ESTI File Copy

ESD-TR-69-193

INCREMENTAL METHODS FOR COMPUTER GRAPHICS

D. Cohen

April 1969

DIRECTORATE OF PLANNING AND TECHNOLOGY
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Sponsored by: Advanced Research Projects Agency
Washington, D. C.

ARPA Order No. 952

(Prepared under Contract No. F19628-68-C-0379 by Harvard University, Cambridge, Massachusetts.)

AD0694550

ESD-TR-69-193

INCREMENTAL METHODS FOR COMPUTER GRAPHICS

D. Cohen

April 1969

DIRECTORATE OF PLANNING AND TECHNOLOGY
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Sponsored by: Advanced Research Projects Agency
Washington, D. C.

ARPA Order No. 952

(Prepared under Contract No. F19628-68-C-0379 by Harvard University, Cambridge, Massachusetts.)

FOREWORD

This report describes work accomplished under Contract F-19628-68-C-0379 from June 1968 through April 1969. This contract is concerned with research on computer graphics and computer networking. In particular it is directed to the development of new insights into the creation, analysis and presentation of information. This report is concerned with incremental methods for computer graphics.

The report is based on a thesis submitted April 30, 1969 by Mr. Dan Cohen to Harvard University, Division of Engineering and Applied Physics in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Professor Anthony G. Oettinger was the principal investigator for the contract. Dr. Lawrence G. Roberts was the ARPA director. Dr. Sylvia R. Mayer was the ARPA agent at Electronic Systems Division. Lt John McLean of Electronic Systems Division provided technical guidance.

This technical report has been reviewed and is approved.

Sylvia R Mayer

SYLVIA R. MAYER
Research Psychologist
Command Systems Division
Directorate of Planning & Technology

[illegible]

WILLIAM F. HEISLER, Colonel, USAF
Chief, Command Systems Division
Directorate of Planning & Technology

Lawrence G. Roberts

Lawrence G. Roberts
Special Assistant for
Information Sciences, ARPA

ABSTRACT

This report is concerned with incremental methods for computer graphics. The application of the incremental approach to some advanced problems in computer graphics is discussed and demonstrated. In Section II, the problem of fast curve generation and display is discussed. This section uses the mathematical approach to curves as a perspective projection of polynomial-curves in higher-dimensional spaces. Section III, also discusses a fast generation of curves, but using another mathematical approach, the linear-differences method, only two-dimensional curves are discussed. Section IV, shows how to make the Warnock algorithm, for hidden lines elimination, incremental. Section V, discusses the generation of half-tone images, in real-time. The technique suggested there requires a special-purpose hardware to be built. Section VI, discusses an incremental method for finding the intersection of given line-segments. The technique suggested there also requires a special hardware for implementation.

TABLE OF CONTENTS

LIST OF FIGURES

SECTION I

INTRODUCTION

The spirit of the incremental methods is to organize the steps of a computation so that each step can use information prepared by earlier steps. Such procedures eliminate redundant repetitive computation, and simplify each computation step.

In this thesis I will exhibit five examples of incremental methods. The examples shown here all apply to computer graphics. Computer graphics is particularly responsive to the application of incremental methods because a large amount of information must be processed quickly to produce a picture, while the basic operations to be performed on the data are relatively simple. Computer graphics is also particularly receptive to incremental techniques because conventional general-purpose computers do not do this job nearly as well as it can be done by incremental techniques. Moreover, the incremental approach is applicable to a wide variety of computing problems, and not to computer graphics only.

The five examples exhibited here are: linear-difference scheme for curve generation, perspective scheme for curve generation, the Warnock algorithm for hidden-line elimination, half-tone image production and a fast lines-intersecting technique.

The incremental method for perspective curves, which is described in Section II, was implemented in the three-dimensional-graphic-hardware project at Harvard, and was simulated on the PDP-1 computer. The incremental method for generating curves, which is described in Section III, is used by a PDP-1 system for drawing curves on a CRT, from which the pictures appended to that section were taken. The search-ordering strategy which makes the University of Utah hidden-line elimination algorithm incremental was incorporated into their hidden-line elimination system by its author, John Warnock. This approach is described in Section IV. A variation of the incremental method for half-tone image production, which is described in Section V, was implemented in hardware by the General Electric Company.

SECTION II

DISPLAY OF CURVES

(II. 1) Introduction

The ability to display straight lines on a CRT is a standard feature of these devices. However, many applications require curvilinear display. Typical schematic drawings like block-diagrams and logic-designs can be drawn with straight lines only, but attempts to display real world objects like machined components require curve display since the world is not rectilinear.

