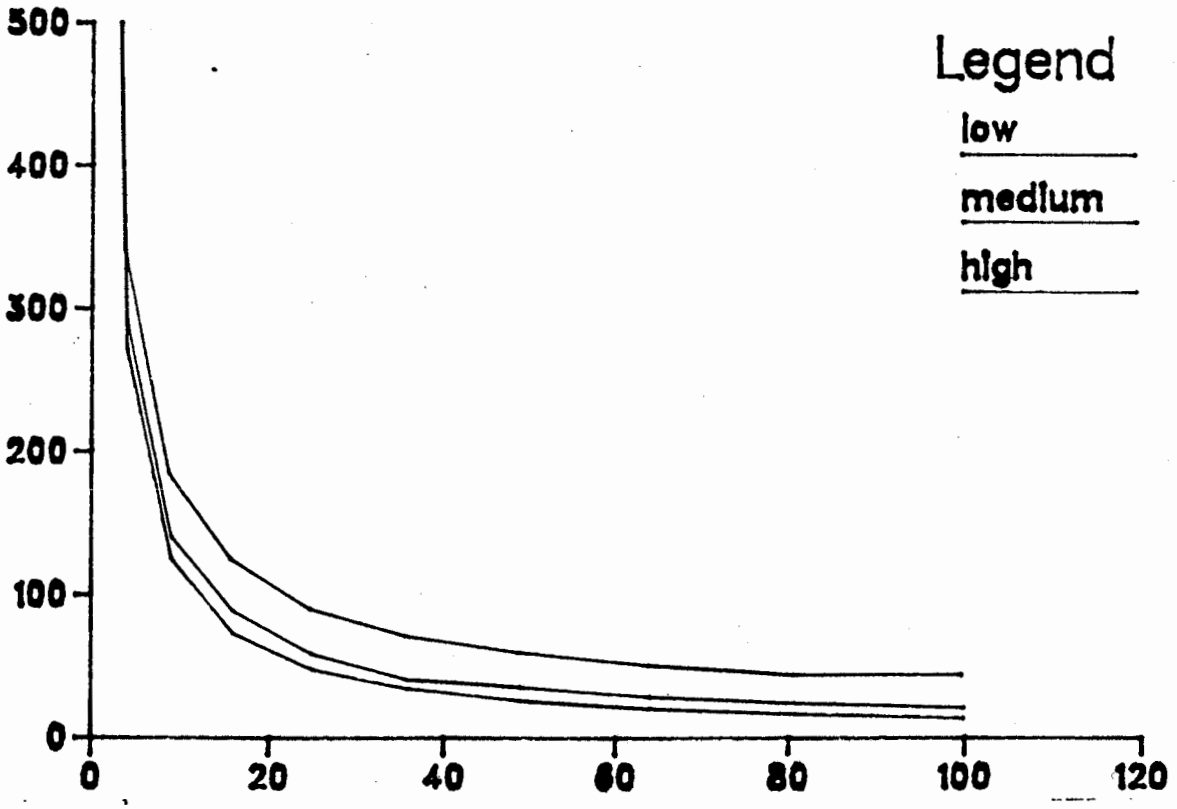
[1]The breadth of a convex set in a direction *d* is the distance between the two support lines (tangents), which are perpendicular to d and are on opposite sides of the set [SANT76].

In Table 5.3, the expected numbers of pel-processes required to render an object for a low, medium and high quality scene with different Regional-Rasterization implementations are tabulated. As shown, the expected number drops as the number of image space partitions rises. Nevertheless, when the data from Table 5.3 is plotted out, it is more obvious that the rate of improvement (the amount of pel-processes reduced per image space partition increased) drops as the partition number increases. A tremendous saving can only be achieved by using more partitions when the partition number is small, say from 1 to 9.

# Partitions	Low (N_p=500)	Medium (N_p=5,000)	High (N_p=50,000)
1	1048.576	1048.576	1048.576
4	342.622	291.766	272.630
9	185.131	139.927	125.247
16	125.239	88.211	73.204
25	90.722	58.174	47.940
36	71.857	40.244	34.457
49	60.432	34.967	26,084
64	51.331	28.311	20.480
81	44.597	23.910	16.661
100	44.669	20.468	13.799

Table 5.3. Expected number of pel-processes required to rasterize an object, for various scene complexities. Number of image-space partitions varies from 1 to 100.

In Graph 5.2, the corresponding expected numbers of pel-processes required to render an object in a low, medium, and high quality scenes are plotted for various number of image space partitions. From all the three graphs, the expected number



Graph 5.2 Expected numbers of pel-processes plotted vs number of partitions.

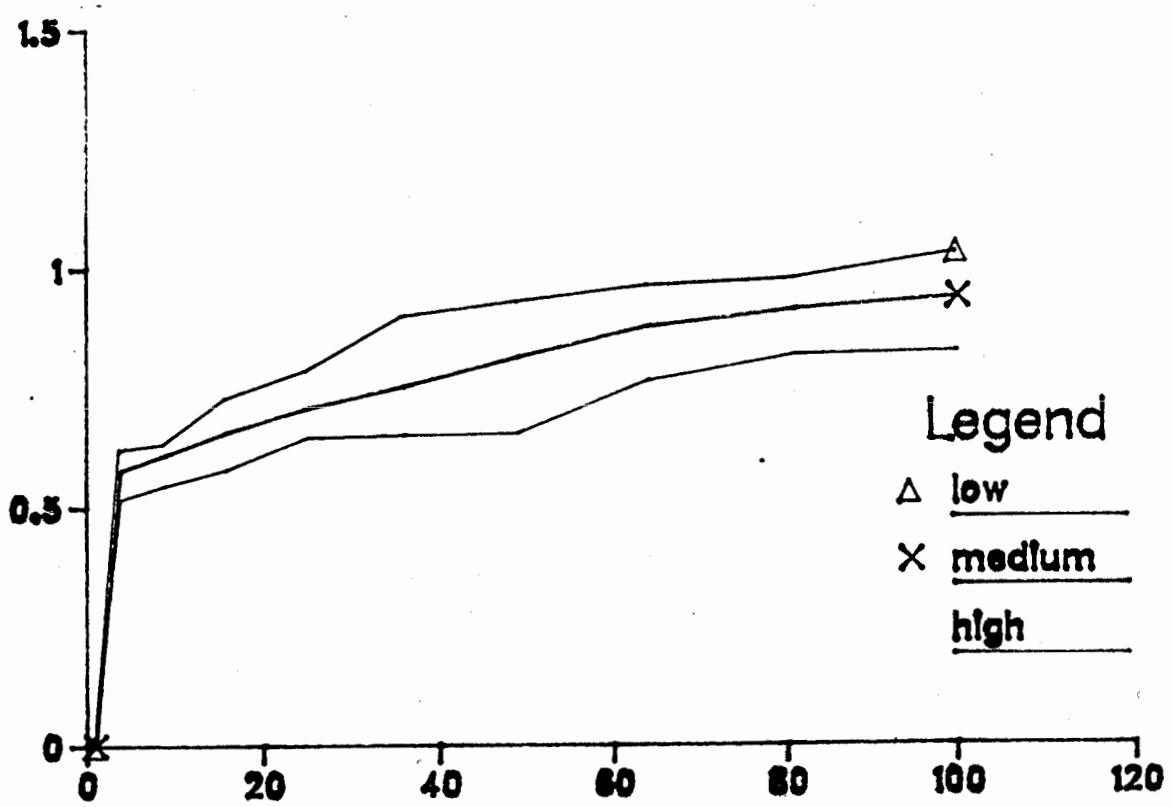
of pel-processes needed for each environment approaches asymptotically to a lower bound. This implies that improvement will stop when a certain lower bound is hit. Theoretically, this lower bound is the average area of the graphics objects in the graphics environment. Therefore, the second conclusion is: it is not rcommended to divide the image space into a large number of partitions.

5.6.2 Analysis of the SD_{op} Estimates

Expression (5.4) in Section 5.1 and statement 6 of the C-Bomosaic routine (see Section 5.5) is the standard deviation of the number of graphics objects that fall in various image space partitions. The major objective of studying the behavior of the SD_{op} is to analyze the distribution of graphics objects in the image space. However, the SD_{op} alone is meaningless since it cannot display the variation of graphics object population density in the image space partitions. For example, the standard deviation of 4 for a set of samples whose mean is 4 indicates a great fluctuation among the sample values. Whereas, if the same standard deviation is found in a sample set whose mean is 400, it can be concluded that all the sample data cluster around the sample mean. Therefore, in Table 5.2, the value of standard deviation divided by the sample mean is used.

In Table 5.2, the spatial distribution of the graphics objects in a scene seems to be correlated with both the complexity of the scene and the number of image space

partitions. The entries in Table 5.2 are plotted against the number of $SD_{op}/|\overline{O_i}|$ image space partitions in Graph 5.3. As illustrated, smaller $SD_{op}/|\overline{O_i}|$ is found for higher quality images. However, the influence of the number of graphics objects



Graph 5.3 Estimate of $SD_{op}/|\overline{O_i}|$ plotted vs partition number.

in a scene is not very serious. As a result, all the three curves are close to each other. This is due to the argument that spatial characteristics of bomb circles in an image are not altered by the number of tessellation cells found in each bomb circle. Since the spatial characteristics of bomb circle are similar in low, medium and high quality cases, the evenness of graphic object distribution is almost the same in the three kinds of images. Nevertheless, owing to the MDT algorithm used to generate the C-bomosaic images, images with larger N_p will, in general, have more bomb circles. Since these circles will distribute uniformly in the image space, smaller $SD_{op}/|\overline{O_i}|$ in images with larger N_p is sensible.

In Graph 5.3, it is also illustrated that larger $SD_{op}/|\overline{O_i}|$ is resulted when an implementation incorporates a larger number of image space partitions. To explain this phenomenon, we must consider the clustering property of the graphics objects. Since they form networks, the graphics objects localize at few certain positions in the image space. This implies that graphics objects in the cluster should fall in few abutted partitions only (see Figure 5.8(a)). The effect of this characteristics on the evenness of object distribution is depicted in Figure 5.8(b). As shown, the chance that a partition is mostly filled or mostly empty is higher if there are more partitions. Consequently, the numbers of objects in the partitions has a greater fluctuation.

The most important concept to be gathered from our studies in this chapter is that Regional-Rasterization is most effective

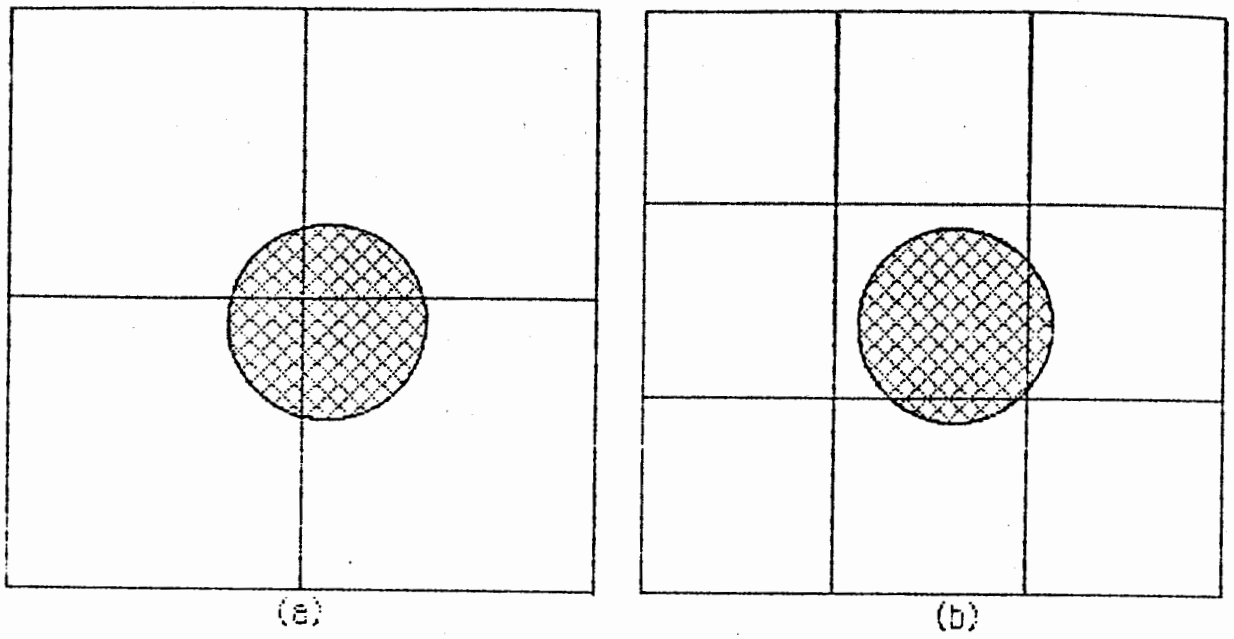


Figure 5.8 (a) Number of graphics objects in each partition is almost equal.
(b) When there are more partitions, most objects are local to only a few partitions.

when the number of partitions is small (eg. four).

CHAPTER 6

IMPLEMENTING THE REGIONAL-RASTERIZATION TECHNIQUE

The Regional-Rasterization technique is based on two major elements: (1) a scheme to partition the image-space, and (2) a process to identify graphics objects local to a certain partition. In this chapter, various design considerations in implementing Regional-Rasterization are discussed. In sections 6.1, 6.2, and 6.3 we will concentrate on finding a suitable image space partitioning scheme and on the hardware design of a high speed local graphics object identifier.

Through the discussions in this chapter, the feasibility of Regional-Rasterization is confirmed. In sections 6.4 and 6.5, the applicability of the technique to parallel-processing image rasterizers is demonstrated. Two examples are given: one describes the application of Regional-Rasterization to Pixel-Planes system [FUCH81]; and the other describes how the technique can be applied to Fussel's system [FUSS82]. These two systems represent two extreme architectural structures for image rasterization: the image-space partitioned structures and the object-space partitioned structures.

6.1 Modular Approach to Image Rasterization

Let a *module* be a "black-box" which is specially made to attack the child problems (see Section 4.2.1). Obviously, a

sequential implementation requires only one module. However, in the case of parallel implementations, at least p modules are needed (p is the number of partitions employed). A module is basically a device for image rasterization. There are many ways to build a module.

"> "> "> "> By the generalized notation for graphics display architectures (see Chapter 3), a module can be defined with the α-β-γ-λ notation. With that notation, its hardware and processing-time complexities can be defined by $\beta\lambda$ and $\alpha\gamma$ respectively.

Recall the definition of α, β, γ, and λ: the product $\alpha\beta$ must be greater than or equal to the number of graphics objects to be scan-converted; and the product $\gamma\lambda$ must be greater than or equal to the number of pixels in the image space. For the module associated with the image space I_i, the corresponding value $\alpha\beta$ must be greater than the number of objects in O_i, and the value $\gamma\lambda$ must be greater than the pixels in I_i. Assume the object space of the parent problem contain n objects and its image space has m pixels. "> "> Then, the number of pixels in I_i is $m^*=m/p$. Since all the partitions are of the same size, the $|I_i|$'s are all equal for all $i=1,2,\ldots,p$. Owing to the spatial characteristics of the graphics objects, the total number of objects to be considered by all the modules is $n^*=N_{ipa}(n/p)$ where N_{ipa} is the average number of partition(s) to which a graphics object is local (see Section 5.1).

If a $1\text{-}n^*\text{-}m^*\text{-}1$ architecture is employed by a module, it will require n^* processors and its computational rate is m^*. Therefore, a sequential implementation with a $1\text{-}n^*\text{-}m^*\text{-}1$ module architecture will need $n^*=N_{ipa}(n/p)$ processors and its computation rate becomes pm^*, which is equal to m. For a parallel implementation, the hardware complexity will be $pn^*=N_{ipa}(n)$ and the computational rate will be $m^*=(m/p)$.

If a $n^*\text{-}1\text{-}1\text{-}m^*$ architecture is employed by the modules, a sequential implementation will have a hardware complexity $m^*=(m/p)$ and a computational rate $pn^*=N_{ipa}(n)$. Similarly, for a parallel-implementation, the hardware complexity is of order $p(m^*)=m$ and the computational rate $n^*=N_{ipa}(n/p)$. These measures are tabulated in Table 6.1.

	$1\text{-}n^*\text{-}m^*\text{-}1$	$n^*\text{-}1\text{-}1\text{-}m^*$
Sequential Implementation		
Hardware Complexity	$N_{ipa}(n/p)$	m/p
Processing-Time Complexity	m	$N_{ipa}(n)$
Parallel Implementation		
Hardware Complexity	$N_{ipa}(n)$	m
Processing-Time Complexity	m/p	$N_{ipa}(n/p)$

Table 6.1. Hardware and Processing-time Complexities for Sequential and Parallel Implementations of Regional-Rasterization.

The choice of a sequential or a parallel implementation all depends on what is the objective for incorporating the Regional-Rasterization technique. Obviously, sequential implementation is particularly suitable for cases in which the hardware quantity is to be cut down. As illustrated in Table 6.1, a sequential implementation can achieve a factor of (N_{ipa}/p) and $(1/p)$ cut down in hardware for the 1-n^*-m^*-1 and n^*-1-1-m^* cases, respectively. However, parallel implementations are good in improving the system's processing-time. Again, from Table 6.1, the potential of a parallel implementation can be shown by the improvement factor of $(1/p)$ and (N_{ipa}/p) in processing-time respectively for the 1-n^*-m^*-1 and n^*-1-1-m^* cases. As is shown, the performance of an implementation is closely related to the value of N_{ipa} and p.

6.2 IP-Scheme for the Optimal Implementation

In Table 6.1, performance measures for various implementations of the Regional-Rasterization technique are posted. A general characteristic of these measures, due to p, is that the effectiveness of the technique increases as the number of partitions increase. Intuitively, the image space partitioning scheme employed in Regional-Rasterization should have a large number of partitions for better performance. However, a factor, N_{ipa}, causes a considerable degree of degradation to an implementation's effectiveness if p is large. From the performance study in Chapter 5, we know that N_{ipa} is a

monotonic increasing function of p. That is, N_{ipa} increases as p increases. In addition, the rate of increase of N_{ipa} is faster than the growth of p. Consequently, as p increases, the effectiveness improvement will eventually be compensated by the degradation in effectiveness induced by N_{ipa}. This suggests that there is an optimal value for p at which the effectiveness of the Regional-Rasterization technique is maximal. This behavior has been depicted in Graph 5.2.

In the graph, the expected number of pel-processes required to execute in order to rasterize an object is plotted against p. For the case N_p=500 (N_p is the maximum number of objects in the graphics environment), a local minimum is found at p=81. Furthermore, it has demonstrated that when N_p=5,000 or 50,000, the expected pel-processes number approaches their minimum asymptotically. However, the theoretical optimal IP-scheme may not be the most suitable one to be used mainly because of its cost-effectiveness.

Upon implementation of optimal IP-scheme, there is usually a large number of image space partitions formed. This amplifies the difficulty in performing the local object identification. In addition, the number of partitions becomes less significant to the system performance as p grows large. This characteristic is as well depicted in Graph 5.2. As is shown, the pel-processes number drops rapidly as p shrinks. Nevertheless, the rate of decrease slows down as p increases. Therefore, it is advisable to adopt an IP-scheme thereby forming a small number of image

space partitions, such as 4, 9 or 16.

Based on the above arguments, we can reasonably assume that the IP-scheme for the optimal implementation of Regional-Rasterization will employ a smaller number of image space partitions than the theoretical optimal IP-scheme. (Note that it is not necessary for the optimal implement to employ the theoretical optimal IP-scheme.) This number may be much smaller than the one corresponding to the theoretical optimal IP-scheme. Supported by this idea, the local object identifier presented in the next section is designed aiming at a small number of partitions.

6.3 Local Graphics Object Identifier

One of the key components of the Regional-Rasterization technique is the local graphics object identifier. The major function of this component is to identify the partition(s) that a graphics object is local to. This component is conceptually an algorithm which can be implemented in software or hardware. In this section, the design of a low-cost high-speed local graphics object identifier is discussed.

6.3.1 Input and Output

The local object identifier is a device which tests each graphics object against all the image space partitions and identifies the partitions that the object is local to. The input

to the device must contain information about the boundary of an object. The borders of the partitions must also be known to the identifier. However, since partition borders are static and will never change, they are assumed to have been "burnt-in". Information regarding the objects is obtained from the output of a series of geometric transformation processes, which are usually done by special hardware.

The identifier can be fitted in the scene rendering process pipeline preceding the image rasterizer, where Regional-Rasterization is implemented. At this stage, the boundary of an object is described by the vertices of the object expressed in screen coordinates. After an object is fed to the identifier, an identification procedure is executed immediately. The output of the device is a set of labeled objects. The labels are used to identify a set of partition(s) that a graphics object is local to. The output can be in any format. However, a string of p bits, where p is the number of partitions to be considered, is sufficient. If p is too large, the output can be in a more compact encoded form.

6.3.2 Speed Requirement

If the local graphics object identifier is fitted in the scene rendering process, it must be capable of identifying an object at a rate not slower than any other device in the pipeline. Otherwise, the identifier will slow down the throughput of the entire system. With the current technology,

other devices in the scene rendering pipeline such as the geometric transformers, clippers, ..., etc., may have a very high throughput rate. Therefore, the primary design criterion for the identifier is speed.

However, the identification process is not a simple task. To check for overlaps of an object and a partition, all the edges of the object must be tested against all the borders of the partition. This involves many multiplications and additions. If any intersection occurs, the object must overlap with the partition and hence be local to it. Direct implementation of this approach will require very complex circuitry.

6.3.3 Algorithm "INEXACT"

The algorithm used in our design is not exactly the one described in the previous section. This algorithm is given the name INEXACT in the sense that it is not solving the mentioned identification problem exactly. However, the result produced by INEXACT is so close to the optimal solution that it is very useful for our application.

INEXACT views the image space as a matrix of partitions and an object as a set of vertices (see Figure 6.1). It considers all the partitions in the *minimal rectangular* sub-matrix of partitions, that enclose the object entirely, as the partitions that the object is local to. This approach sometimes mistakenly identifies an object and forces it to be unnecessarily rendered in partitions that it is not local to; and hence wastes

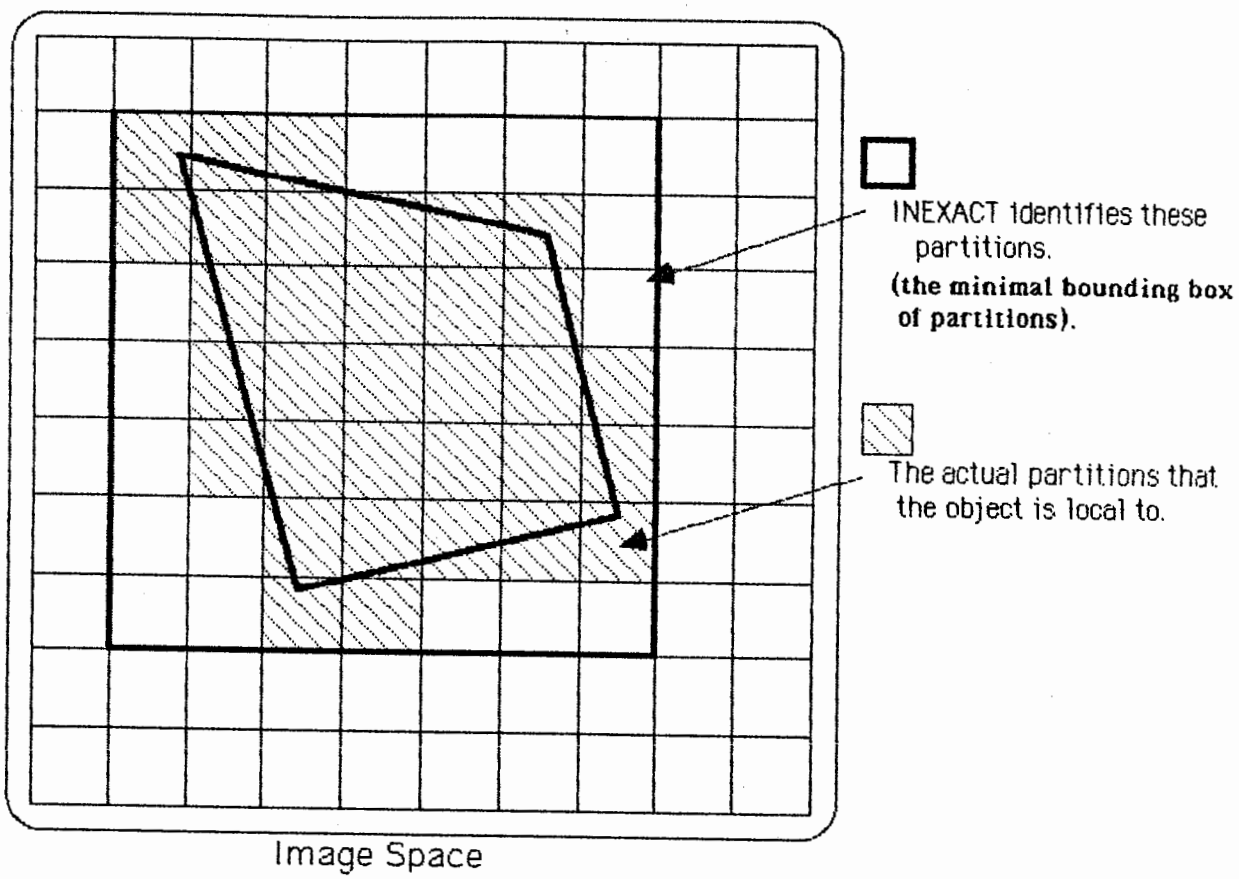


Figure 6.1 Conceptual View of the Image Space by the INEXACT Algorithm.

computational effort. For example, in the case of Figure 6.1, the actual number of partitions that the object local to is 33. But the INEXACT algorithm identifies $7^2=49$ partitions for the object. That is, 16 mistaken partitions are identified. However,

we have found out that, on the average, the waste of computation effort introduced by INEXACT is actually extremely small. In fact, we found that 99.54 percent of graphics objects generated in Chapter 5 can correctly be labeled by INEXACT. In many cases, INEXACT makes no error in identifying objects. Figure 6.2 depicts some of the cases in which INEXACT produces "exact" results.