Any curved segment can be displayed accurately up to the scope resolution by displaying a set of points close enough together on the curve. Curve display by separate calculation and separate storage of each point is not feasible for many applications which are restricted by real time requirements or by storage limitation. A similar difficulty, existing in the past for straight lines, has been resolved by the introduction of line-generators which generate all the points of a line segment[1] given its end points. Carrying this approach forward to curves leads to a "curve-generator", based upon a small number of parameters, which can generate a curve segment in real time.

Two incremental curve generators are suggested in this work. One is based on a three-dimensional perspective approach (Section II) and the other on a two-dimensional approach (Section III). Both generate points on the curve so that these points may then be connected by a line generator.

Each of these curve generators can generate a family of curves. The two families are not the same, but both include all the conics. A comparison of these generators is made in (II. 6). The mathematical justification of our methods, and of the assumptions used in this section can be found in (II. 7).

---

[1] up to the scope resolution

(II. 2) On 2D Curves (Perspective Approach)

As the displaying screen is a 2D surface, 2D curves are the core of any curve representation. The ability to specify and display curves in 2D is the key to representing curves in any space. In (II. 7. 1), we prove that any conic segment in 2D is a perspective image of the canonic-parabola in 3D[1], or of its images under linear transformations. Therefore the problem of specifying a conic segment in 2D is equivalent to the problem of finding the linear transformation of the canonic-parabola which transforms it into a 3D curve whose 3D perspective projection is the desired conic. As any linear transformation can be realized by a matrix multiplication, the problem of specifying a conic segment is replaced by the problem of finding the matrix of the linear transformation. This concept of perspective images was already discussed in the late 19th century[2], [3].

The family of conics includes ellipses and circles, hyperbolas, parabolas and straight lines as a degenerate case. Conics never intersect themselves and never have an inflection point, as shown in (II. 7. 1). If more general curves are required, one can use an "mth-degree-conic" which is defined to be the perspective projection of a 3D curve whose components are polynomials of degree m in some parameter t. "Conics" usually mean "2nd-degree-conics" but I will not distinguish them from the general case, in spite of the importance of second degree conics for many applications. Properties of the mth-degree-conics can be found in an unpublished memorandum by Professor S. A. Coons and in [4]. In general, the higher m is, the more general is the mth-degree-conic, but of course, more conditions are required to specify it uniquely. In order to simplify the notation hereafter, we will use $m = 3$. The changes required for other values of m are self-evident.

---

[1] The canonic parabola in 3D is the curve given by $(x, y, z) = (t^2, t, 1)$.

The explicit form of the mth-degree-conic (for m = 3) is:

$$x = x(t) = x_3 t^3 + x_2 t^2 + x_1 t + x_0$$

$$y = y(t) = y_3 t^3 + y_2 t^2 + y_1 t + y_0$$

$$z = z(t) = z_3 t^3 + z_2 t^2 + z_1 t + z_0$$

which can also be written as:

$$(x, y, z) = (t^3, t^2, t, 1) \begin{pmatrix} x_3 & y_3 & z_3 \\ x_2 & y_2 & z_2 \\ x_1 & y_1 & z_1 \\ x_0 & y_0 & z_0 \end{pmatrix} .$$

A computational implementation of this representation, for generating a sequence of points on the curve may consist of storing $\{t^3, t^2, t, 1\}$ and using a matrix multiplier[1] for getting the (x, y, z) values. This is very useful for devices which happen to have a matrix multiplier capable of performing the multiplication of a $[1\times(m+1)]$ vector by a $[(m+1)\times 3]$ matrix. The scheme can be improved by generating the sequence $\{t^3, t^2, t, 1\}$ instead of storing it. This sequence can be generated very fast incrementally. However generating the polynomials x(t), y(t) and z(t) can be done as fast as generating the $\{t^3, t^2, t, 1\}$ sequence. The curve generator should therefore generate successive values of x(t), y(t), z(t) and project these points on the scope plane by dividing x(t) and y(t) by z(t) to get the scope coordinates of the points[2].

---

[1]An $\ell\times k$ matrix multiplier is a device which can store an $\ell\times k$ matrix A and multiply it by any given $1\times\ell$ vector V, to find the $1\times k$ vector VA. This operation requires $\ell\times k$ multiplications and $(\ell-1)\times k$ additions. By using parallel processing, a matrix multiplier can execute the multiplication in ℓ multiply-times only. Such a 4x4 matrix multiplier is described in the final report of Harvard contract XG-2972.