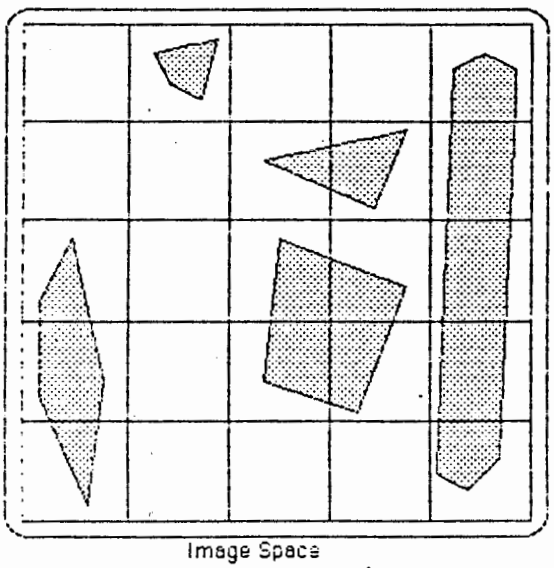


Figure 6.2 Various Classes of Objects that are Handled Properly by the INEXACT Algorithm.

6.3.4 INEXACT-Identifier

INEXACT-identifier is the device which is designed based on the INEXACT algorithm for a matrix of 4-by-4 partitions (see Figure 6.3). In Figure 6.3, the address of a partition is determined by the column and row number of the partition. The row number is counted from the lower row upward. The lowest row has a number 00_b and the highest row has a number 11_b. Similarly, the columns are numbered from left to right. The leftmost column is column 00_b and the rightmost colmun has an address

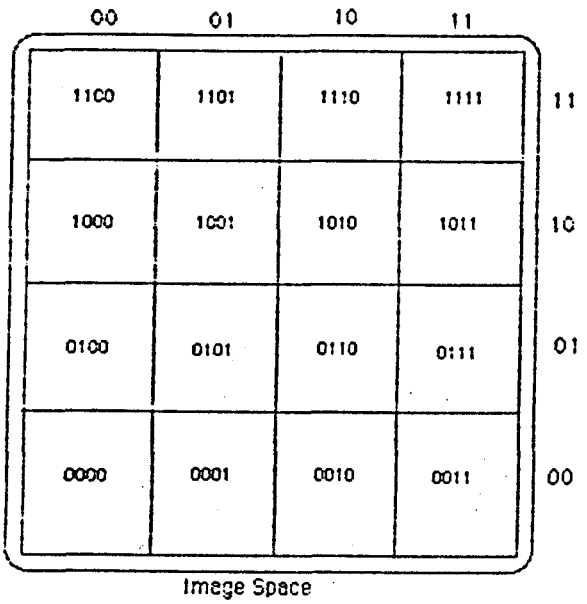


Figure 6.3 Labeling Scheme for an Image Space of 16 Partitions.

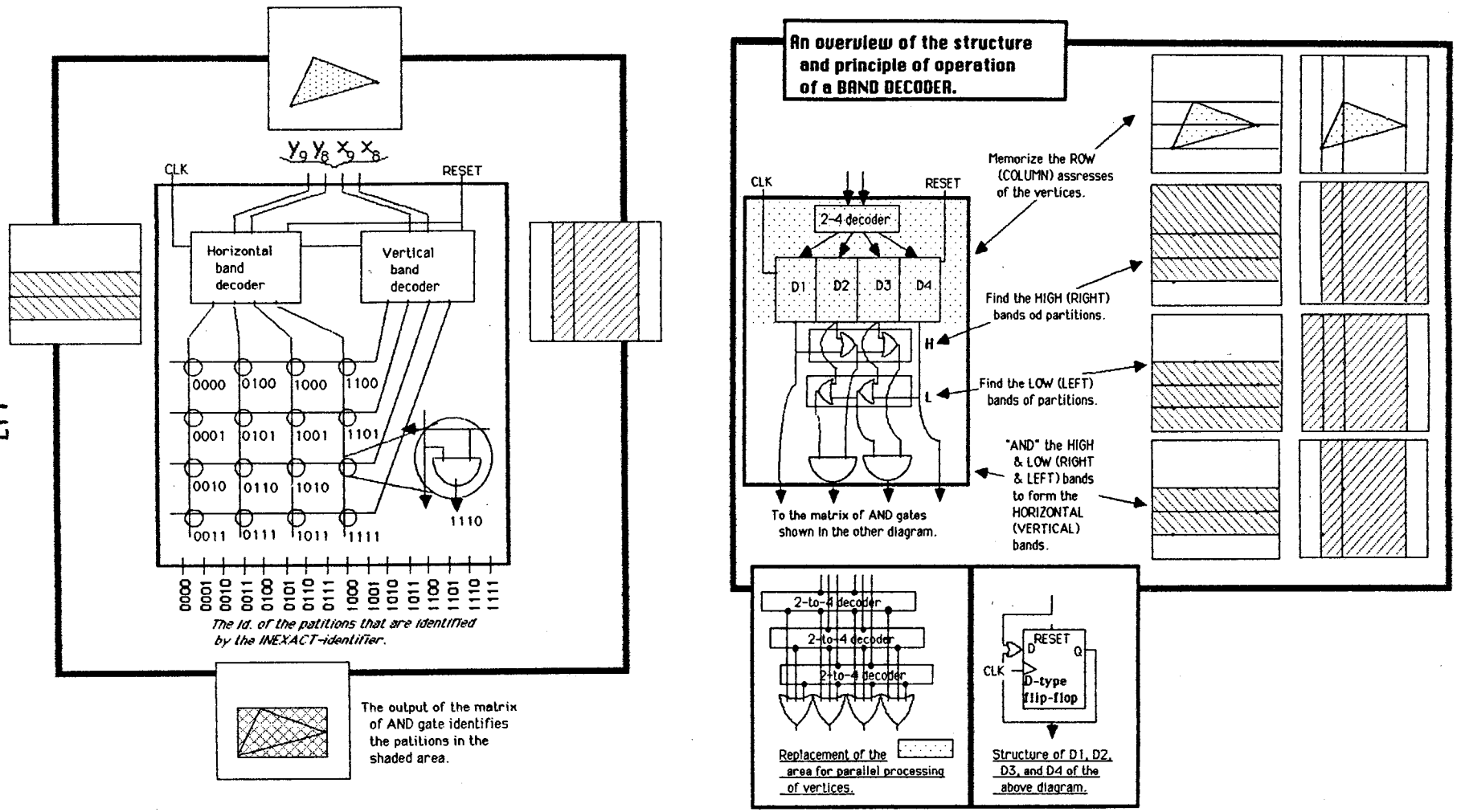
$_b$. A partition address is the concatenation of a row address and a column address. For example, the partition on row 10_b and in column 11_b has an address 1011_b.

Assume the resolution of the display is 1024x1024. Therefore, the screen coordinates are 10 bits long. For a point (x,y), the two most significant bits, x_9 and x_8, of the x coordinate and the two most significant bits, y_9 and y_8, of the y coordinate can be used to identify the partition enclosing the point. The partition that the point (x,y) is local to has an address $y_9y_8x_9x_8$. For example the point (0110111010_b,101010000_b), which is the point (442,687), is in the partition 1001_b.

In Figure 6.4, an overview of the INEXACT-identifier hardware is shown. When the vertices of an object are given, the identifier will determine the bounding box of partitions that the object is local to. The major input to the device are sets of 4-bit strings. Each 4-bit string is a concatenation of the two most significant bits of the X and Y coordinate of a vertex "$y_9y_8x_9x_8$". Recall that this is the partition address that the vertex is local to. Upon the entry of a new object (a new set of 4-bit strings), the RESET signal will be sent to the identifier to notify the start of a new identification process. Then, for each CLK cycle, a vertex (4-bit string) in the set is read and processed.

The INEXACT-identifier consists mainly of a *horizontal band decoder*, a *column band decoder*, and a number of AND gates. The

Figure 6.4 Internal Structure of the INEXACT-Identifier.



main function of the horizontal band decoder is to identify a band of rows which bound the object from the top and the bottom. While the column band decoder finds the band of columns which bound the object from the left and the right. Then the horizontal band and vertical band are ANDed in order to obtain the desired minimal rectangular sub-matrix of partitions (see Figure 6.4).

The horizontal band decoder takes the sub-string "y_9y_8" as input. This input is decoded by a 2-to-4 decoder. Then the decoded row address (d1,d2,d3,d4) is passed down to a *band register*, which consists mainly of 4 D-type flip-flops (D1,D2,D3,D4 where Di represents a flag for the ith row of partitions). These flip-flops are hooked up in such a way that Bi, for i=0,1,2,3, takes the "OR" of di and itself, that is "Bi OR di", as its D input. With this arrangement, Di will always memorize a 1 after a 1 is read from di (until the identifier is reset). That means the band register is able to keep a record of rows that have been addressed. This record is then read by a *low band generator* (L) and a *high band generator* (H).

The L network evaluates the logic function "equal or below the highest addressed row", for D1, D2, D3, and D4. In contrast, the H network identifies all the rows which are above or equal to the lowest addressed row. Therefore, when the results from L and H are ANDed, a band of rows between the highest and lowest addressed rows is obtained.

The column band decoder works in a similar manner as the row band decoder. These two devices operate simultaneously and hence a horizontal (row) and vertical (column) band can be obtained at the same time. Notice that the horizontal band bounds the object from the top and bottom, while the column band bounds the object from left to right. When the bands are ANDed, the minimal rectangle that encloses the object entirely is obtained. As shown in Figure 6.4, the AND gate network has an output signal for each of the 16 partitions. If a partition is within the horizontal band and the column band, the corresponding AND gate output will be 1.

The INEXACT-identifier is very fast. As illustrated, it just takes one clock cycle for each vertex of the object. Although the design given was for the 4-by-4 IP-scheme, it can easily be generalized to the k-by-k case. Also, the device can easily be modified to accommodate parallel-processing capability for object vertices. For example, if the objects are triangles, then the replacement of the 2-to-4 decoder and the band register, with the structure shown at the bottom of Figure 6.4, allows the INEXACT-identifier to process 3 vertices simultaneously.

6.4 Applying Regional-Rasterization to the Pixel-Planes System

In order to get a better feeling for the Regional-Rasterization technique, two examples are given to show how it can be installed into a multi-processor graphics display

to "cut-down" 1) the required number of processing units, or 2) the time required to render a scene. The system being considered is the Pixel-Planes, which is an n-1-1-m-system. That is, there is one (logical) pixel-processor for each pixel of the raster. Assume the resolution of the display is 1024-by-1024. Therefore it requires a matrix of $(1024)^2$ pixel-processors to implement the Pixel-Planes.

In the examples, the image space is to be divided into four quarters, namely I_1, I_2, I_3, and I_4 (see Section 3.3). Conceptually, each of the partitions can be viewed as an independent 256-by-256 sub-raster and thus can be handled by a 256-by-256 Pixel-Planes system. From now on, we will call such 256-by-256 matrices of Pixel-Planes structure used for the I_1, I_2, I_3, and I_4 as *$^1/_4$-Pixel-Planes*. The notion of the $^1/_4$-Pixel-Planes is depicted in Figure 6.5.

Again, the object space is divided into four groups: O_1, O_2, O_3, and O_4. The O_i, where i=1, 2, 3, or 4, contains only the graphics objects that are local to I_i. Since the O_i's are (logically) independent sets, they can be rendered either sequentially or simultaneously. If the object groups are rendered sequentially, only one $^1/_4$-Pixel-Planes module is required. If the four object groups are processed in parallel, the total rendering time will be (almost) quartered.

After a brief overview of the Pixel-Planes concept, two examples are given to demonstrate how Regional-Rasterization can

be used to enhance the ordinary Pixel-Planes system to achieve improvement on hardware utilization or rendering-time.

6.4.1 Architecture and Operation Theory of Pixel-Planes

The fundamental operation of the Pixel-Planes system is the calculation, simultaneously at each pixel, of the function $F(x,y)=Ax+By+C$, where x and y are the coordinates of the pixel in the display space. Figure 6.5 is a (conceptual) diagram of the Pixel-Planes system. The system consists of an array of identical memory cells connected to a grid which carries data to the cells. Two serial multiplier trees appear at the top and left-hand side of the array. These multipliers accept data from the pre-processor and calculate, simultaneously for all values of x and y, values for the functions $Ax+C'$ and $By+C"$. Each memory cell contains an adder which calculates the sum of these two functions in order to generate $F(x,y)=Ax+By+C'+C"$, where $C'+C"=C$. A block diagram for the X-multiplier is shown in Figure 6.6. (The Y-multiplier is identical.) Data for the coefficients A and C' is input to the multiplier in bit-serial form. As shown, the multiplier calculates the expression $Ax+C'$ for all values of x in the range of the display. The Y-multiplier accepts bit-serial data for coefficients B and C" and generates the expression $By+C"$ (for all y) synchronously with the X-multiplier's $Ax+C'$.

Owing to the limited area of each integrated circuit, only a small fraction of the pixels in the display can be put on a

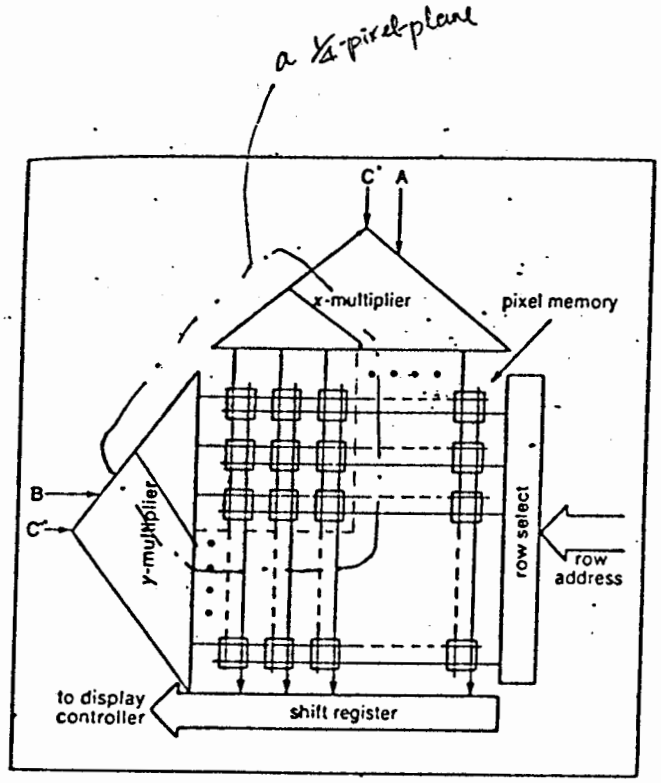


Figure 6.5 The Conceptual Structure of the Pixel-Planes System (from [FUCH81]).

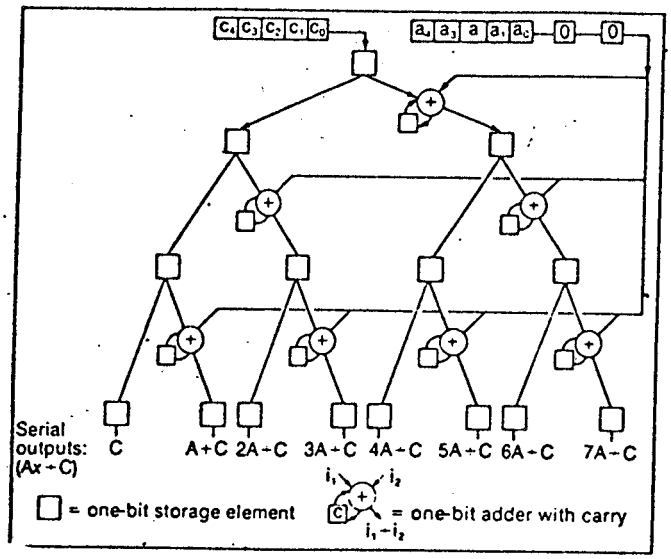


Figure 6.6 X-Multiplier Tree of the Pixel-Planes (from[FUCH81]).

single chip. Therefore it is necessary to break the multiplier trees into multiple chips. In the actual design of the Pixel-Planes circuit, a small subtree is implemented on each chip to cover the pixels on the chip. A 'supertree' structure is built on top of the subtrees to implement the tree levels above the subtrees [POUL85]. As shown in Figure 6.6, a supertree contains one multiply/accumulate stage for each level above the subtree. Registers in the supertree can be programmed to map a path through the full tree to the local subtree. With this arrangement, any Pixel-Planes chip can be easily relocated by

simply re-programming the registers of its supertree.

Figure 6.5 also shows a conceptual scheme for raster-scanning the memory cells. A row-select decoder driven by the display refresh controller selects a row of pixel memory cells, whose data is output in parallel to a shift-register. This shift register then allows the video data to be shifted out to the refresh controller.

Figure 6.7 shows the structure of an individual pixel memory cell. The memory cell contains four registers: Z, which contains

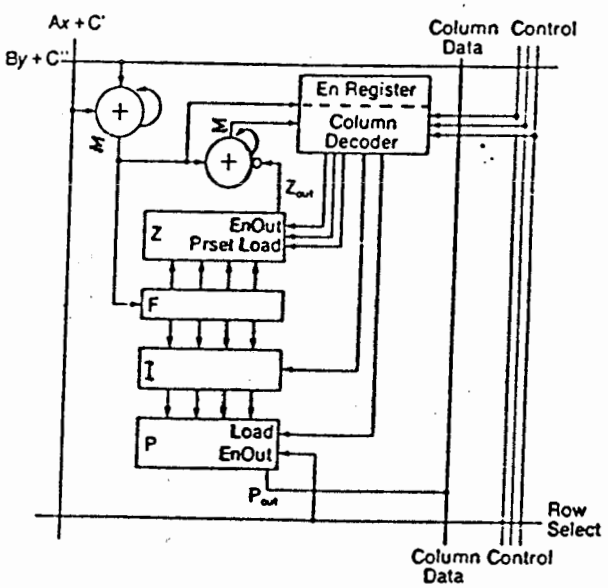


Figure 6.7 Internal Structure of a Pixel-Planes Memory Cell (from [FUCH81]).

the smallest z-value so far received at the pixel (portion of displayed object closest, and therefore most visible, to the viewer); F, which provides temporary storage for the function (F(x,y) output by the adder); I, the image register in which the results of the current polygon 'painting' operation are stored; and P, in which the intensity values for the previously constructed complete image are stored. The image stored in the P registers is the one currently being displayed. The P registers can be accessed by the display refresh circuitry independently of any processing operations in the pixel cells. In [POUL85], the processing circuitry and storage circuitry of a pixel are seperated. The processing circuitry (called the *ALU*) provides all the local processing power of the pixel cell.

6.4.2 Application I : Hardware Cut-down

Figure 6.8 depicts the Pixel-Planes system with Regional-Rasterization implemented. Few minor modifications to the system architecture can be found. Firstly, the pixel memory cell in the $^1/_4$-Pixel-Planes module is modified to accommodate 4 pixels. The modified pixel memory cell now has a P register file and a Z register file. The P and Z register files contain 4 registers each. The pixels stored in P_i make up the image displayed in I_i. Two output paths are used to output the pixels to the shift register for video refreshing. The left path is used by the P_1's and P_3's, while the right one is for P_2's and P_4's.

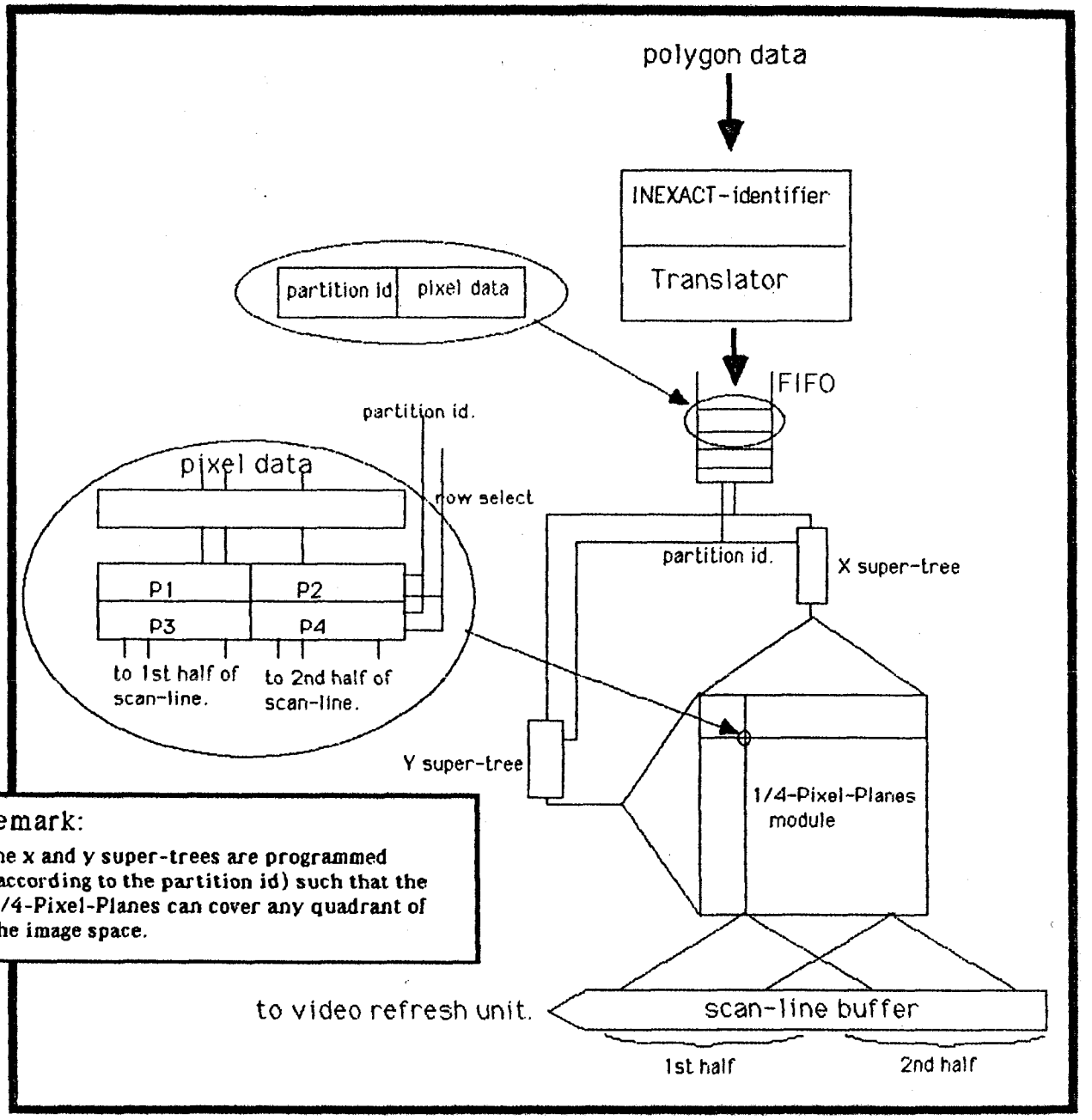


Figure 6.8 Application I: Hardware Cut-down.