[2]It is easy to see that if the viewing plane is z=1, and the projection center is the origin, then the point (x, y, z) is projected on the point $(\frac{x}{z}, \frac{y}{z}, 1)$ on the viewing plane.

The scope coordinates are given to a line generator which generates line segments between the curve points while the new points are generated. The incremental method for generating successive values of $\{x(t),\ y(t),\ z(t)\}$ is described in (II. 4) below.

(II. 3) On 3D Curves (Perspective Approach)

For many applications it is very important to specify curves which satisfy conditions in the 3D space. After meeting these conditions, the curves are projected into the 2D space for display. One can easily generalize the 2D methods to 3D by introducing one more component and treating a 3D curve as a perspective projection of some 4D curve into 3D space. As in the 2D case, specifying a curve is equivalent to finding some matrix. Because of the additional component, there are more unknowns than in 2D in the equations defining the matrix and more conditions are required to specify curves uniquely. The perspective transformation from the 4D space to the 3D space (for curve definition) and the transformation from the 3D space to the 2D space (for display) can be combined in one perspective transformation, in order to simplify the displaying process.

Consider the following example: let $\{x(t),\ y(t),\ z(t),\ w(t)\}$ be a curve in the 4D space. Its perspective projection into the 3D space is the following curve: $\left(\frac{x(t)}{w(t)}, \frac{y(t)}{w(t)}, \frac{z(t)}{w(t)}\right)$ whose 2D perspective projection is the following curve:

$$\left(\frac{x(t)}{w(t)} : \frac{z(t)}{w(t)}, \frac{y(t)}{w(t)} : \frac{z(t)}{w(t)}\right) = \left(\frac{x(t)}{z(t)}, \frac{y(t)}{z(t)}\right) .$$

Note that there is no need to evaluate w(t) at all. There is only need to evaluate $\{x(t),\ y(t),\ z(t)\}$ and to display $\{x(t)/z(t), y(t)/z(t)\}$, exactly as in the 2D case, since the 3D curve is displayed in 2D regardless of the dimension of the space in which it is defined.

(II. 4) The Incremental Method for Fast Generation of Polynomials

For each of the 3 components $\{x(t),\ y(t),\ z(t)\}$ we have to evaluate a polynomial of degree m, at some set of values of the parameter $\{t_0, t_1, t_2 \ldots t_N\}$. For simplicity we choose the range of t to be $[0, 1]$

and $t_i = \frac{i}{N}$, i.e., equally spaced values of t. The usual techniques of using multiplications to evaluate polynomials take too much time. The fast way to evaluate these polynomials is the finite differences scheme [4] which is an incremental method, that prepares information from each point to be used in the generation of the next point. This incremental method requires only m addition for evaluating a polynomial of degree m. These additions can be executed simultaneously, and thus need only one addition time. The implementation of this method as described here requires m adders for each polynomial. If there is only one adder for each polynomial, the method can be modified as shown below.

We have 3 polynomials x(t), y(t), and z(t) to evaluate. As the generating procedure is the same for all of them, we consider here one polynomial only, say x(t).

We define: $\delta = \frac{1}{N}$, the space between successive values of t. In the usual fashion we define the forward difference operator [5]: $\Delta f(s) = f(s+\delta) - f(s)$. If x_i is defined as $x_i = x(t_i) = x(i\delta)$ then $\Delta x_i = x_{i+1} - x_i$. We define further: $\Delta^{\ell} f(s) = [\Delta^{\ell-1} f(s)]$ and $\Delta^0 f(s) = f(s)$. Note that as is well known, $\Delta^m t^m = m!\,\delta^m$, and this is independent of t. From the definition of $\Delta^{\ell} f(s)$ it follows that

$$\Delta^{\ell} x_i = \Delta[\Delta^{\ell-1} x_i] = \Delta^{\ell-1} x_{i+1} - \Delta^{\ell-1} x_i \quad ,$$

and

$$\Delta^{\ell-1} x_{i+1} = \Delta^{\ell-1} x_i + \Delta^{\ell} x_i \; .$$

Write the last equation for $\ell = 1, 2, 3$ and get:

$$x_{i+1} = x_i + \Delta x_i \qquad (\ell=1)$$

$$\Delta x_{i+1} = \Delta x_i + \Delta^2 x_i \qquad (\ell=2)$$

$$\Delta^2 x_{i+1} = \Delta^2 x_i + \Delta^3 x_i \qquad (\ell=3) \; .$$

This can be written as:

$$\begin{pmatrix} x \\ \Delta x \\ \Delta^2 x \\ \Delta^3 x \end{pmatrix}_{i+1} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ \Delta x \\ \Delta^2 x \\ \Delta^3 x \end{pmatrix}_{i} \quad \text{or} \quad F_{i+1} = S F_i$$

where F_i is the Forward differences vector at $x = x_i$, and S is the above matrix.