Before a polygon (P) is passed down to the system, it is examined by an INEXACT-identifier which determines the image space partition(s) that the polygon is local to (see Section 3.3). The result of the local object identification procedure will be the tuple $P{:}i$ where i is the object space that P is local to. If P is local to more than one object space partition, copies of P, each *tagged* with a different i, are generated. For example, if a polygon P belongs to both O_1 and O_2, then $P{:}1$ and $P{:}2$ are produced.

The tagged polygon, $P{:}i$, is then passed down to a FIFO buffer which is the entrance to the $^1/_4$-Pixel-Planes. This FIFO is used to synchronize the polygon data input rate with the computation speed of the $^1/_4$-Pixel-Planes. When a $P{:}i$ leaves the FIFO, it enters into the ith "supertrees". The i tag will determine which supertree should feed the $^1/_4$-Pixel-Planes.

In the $^1/_4$-Pixel-Planes, the P is rendered as described earlier. The resulting image of the $P{:}i$ is stored in the P_i of the pixel memory cells.

For video refreshing, each scan-line is a concatenation of two sub-lines. The first half is made up of the pixels from the P_1 (P_3) registers of a row in the $^1/_4$-Pixel-Planes module. The second half is formed by the P_2 (P_4) pixels of the same row. Since pixels of the two halves of the scan line are output via two separate data paths (see Figure 11), the entire scan-line can be obtained with a single access to the P register file of

each pixel memory cell. With this arrangement, the video refreshing activity can be carried out with no unnecessary delay.

With the Pixel-Planes implemented in this way, the amount of hardware required for processing (not storing) pixel information is reduced. Theoretically, this implementation can save up to 75% of the amount of silicon area which was used previously for the multiplier trees and ALU of the pixel cells. This 'cut-down' of hardware is significant not only because it reduces the cost, but also because it may provides an alternative way to the Pixel-Planes design.

The major disadvantage of this implementation is the degradation in throughput. Notice that each object must be processed once for every partition that it is local to. Therefore, the modified Pixel-Planes is expected to be slower. However, if the average size of the graphics objects is small and the partition size is large, the chance of "multi-rendering" is very small. We expect the degradation in speed will be less than 10 percent. Moreover, there is also hardware overhead. It includes the extra facilities such as the local object identifier and the FIFO buffer, and extra supertree must be included. Therefore, this implementation of Regional-Rasterization might not be appropriate when the required resolution of the display is low. However, for high resolution applications this approach becomes more cost effective.

6.4.3 Application II: Scene Rendering Speed-up

A major weakness of the Pixel-Planes system is its limitation on the number of graphics objects that it can render in a fixed time interval. By enhancing the Pixel-Planes with Regional-Rasterization the scene rendering speed of the system can be quadrupled. An example of such an implementation is shown in Figure 6.9. In this implementation of Regional-Rasterization, all of the four $^1/_4$-Pixel-Planes modules are able to work on different (labeled) polygons simultaneously.

The sequential stream of polygon data can now be fed into the modified system at four times the previous rate. Each polygon (P) will be examined by the local object identifier which determines which FIFO it should be put in. The four FIFO's in Figure 6.9 correspond to the I_1, I_2, I_3, and I_4 regions. Similar to the previous case, duplicated P's may be produced if a polygon spans over more than one partition. Here duplicated P's are fed into more than one FIFO. The $^1/_4$-Pixel-Planes modules read data from their associated FIFO buffers and render polygons independently. Note that in this example, the internal architecture of a $^1/_4$-Pixel-Planes module is identical to an ordinary Pixel-Planes. The video refreshing activity can be carried out as usual.

With this implementation, the scene rendering speed can be potentially quadrupled. The actual speed improvement is determined by the evenness of the spatial distribution of the

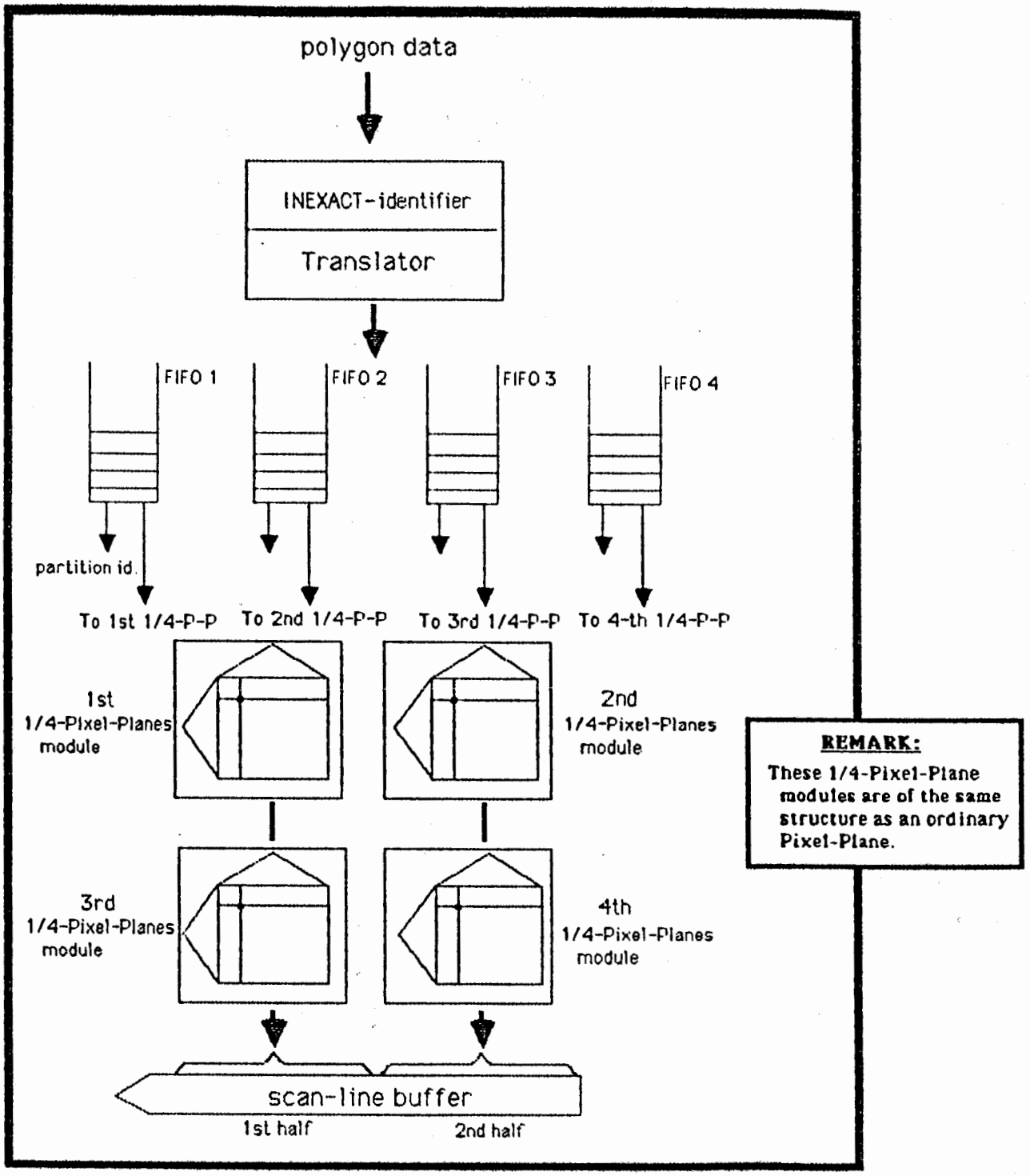


Figure 6.9 Application II: Scene Rendering Speed-Up.

graphics objects in the image space. Since all the 4 $^1/_4$-Pixel-Planes are working simultaneously, the time required to render all the objects is equal to the maximum time required by the $^1/_4$-Pixel-Planes to process their objects. If the distribution is very even, then there are approximately 1/4 of the total number of graphics objects local to each partition. In this case, the system is speeded up about 4 times.

In the worst case, all the objects are local to a single partition. There will not be any improvement in the processing time. However, it is guaranteed that the worst case performance of this implementation is the same as an ordinary Pixel-Planes system.

Owing to the limited area of each integrated circuit, only a small fraction of the pixels in the display can be put on a single chip. Therefore it is necessary to break the multiplier trees into multiple chips. In the actual design of the Pixel-Planes circuit, a small sub-tree is implemented on each chip to cover that pixels on the chip [POUL85]. A supertree structure is built on top of the subtrees to implement the tree levels above the subtrees. With this system, separate X and Y multiplier trees for each individual $^1/_4$-Pixel-Planes module are already implemented implicitly.

The major overhead of this implementation comes from the extra hardware for the local object identifier, and the 4 FIFO buffers. However, we estimate that this overhead is negligible

compared with the cost of the entire display. Therefore, advantages of Regional-Rasterization are even more amplified in this case.

6.5 Applying Regional-Rasterization to Fussel's Real-Time Scan-Conversion Engine

In the previous section, an implementation of the Regional-Rasterization technique into an n-1-1-m system was exemplified. In this section, the applicability of the technique to a 1-n-m-1 system is investigated. The system being considered is Fussel's real-time scan-conversion engine. An example is used to demonstrate how the Regional-Rasterization technique helps in reducing the hardware quantity for the engine. Speeding up Fussel's system is not of interest since it is already powerful enough to rasterize a scene in a single frame time. Therefore, the discussion of implementing Regional-Rasterization to improve system throughput is skipped.

In this section, the number of image space partitions is still assumed to be four. Thus, the object space is divided into four groups: O_1, O_2, O_3 and O_4.

6.5.1 Architecture and Operation Theory of Fussel's System

The overall organization of the system is depicted in Figure 6.10. In this system, the "frame buffer" consists of a collection of triangles rather than points. These triangles are

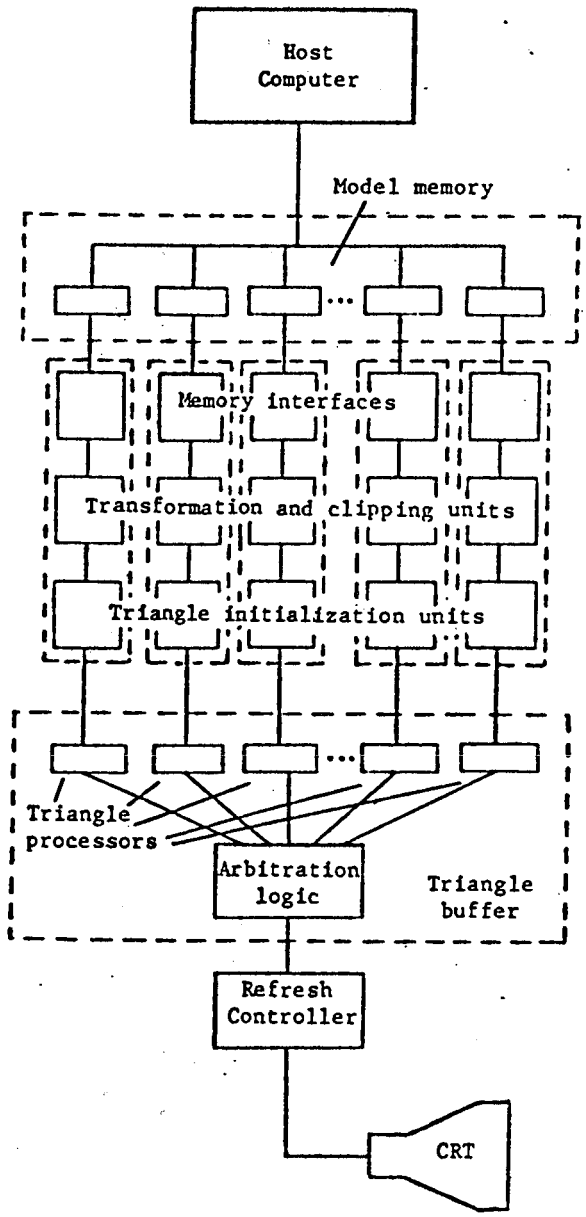


Figure 6.10 The Overall Structure of Fussel's System (from[FUSS82]).

stored in the so-called model memory. Segments of the model memory are hooked up to a number of individual preprocessing pipelines through which the triangles in the segments are transformed and clipped. The actual image rasterization task is performed in the so-called triangle buffer which contains a large number of triangle processors. Each triangle processor performs a scanout of the triangle it received from its preceding preprocessing pipeline, incrementally determining the color and z coordinate value for each pixel. The output of the triangle processors is then fed through the arbitration logic that determines which pixel is closest to the observer and uses the color of that pixel for video refreshing.