We define the backward difference operator by $\nabla f(s) = f(s) - f(s-\delta)$. ∇^0 and ∇^ℓ are defined similarly to Δ^0 and Δ^ℓ. In general $\nabla^\ell f(s) = \Delta^\ell f(s-\ell\delta)$. It is easy to see that $\nabla^{\ell-1} x_{i+1} = \nabla^{\ell-1} x_i + \nabla^\ell x_{i+1}$.

Substituting $\ell = 1, 2, 3$ gives:

$$\begin{aligned} x_{i+1} &= x_i + \nabla x_{i+1} \\ \nabla x_{i+1} &= \nabla x_i + \nabla^2 x_{i+1} \\ \nabla^2 x_{i+1} &= \nabla^2 x_i + \nabla^3 x_{i+1} \end{aligned}$$

which implies:

$$\begin{aligned} \nabla^2 x_{i+1} &= \nabla^2 x_i + \nabla^3 x_{i+1} = \nabla^2 x_i + \nabla^3 x_i \\ \nabla x_{i+1} &= \nabla x_i + \nabla^2 x_{i+1} = \nabla x_i + \nabla^2 x_i + \nabla^3 x_i \\ x_{i+1} &= x_i + \nabla x_{i+1} = x_i + \nabla x_i + \nabla^2 x_i + \nabla^3 x_i \quad . \end{aligned}$$

This can be written as:

$$\begin{pmatrix} x \\ \nabla x \\ \nabla^2 x \\ \nabla^3 x \end{pmatrix}_{i+1} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ \nabla x \\ \nabla^2 x \\ \nabla^3 x \end{pmatrix}_{i} \quad \text{or} \quad B_{i+1} = T B_i$$

where B_i is the Backward differences vector at $x = x_i$ and T is the above matrix. We have in mind placing the current values of x_i, $\nabla x_i \ldots \nabla^m x_i$ (or the $\nabla^\ell x_i$) in a set of fast registers. The successive values in the x_i register are used for the display and the other registers are used for conveying information from each point to its descendants. The $\Delta^m x$ register does not have to be modified as t changes over its range, because $\Delta^m t^m = \nabla^m t^m = m!\delta^m$. Let us factorize S and T to indicate the computational procedure:

$$S = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = A_3A_2A_1$$

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = A_1A_2A_3$$

A_ℓ is the operation of adding the $\nabla^\ell x$ (or the $\Delta^\ell x$) register to the $\nabla^{\ell-1} x$ (or the $\Delta^{\ell-1} x$) register. S is equivalent to executing A_1, then A_2, then A_3. T is equivalent to the execution of the same operations, in reverse order. There are only three additions involved in either of these methods. However, there is a basic difference between them. In the forward difference scheme, each "new" value $\Delta^\ell x_{i+1}$ depends only on "old" values $\Delta^k x_i$, but in the backward difference scheme, each "new" value $\nabla^\ell x_{i+1}$ depends on the current values of the registers, "old" $\nabla^\ell x_i$ and "new" $\nabla^{\ell+1} x_{i+1}$. It is therefore possible to execute all the additions for the forward difference method scheme in parallel, consuming only one addition time for evaluation successive values of the polynomials. This however requires a great deal of hardware. The backward difference scheme is more economical for sequential computation, but consumes m addition-times. Schematic drawings for the two methods are shown in Figures II. 4. 1 and II. 4. 3.

The behavior graphs[1] of these systems are shown in Figures II. 4. 2 and II. 4. 4. The behavior graph of the forward difference scheme

[1]Behavior graphs are sort of "occurrence-graph" [18].

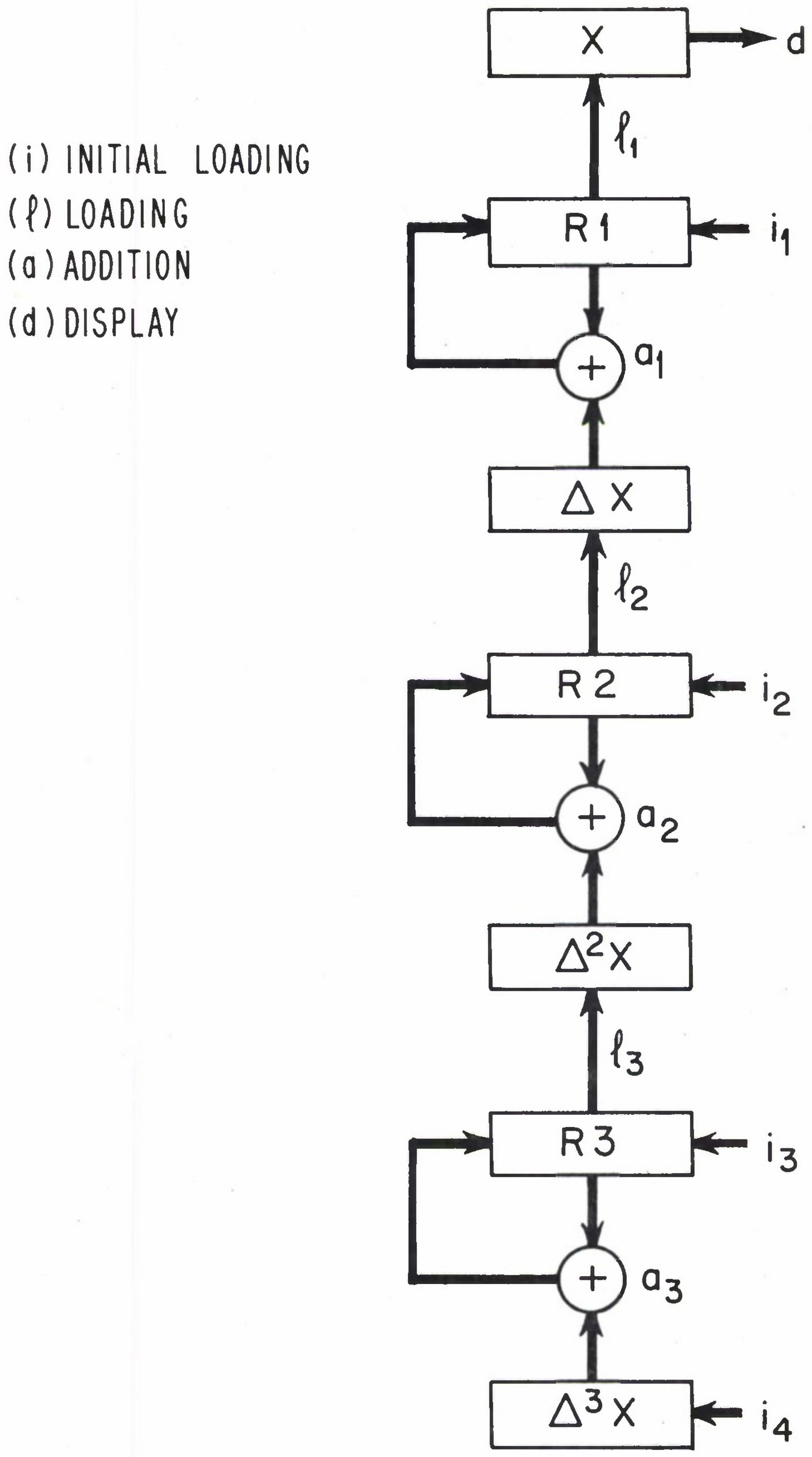


Figure II.4.1 : Schematic drawing of the hardware for the forward differences technique.