According to Fussel's terminology, a system slice consists of the following pipeline: a model memory slice, a model memory interface, a transformation and clipping unit, a triangle initialization unit, and a triangle buffer slice (see Figure 6.11). As shown in Figure 6.11, a triangle buffer slice is in fact an integration of an array of triangle processors and a comparator tree. Each triangle processor consists of 20 registers, with associated addition, comparison, and control logic, as shown in Figure 6.12. Each processor performs a simple scanout of the triangle, based on an identification of the initial left and right boundary edges and the third "alternate" edge by the triangle initialization module (see Figure 6.13).

The triangle processor monitors the address bus until a Y address which is equal to the value stored in the Y register is

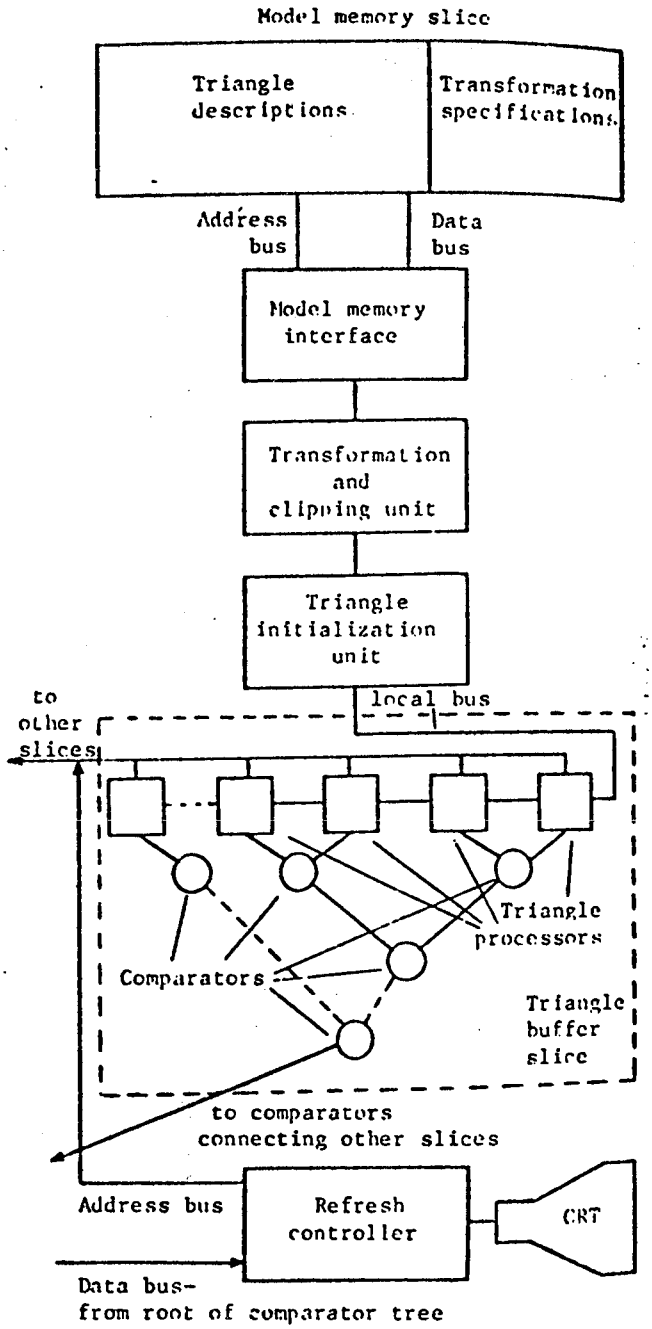


Figure 6.11 Internal Organization of a System Slice (from [FUSS82]).

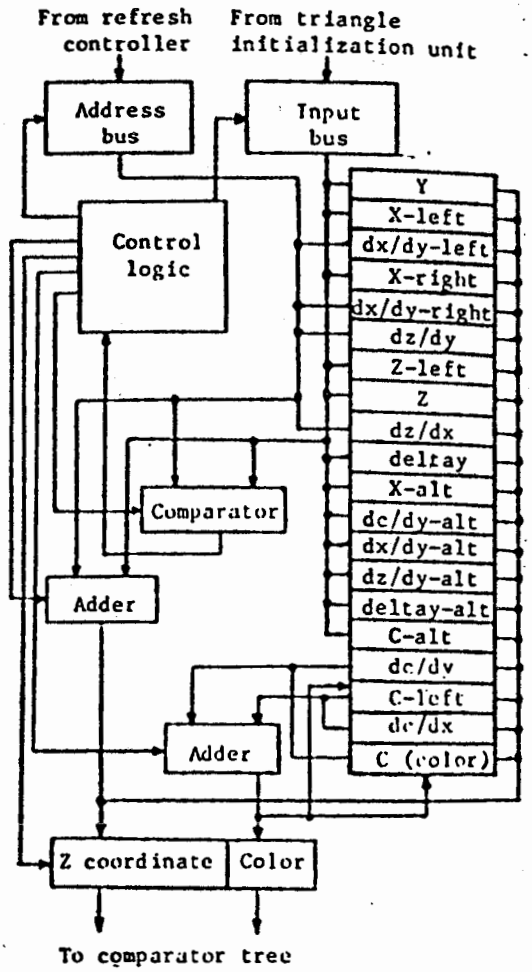


Figure 6.12 Block Diagram of a Triangle Processor (from [FUCH82]).

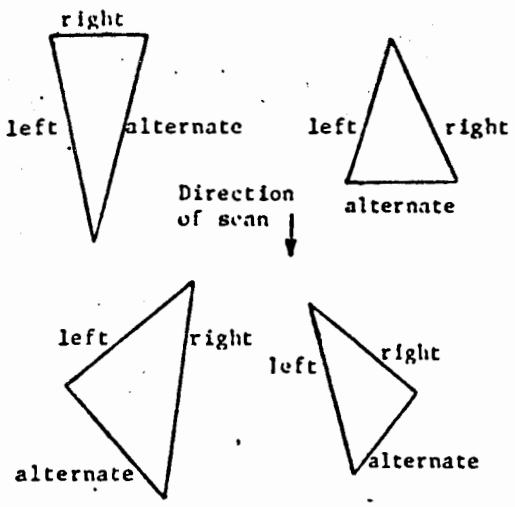


Figure 6.13 Assignments of Triangle Edges.

sent. It then monitors the X address until it reaches a value equal to X-left, at which time it outputs the initial color and Z coordinate. (Notice that the (X,Y) address is generated by the video refresh logic for sweeping out pixels in a raster from top to bottom and from left to right.) For each subsequent pixel, the color is incremented by dc/dx and the Z coordinate value by dz/dx and the new values are output. When the X address reaches X-right, the output is halted and the next scanline is prepared by adding dx/dy-left, dx/dy-right, dz/dy, and dc/dy to X-left, X-right, Z-left, and C-left respectively. The values of Z and C are then set equal to Z-left and C-left and the value of delta y

is decremented by 1. This process continues scan-line by scan-line until delta y reches 0. At this point, X-alt is compared to X-left and X-right to determine whether the left or the right edge should be replaced by the alternate edge. The value of dx/dy-alt, is then copied into dx/dy-left or dx/dy-right as appropriate. In the former case, the values of dz/dy-alt and dc/dy-alt are copied into dz/dy and dc/dy respectively. Delta y is set to the value of delta y-alt, and the processing continues until deltay once again reaches zero. At this point output is disabled, and the triangle processor enables itself for input of a new triangle.

6.5.2 Application I: Hardware Cut-down

In order to make use of the Regional-Rasterization technique, a simple architectural modification to the triangle processors is required. The modified architecture is depicted in Figure 6.14. In a modified triangle processor, there are 4 (i.e., the number of image space partitions) register files instead of 1. The internal structure of the register files is unaltered. That is, each of them still has 20 registers for the Y, X-left, dx/dy-left, ...etc. However, each of the four register files is now bound to a distinct partition in such a way that it is allowed to handle only those objects local to its corresponding partition. With this configuration, a triangle processor can hold up to 4 triangles, each from a distinct image space partition. The computational logic is shared among the 4 objects defined in the 4 register files. A set of multiplexing

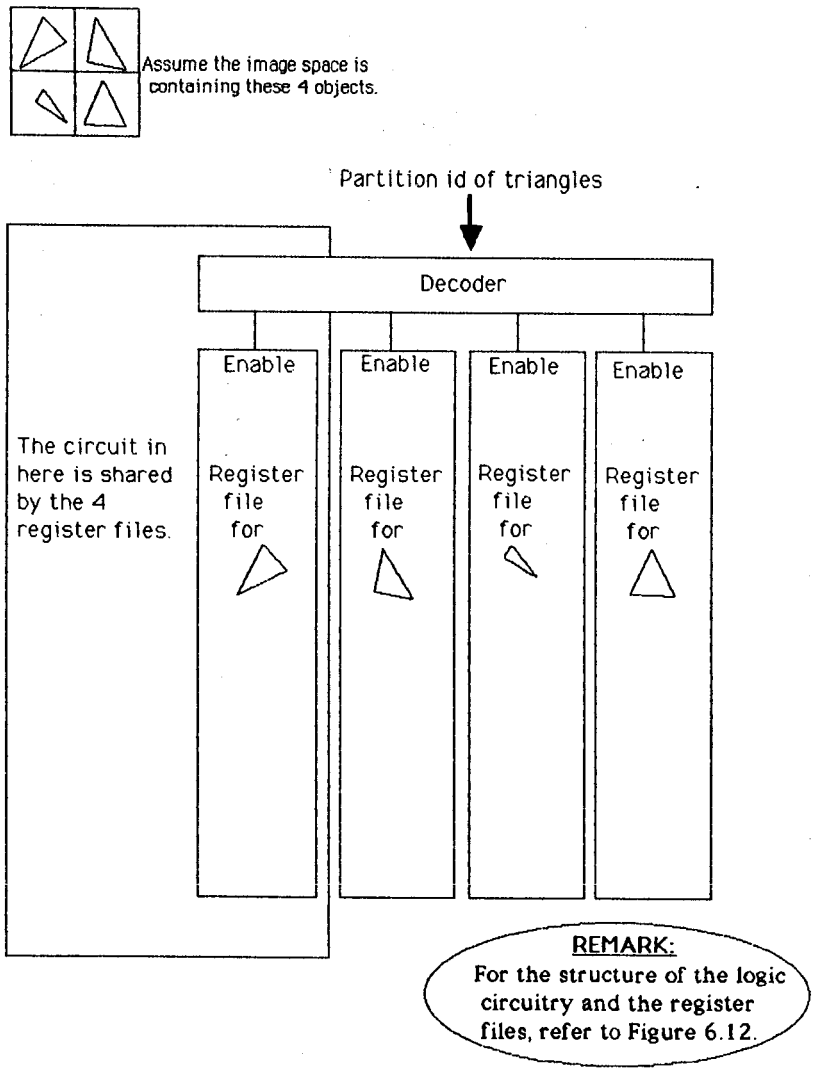


Figure 6.14 A Modified Triangle Processor.

logic which reads the X,Y address is employed to multiplex the computational logic among the register files.

The local object identification process is carried out preceding the triangle initialization step. Each triangle is

labeled by the local object identifier. These labeled triangles will then be handed down to the array of modified triangle processors (see Figure 6.15). As a triangle with label i is received by the processor array, the first processor with a free i register-file will be assigned to the triangle.

It should be obvious by now that the modified triangle processor is functionally equivalent to four ordinary triangle processors. As the scene is scanned-out, a triangle processor switches among 4 different triangles which are local to 4 different image space partitions. Therefore, in a video refresh cycle, a single modified triangle processor is able to rasterize up to 4 objects. Assuming that t slices are required in the original Fussel's system, with the modified triangle processors, approximately $t/4$ slices are needed. Hence, a cut down of up to 75% of the original amount of hardware is potentially achievable. The schematic of the Fussel's system with the Regional-Rasterization implemented is shown in Figure 6.15.

The overhead for this implementation includes the extra silicon area required to implement extra register files in the triangle processors. In addition, there must be a local object identifier for each system slice.