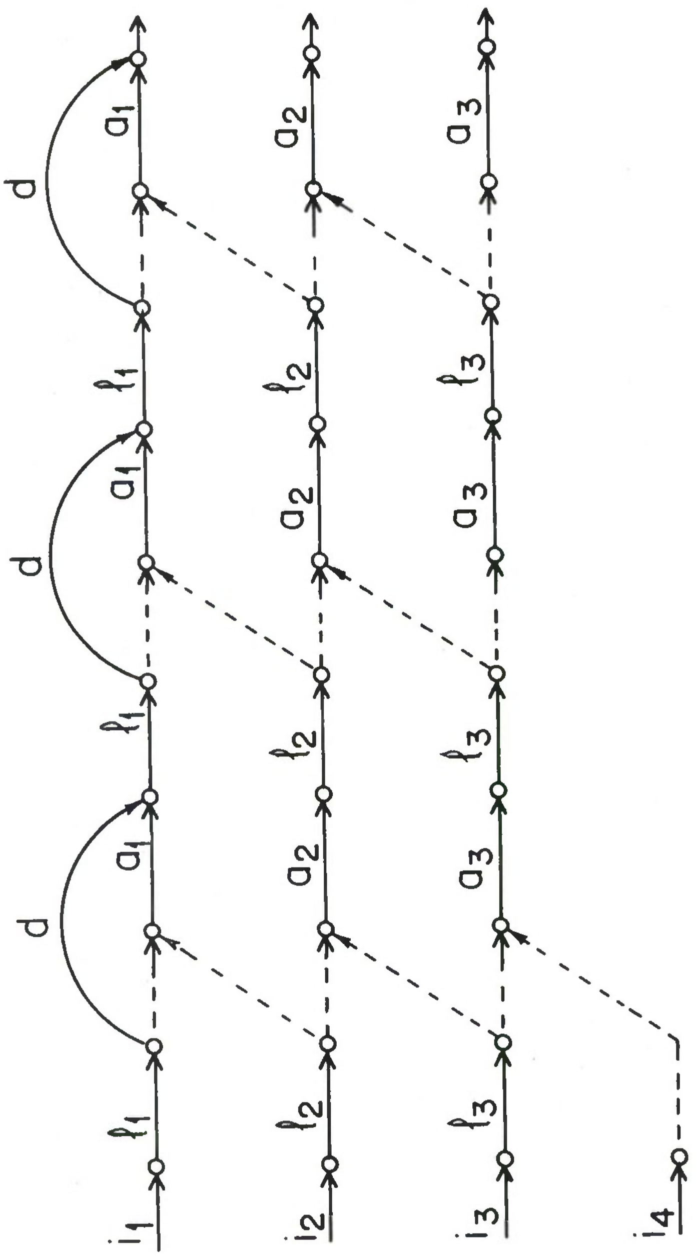


Figure II.4.2 : The behavior graph of the system in figure II.4.1

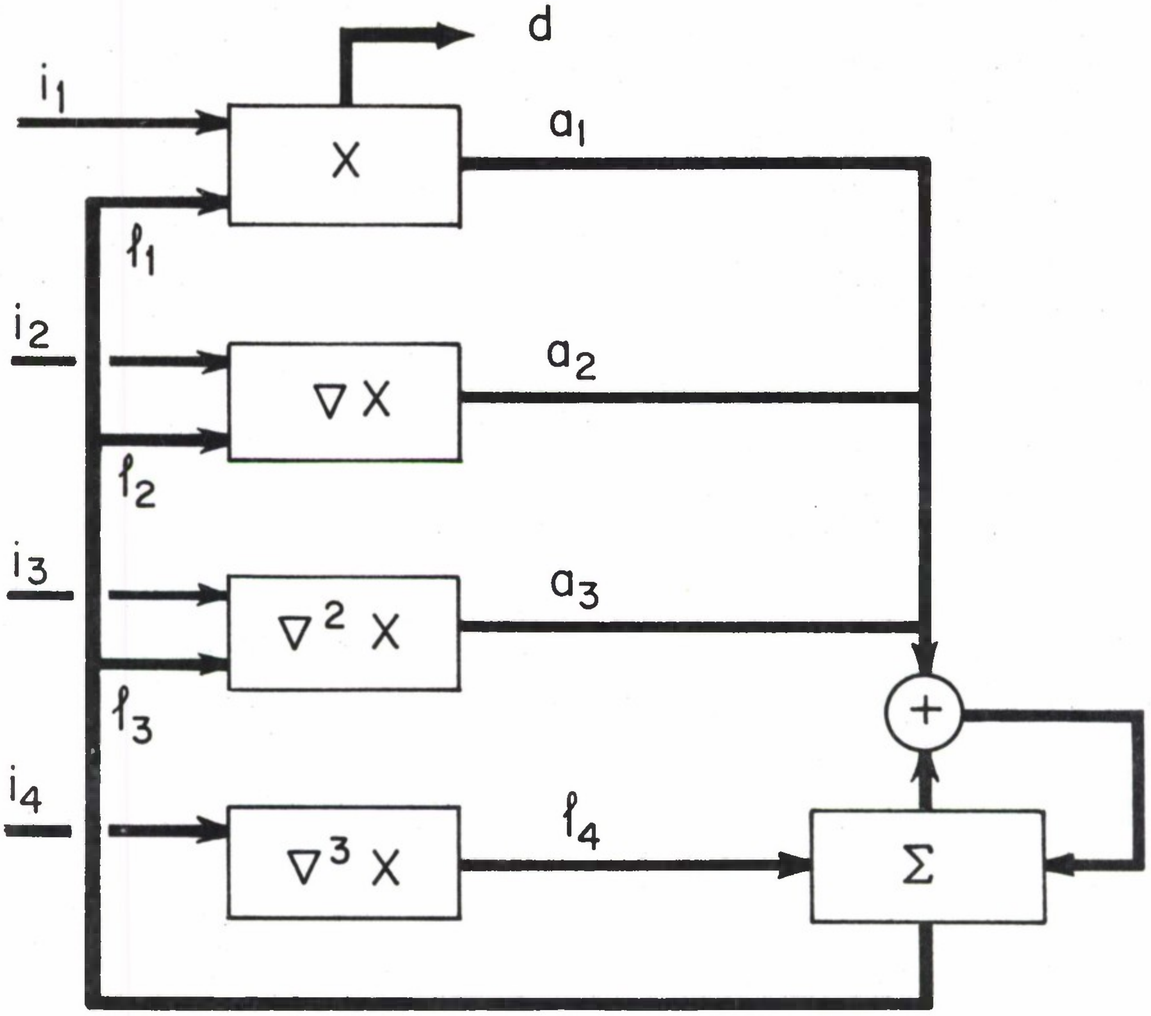


Figure II.4.3 : The hardware for the backward differences technique.

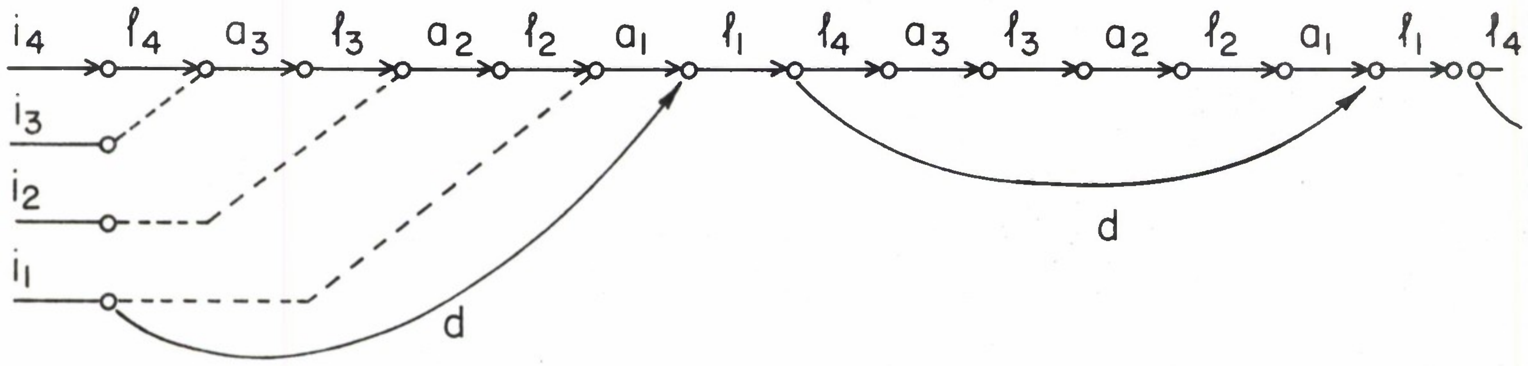


Figure II.4.4 : The behavior graph of the system in figure II.4.3

has 3 parallel paths which indicate the parallel processing. It is possible to introduce some parallelism in the backward difference scheme, but this requires more hardware than is needed for the forward difference scheme. However the critical operation is the division. If the division is slower than 3 additions (as it always is in digital systems) there is no sense in providing the extra hardware which is required by the forward difference method. In this case, both the forward and the backward difference method require on <u>divide-time</u>.

The forward difference scheme is initialized by loading $\Delta^{\ell} x_i(0)$ into the registers, and the backward difference scheme is initialized by loading $\nabla^{\ell} x_i(0)$.

In (II. 7. 2) we show how the initial differences $\{\nabla^{\ell} x_i(0)\}$ and $\{\Delta^{\ell} x_i(0)\}$ can be found.

(II. 5) <u>The Errors in the Incremental Computation of Polynomials</u>

The iteration process which is described in the preceding subsection may introduce some errors due to the finite precision of the machine. We cannot, in general, better the precision, but we can modify the iterations to minimize the errors where they are most important, at the end-points (e. g., for curve closure).

The source of the errors is not the iteration process itself (which is multiplication free) but the propagation of the errors in the initial values of the differences.