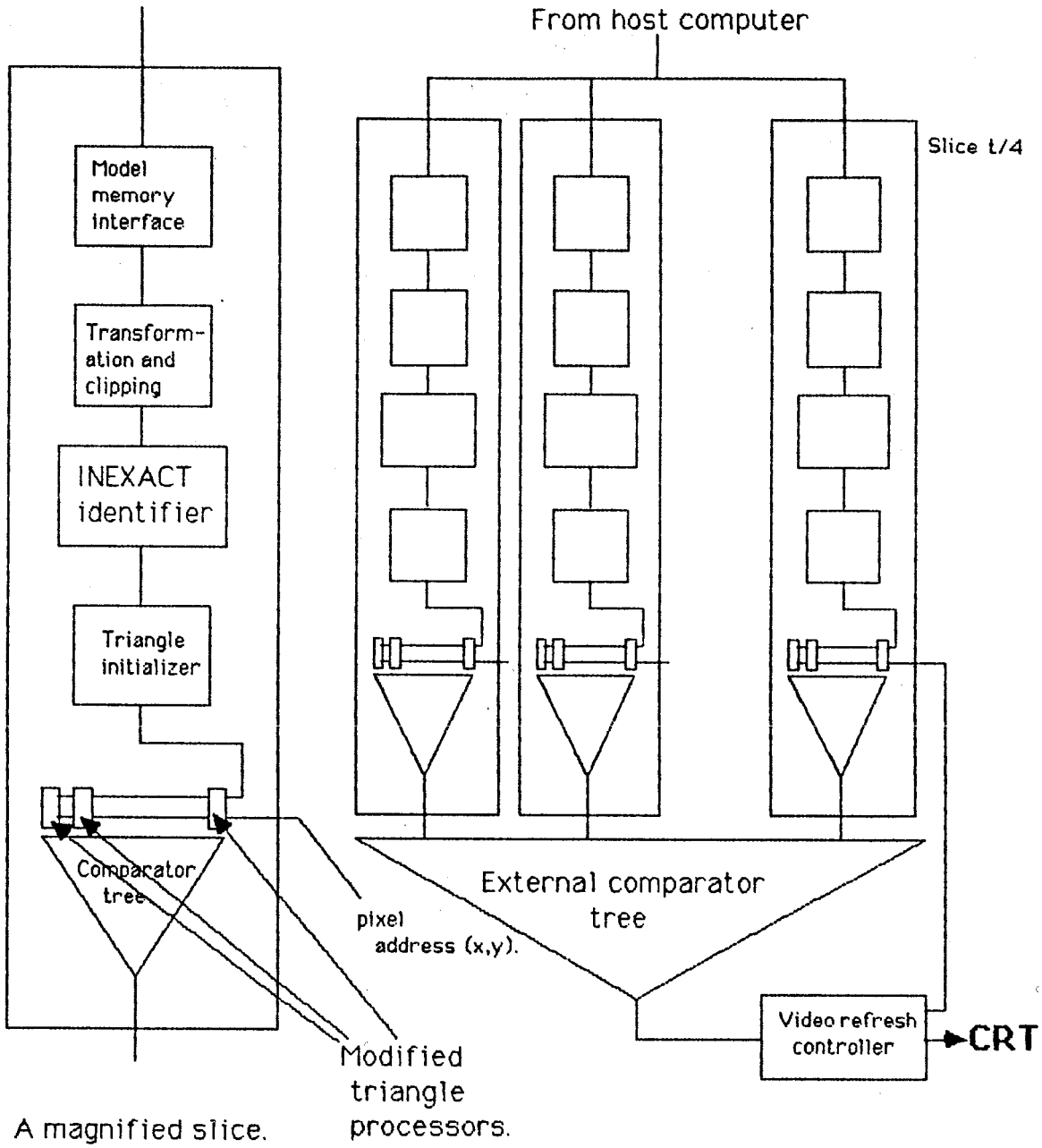


Figure 6.15 Application of the Regional-Rasterization technique to Fussel's system.

CHAPTER 7

CONCLUSION

A mathematical model was created to measure the expected number of memory cycles required to update a pattern for the scan-line mapping scheme and the symmetric mapping scheme for conventional frame buffered system. Three benchmark cases were used to demonstrate the applicability of this model. In the three benchmark cases it was shown that on the average the symmetric mapping scheme is faster than the scan-line mapping scheme. However, when the frame buffer consists of a small number of memory chips (e.g. ≤ 16), the speed advantage of the symmetric mapping scheme is so small that scan-line mapping scheme may be more cost effective mainly because of its simpler implementation. This discourages the use of conventional frame buffer architecture for applications requiring real-time image rasterization capability; because, even with symmetric mapping scheme implemented, performance will never be much superior to a conventional system.

Multi-processor architectures provide a promising solution to the real-time image rasterization problem. To achieve the desired computational power, the number of processing units is usually tremendous. These identical processing units are commonly structured in a loosely-coupled manner. Furthermore, perhaps more important, data separability is inherent in the scan-conversion problem, which means that littel or no

inter-processor communication is necessary [KAPL79].

However, when independent parallel computation tasks are distributed across a network of loosely-coupled processors, much of the coherence information about the graphics objects to be rendered will be lost. The performance of a scan-conversion algorithm can be improved substantially by taking advantage of some kind of coherence [NEWM79]. The regional-rasterization technique basically does this in a parallel-processing environment.

An important argument was given to support the idea of regional-rasterization: if the number of graphics objects is large, the average size of the graphics objects will be small. As a consequence, regional-rasterization is most effective in systems where the scene consists of many small objects. A major advantage of the regional-rasterization technique is its applicability to a variety of multi-processor architectures for graphics displays. Regional-rasterization requires only a small amount of extra hardware and introduces some processing overhead. However, comparatively speaking, these overheads are usually negligible.

In this thesis, only symmetrical partitioning of the image space has been considered. However, it is possible to have different forms of partition geometry and they might be valuable to some specific architectures (see Figure 7.1 for some).

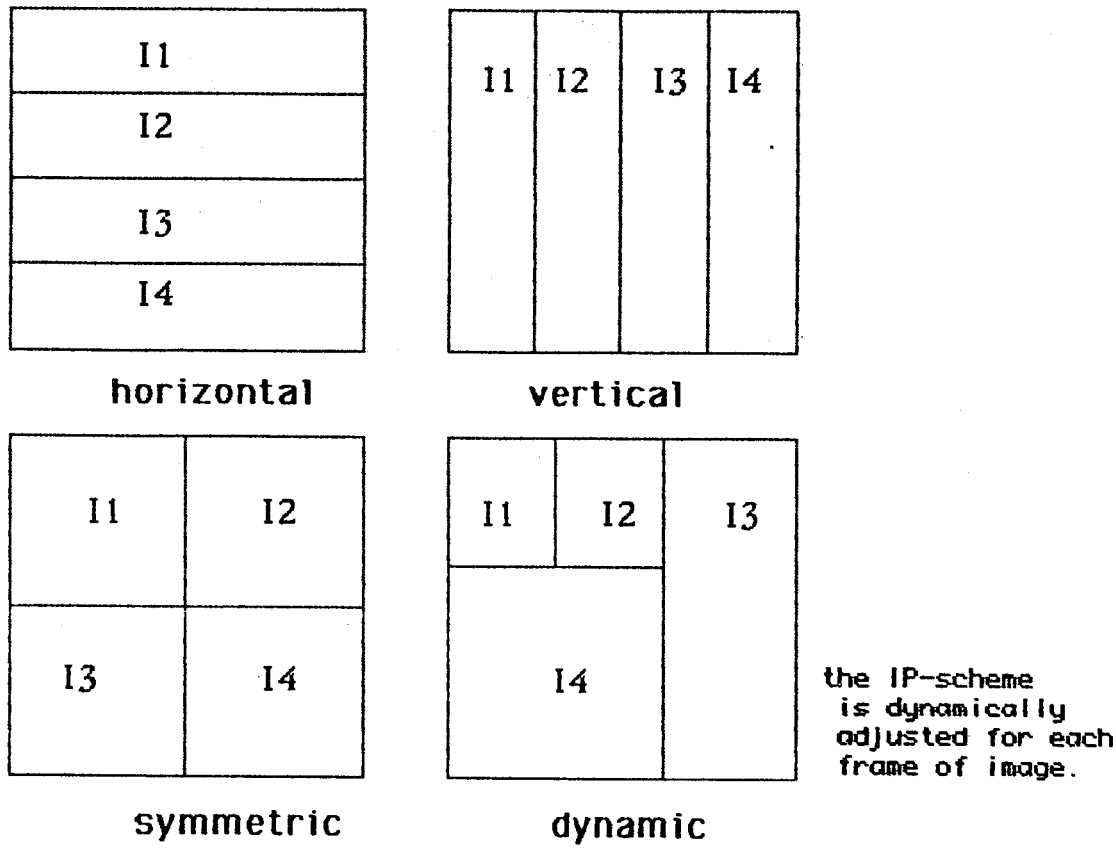


Figure 7.1 Various Image Space Partition Schemes.

Currently, we have only considered the case where image space is partitioned into static regions. However, the dynamic nature of computer graphics suggests that *dynamic partitioning* of the image space is an interesting refinement for the regional-rasterization technique. Some common ideas related to

the dynamic partitioning can be found in the paper by Tamminen [TAMM81] and Dippé [DIPP84]. With the ability to partition the image space dynamically, the regional-rasterization technique can self-adjust in such a way that each image space partition shares (almost) equal number of graphics objects. If this can be done, performance is optimized for each frame of image. However, we predict that adjusting the partition size dynamically is not easy to achieve due to the static configuration of the processor network commonly used in the image rasterizer.

REFERENCES

[AHUJ81] Ahuja, N., and B. J. Schachter, "Image Models," *Computing Surveys,* Vol. 13, No. 4, DEC. 1981, pp. 373-397.

[ABRA84] Abram, G. D., and H. Fuchs, "VLSI Architectures for Computer Graphics", *Proc. of NATO Adv. Study Inst. on Microarchitecture of VLSI Computer,* Sogesta-Urbino, Italy, July 9-20, 1984.

[BARN83] Barnhill, R. E., and W. Boehm, **Surfaces in Computer Aided Geometric Design**, North-Holland Publishing Company, 1983.

[CARL84] Carlbom, I. B., **High-Performance Graphics System Architecture: A Methodology for Design and Evaluation**, UMI Reasearch Press, Ann Arbor, Michigan, 1984.

[CHOR82] Chor, Benny, C. E. Leiserson, and R. L. Rivest, "An Application of Number Theory to the Organization of Raster-Graphics Memory," *M.I.T. VLSI Memo No.82-106,* JUN. 1982.

[CLAR80a] Clark, J. H., "A VLSI Geometry Processor for Graphics," *COMPUTER,* Vol. 13, No. 7, JUL. 1980, pp. 59-68.

[CLAR80b] Clark, J. H., and M. R. Hannah, "Distributed Processing in a High-Performance Smart Image Memory," *LAMBDA,* 4TH QUARTER 1980, pp. 40-45.

[CLAR82] Clark, J. H., "The Geometry Engine: A VLSI Geometry System for Graphics," *Computer Graphics,* Vol. 16, No. 3, JUL. 1982, pp. 127-133.

[COHE84] Cohen, C. L., "Full Wafer Memory for Color Displays has 1.5-MB Capacity," *Electronics,* JAN. 26, 1984, pp. 77-78.

[CRAI76] Crain, I. K., and R. E. Miles, "Monte Carlo Estimates of the Distributions of the Random Polygons Determined by Random Lines in a PLane," *Journal of Statistics, Computing, and Simulation,* Vol. 4, 1976, pp. 293-325.

[DARK80] Darke, F. I., "Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems," *Proceeding of SIGGRAPH 80,* 1980.

[DEME85a] Demetrescu, S., "High Speed Image Rasterization Using Scan Line Access Memories," *Chapel Hill Conference on VLSI*, 1985, pp. 95-102.

[DIPP84] Dippé, M., and J. Swensen, "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis," *Computer Graphics*, Vol. 18, No. 3, JUL.1984, pp. 149-158.

[ENGL86] England, N., "A Graphics System Architecture for Interactive-Specific Display Functions," *IEEE Computer Graphics and Applications*, Vol. 6, No. 1, JAN. 1986, pp. 60-70.

[FALL84] Fallin, J., "CHMOS DRAMs in Graphics Applications," *Solutions*, INTEL CO., MAY/JUNE 1984, pp. 20-27.

[FINK83] Finke, D. L., "Dynamic RAM Architectures for Graphics Applications," *AFIPS NCC Conference Proceedings*, 1983, pp. 479-485.

[FOLE82] Foley, J. D., and A. Van Dam, **Fundamentals of Interactive Computer Graphics**, Addison-Wesley, 1982.

[FRAN80] Frankiln, W. R., "A Linear Time Exact Hidden Surface Algorithm," *Computer Graphics*, Vol. 12, No. 3, MAR. 1980, pp. 117-123.

[FRAN81] Franklin, W. R., "An Exact Hidden Sphere Algorithm That Operates in Linear Time," *Computer Graphics and Image Processing*, Vol. 15, No. 2, APR. 1981, pp. 364-379.

[FUCH79] Fuchs, H., and B. W. Johnson, "An Expandable Multiprocessor Architecture for Video Graphics (Preliminary Report)," *Proc. of The 6th Annual Symposium on Computer Architecture*, 1979, pp. 58-67.

[FUCH81] Fuchs, H., and J. Poulton, "PIXEL-PLANES : A VLSI-Oriented Design for a Raster Graphics Engine," *VLSI DESIGN*, Vol. 2, No. 3, THIRD QUARTER, 1981, pp. 20-28.

[FUCH82b] Fuchs, H., J. Poulton, A. Paeth, and A. Bell, "Developing PIXEL-PLANES, A Smart Memory-Based Raster Graphics System," *Proc. of 1982 Conf. on Advanced Research in VLSI, M.I.T.*, 1982, pp. 137-146.

[FUCH85] Fuchs, H., J. Goldfeather, J. Hultquist, S. Spach, J. Austin, F. Brooks, Jr., J. Eyles, and J. Poulton, "Fast Sphere, Shadows, Textures, Transparencies, and Image Enhancements in PIXEL-PLANES," *ACM SIGGRAPH*, Vol. 19, No. 3, 1985, pp. 111-120.

[FUJI84] Fujimoto, A., C. G. Perrott, and K. Iwata, "A 3-D Graphics Display System with Depth Buffer and Pipline Processor," *IEEE Computer Graphics and Applications*, Vol. 4, No. 6, JUN. 1984, pp. 11-23.

[FUSS82] Fussel, D., and B. D. Rathi, "A VLSI-Oriented Architecture for Real-TIme Raster Display of Shaded Polygons (Preliminary Report)," *Proc. of Graphics Interface '82*, 1982, pp. 373-380.

[GHAR85] Gharachorloo, N., and C. Pottle, "SUPER BUFFER: A Systolic VLSI Graphics Engine for Real Time Raster Image Generation," *Chapel Hill Conference on VLSI*, 1985, pp. 285-305.

[GOLD86] Goldfeather J., and H. Fuchs, "Quardratic Surface Rendering on A Logic-Enchanced Frame-Buffer Memory," *IEEE Computer Graphics and Applications*, Vol. 6, No. 1, JAN. 1986, pp. 48-56.

[GRIF79] Griffiths, J. G., "Eliminating Hidden Edges in Line Drawings," *Computer Aided Design*, Vol. 11, No. 2, 1979, pp. 71-78.

[GUPT81a] Gupta, S., R. F. Sproull, and I. E. Sutherland, "A VLSI Architecture for Updating Raster-Scan Displays," *Computer Graphics*, Vol. 15, No. 3, MAR. 1981, pp. 333-340.

[IKED84] Ikedo, T., "High-Speed Techniques for 3-D Color Graphics Terminal," *IEEE Computer Graphics and Applications*, Vol. 4, No. 5, MAY 1984, pp. 46-58.

[JONG86] Jong, W. P., "An Efficient Memory System for Image Processing," *IEEE Trans. on Computers*, Vol. C-35, No. 7, JUL. 1986, pp. 669-674.

[KAPL79] Kaplan, M., and D. Greenberg, "Parallel Processing Techniques for Hidden Surface Removal," *Computer Graphics*, Vol. 13, 1979, pp. 300-307.

[KEDE84] Kedem, G., and J. L. Ellis, "The Raycasting System," *Proc. of 1984 IEEE Intl. Conf. on Computer Design*, OCT. 1984, pp. 533-538.

[KEDE85] Kedem, G., and S. W. Hammond, "THE POINT CLASSIFIER: A VLSI Processor for Displaying Complex Two Dimensional Objects," *Chapel Hill Conference on VLSI*, 1985, pp. 377-393.

[KLEN78] Klenbaum, D. G., and L. L. Kupper, **Applied Regression Analysis and Other Multivariable Methods**, Duxbury Press, North Scituate, Massachusetts, 1981.

[LEED80] Lee, D. T., and B. J. Schachter, "Two Algorithms for Constructing a Delaunay Triangulation," *International Journal of Computer and Information Science*, Vol. 9, No. 3, 1980, pp. 219-242.

[LOCA79] Locanthi, B., "Object Oriented Raster Displays," *CALTECH Conference on VLSI*, JAN. 1979, pp. 215-225.

[MATE60] Matern, B., "Spatial Variation," *Medd Statens Skogsforsknings Institut, Stockholm, Sweden*, 1960, pp. 1-144.

[MATI84] Matick, R., D. T. Ling, S. Gupta, and F. Dill, "All Point Addressable Raster Display Memory," *IBM Journal of Research and Development*, Vol. 28, No. 4, JUL. 1984, pp. 379-392.

[MCCM81] McClure, D. E., "Image Model in Pattern Theory," *in Image Modeling (edited by A. Rosenfield)*, ACADEMIC PRESS, 1981, pp. 259-275.

[MEAD80] Mead, C., and L. Conway, **Introduction to VLSI Systems**, Addison-Wesley, 1980.

[MILE69] Miles, R. E., "Poisson Flats in Euclidean Spaces Part I: A Finite Number of Random Uniform Flats," *Advanced Applied Probability*, Vol. 1, 1969, pp. 211-237.

[MILE71] Miles, R. E., "Poisson Flats in Euclidean Spaces Part II: Homogeneous Poisson Flats and the Complementary Theorem," *Advanced Applied Probability*, Vol. 3, 1971, pp. 1-43.

[MILE80] Miles, R. E., "A Survey of Geometrical Probability in the Plane, with Emphasis on Stochastic Image Modeling," *Computer Graphics and Image Processing*, Vol. 12, No. 1, JAN. 1980, pp. 1-24.

[MODE81] Modestino, J. W., R. W. Fries, and A. L. Vickers, "Stochastic Image Models Generated by Random Tessellations of the Plane," *in Image Modeling (edited by A. Rosenfeld)*, ACADEMIC PRESS, 1981, pp. 301-326.

[MORA73] Moran, P. A. P., "The Random Volume of Interpenetrating Spheres in Space," *Journal of Applied Probability*, Vol. 10, 1973, pp. 483-490.

[NEWM79] Newman, W. M., and R. F. Sproull, **Principles of Interactive Computer Graphics, 2nd ed.**, McGraw-Hill, New York, 1979.

[NICK84] Nickel, R., "The IRIS Workstation," *IEEE Computer Graphics and Applications*, Vol. 4, No. 8, AUB. 1984, pp. 30-34.

[NIIM84] Niimi, H., Y.Imai, and M. Murakami, "A Parallel Processor System for Three-Dimensional Color Graphics," *Computer Graphics*, Vol. 18, No. 3, JUL. 1984, pp. 67-76.

[OSTA84] Ostapko, D. L., "A Mapping And Memory Chip Hardware Which Provides Symmetric Reading/Writing of Horizontal And Vertical Lines," *IBM Journal of Reasearch and Development*, Vol. 28, No. 4, JUL. 1984, pp. 393-398.

[PARK80] Parke, F. I., "Simulation And Expected Performance Analysis of Multiple Processor Z-buffer Memory," *Proc. of ACM SIGGRAPH '80, published as Computer Graphics*, Vol. 14, No. 3, JUL. 1980, pp. 48-56.

[POUL85] Poulton, J., H. Fuchs, J. Austin, J. Eyles, J. Heinecke, C. Hsieh, J. Goldfeather, J. Hultquist, and S. Spach, "PIXEL PLANE: Building a VLSI-Based Graphic System," *Chapel Hill Conference on VLSI*, 1985, pp. 35-61.

[REEV84] Reeves, A. P., "SURVEY: Parallel Computer Architecture for Image Processing," *Computer Vision, Graphics, and Image Processing*, Vol. 25, No. 1, JAN. 1984, pp. 68-88.

[SANT71] Santalo', L. A., and I. Yanez, "Averages for Polygons Formed by Random Lines in Euclidean and Hyerbolic Planes," *Journal of Applied Probability*, Vol. 9, 1972, pp. 140-157.

[SANT76] Santalo', L. A., **Encyclopedia of Mathematics and its Applications Vol. 1 : Integral Geometry and Geometric Probability**, Addison-Wesley Publishing Company, Reading, Massachusetts, 1976.

[SATO85] Sato, H., M. Ishii, K. Sato, M. Ikesaka, H. Ishihata, M. Kakimoto, K. Hirota, and K. Inoue, "Fast Image Generation of Constructive Solid Geometry Using A Cellular Array Processor," *ACM SIGGRAPH*, Vol. 19, No. 3, JUL. 1985, pp. 95-102.

[SCHE79] Scheaffer, R. L., W. Hendenhall, and L. Ott, "Elementary Survey Sampling," *Duxbury Press, North Scituate, Mass.*, 1979.

[SCHM84] Schmidt, D. G., "Color Graphics Display for An Engineering Workstation," *Hewlett-Packard Journal*, Vol. 35, No. 5, MAY 1984, pp. 12-15.

[SPIE61] Spiegel, M. R., **Theory and Problems of Statistics**, McGraw-Hill Book Company, 1961.

[SPRO83] Sproull, R. F., I. E. Sutherland, A. Thompson, S. Gupta, and C. Minter, "The 8 By 8 Display," *ACM Trans. on Graphics*, Vol. 2, No. 1, JAN. 1983, pp. 381-411.

[STEP84] Stepoway, S. L., D. L. Wells, and G. R. Kane, "A Multiprocessor Architecture for Generating Fractial Surfaces," *IEEE Trans. on Computers*, Vol. C-33, No. 11, NOV. 1984, pp. 1041-1045.

[STFU83] Strothotte, T., and B. Funt, "Raster Display of a Rotating Object using Parallel Processing," *Computer Forum*, Vol. 2, 1983, pp. 209-217.

[SUTH74] Sutherland, I. E., R. F. Sproull, and R. A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms," *Computing Surveys*, Vol. 6, No. 1, MAR, 1974, pp. 1-55.

[SWIT65] Switser, P., "A Random Set Process in the Plane with a Markovian Property," *Ann. Math. Stat.*, Vol. 36, No. 6, DEC. 1965, pp. 1859-1863.

[SWIT67] Switzer, P., "Reconstructing Patterns from Sample Data," *Ann. Math. Stat.*, Vol. 38, 1967, pp. 138-154.

[SWIT69] Switzer, P., "Mapping a Goeographically Correlated Environment," *Tech. Rep. 145, Dep. Statistics, Standford Univ.*, STANDFORD, CALIF., 1969.

[TAMM81] Tamminen, M., "The EXCELL Method for Efficient Geometric Access to Data," *ACTA Polytechnica Scandinavica*, Vol. MA34, 1981.

[TENE81] Tenenbaum, J. M., M. A. Fischler, and H. G. Barrow, "Scene Modeling: A Structural Basis for Image Description," *in Image Modeling (edited by A. Rosenfeld)*, ACADEMIC PRESS, 1981, pp. 371-390.

[WALP80] Walpole, R. E., **Mathematical Statistics**, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1980.

[WEIN81] Weinberg, R., "Parallel Processing Image Sythesis And Anti-aliasing," *Computer Graphics*, Vol. 15, No. 3, AUG. 1981, pp. 325-332.

[WEIT84] WEITEK Co., **WTE6000 Tiling Engine**, System Specification, WEITEK Corporation, 1984.

[WHIT81] Whitted, T., "Hardware Enhanced 3-D Raster Display System," *Proc. of The 7th Man-Computer Communication Conference*, 1981, pp. 349-356.

[WHIT84] Whitton, M. C., "Memory Design for Raster Graphics Displays," *IEEE Computer Graphics & Applications*, Vol. 4, No. 3, MAR. 1984, pp. 48-65.