We can change the initial values of the differences in order to make the iterations end as close as possible to the prespecified end-point. We show now a method which converges to an end-point which is within $\frac{N}{2}$ machine resolution points from the given end-point, where N is the number of iterations.

Consider first the forward differences scheme. Define: $e = [1, 0, 0, 0]$ and $S = I + H$,

$$H = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad F(n) = \begin{pmatrix} x(n\delta) \\ \Delta\, x(n\delta) \\ \Delta^2 x(n\delta) \\ \Delta^3 x(n\delta) \end{pmatrix} .$$

We showed that $F(n) = S^n F(0)$ and $x(n\delta) = eF(n)$. The computed end-point is $x(N\delta) = eS^N F(0)$. Let $\hat{x}_e$ be the given end-point. Hence, the error introduced by the iterations is:

$$E = \hat{x}_e - x(N) = \hat{x}_e - [x(0) + N\Delta x(0) + \binom{N}{2} \Delta^2 x(0) + \binom{N}{3} \Delta^3 x(0)].$$

Dividing the error by $\binom{N}{3}$ is a correction for $\Delta^3 x(0)$. The remainder of this division divided by $\binom{N}{2}$ is a correction for $\Delta^2 x(0)$, and the remainder of the second division divided by N is a correction for $\Delta x(0)$. These 3 corrections reduce the magnitude of the error to be less than N. By changing $\Delta x(0)$ by one resolution unit, the magnitude of the error can be reduced not to exceed $\frac{N}{2}$.

This correction process can be programmed as follows:

(1) $\hat{x}_e - [x(0) + N\Delta x(0) + \binom{N}{2} \Delta^2 x(0) + \binom{N}{3} \Delta^3 x(0)] \longrightarrow E$

(2) $\dfrac{E}{\binom{N}{3}} \longrightarrow e_3$

(3) $\Delta^3 x(0) + e_3 \longrightarrow \Delta^3 x(0)$

(4) $\dfrac{E - \binom{N}{3} e_3}{\binom{N}{2}} \longrightarrow e_2$

(5) $\Delta^2 x(0) + e_2 \longrightarrow \Delta^2 x(0)$

(6) $\dfrac{E - \binom{N}{3} e_3 - \binom{N}{2} e_2}{N} \longrightarrow e_1$

(7) $\Delta x(0) + e_1 \longrightarrow \Delta x(0)$

(8) if $[E - \binom{N}{3}e_3 - \binom{N}{2}e_2 - Ne_1] > \frac{N}{2}$ then $\Delta x(0) + 1 \longrightarrow \Delta x(0)$

(9) if $[E - \binom{N}{3}e_3 - \binom{N}{2}e_2 - Ne_1] < -\frac{N}{2}$ then $\Delta x(0) - 1 \longrightarrow \Delta x(0)$

For the backward difference scheme a similar correcting method works by using:

$$x(n\delta) = eT^nB(0) \quad \text{where} \quad T = I + H + H^2 + H^3$$

and

$$T^n = I + nH + [n + \binom{n}{2}]H^2 + [n + 2\binom{n}{2} + \binom{n}{3}]H^3 \quad .$$

If we do not want to change $\Delta x(0)$, and we wish also to get the right value for $\Delta x(N)$ then another correcting scheme can be applied. Expand $F(N) = S^N F(0)$ and get:

$$x(N) = x(0) + N\Delta x(0) + \binom{N}{2}\Delta^2 x(0) + \binom{N}{3}\Delta^3 x(0)$$

$$\Delta x(N) = \Delta x(0) + N\Delta^2 x(0) + \binom{N}{2}\Delta^3 x(0)$$

$$\Delta^2 x(N) = \Delta^2 x(0) + N\Delta^3 x(0)$$

$$\Delta^3 x(N) = \Delta^3 x(0)$$

$x(0)$ and $x(N)$ are given by:

$$x(0) = d_x$$

$$x(N) = a_x + b_x + c_x + d_x \quad .$$

$\Delta x(0)$ and $\Delta x(N)$ can be found directly by:

$$\Delta x(0) = a_x\delta^3 + b_x\delta^2 + c_x\delta$$

$$\Delta x(N) = a_x(\delta^3 + 3\delta^2 + 3\delta) + b_x(\delta^2 + 2\delta) + c_x\delta \quad .$$

Using these values, $\Delta^2 x(0)$ and $\Delta^3 x(0)$ are found from the following equations:

$$\binom{N}{2} \Delta^2 x(0) + \binom{N}{3} \Delta^3 x(0) = x(N) - x(0) - N\Delta x(0)$$

$$N \ \Delta^2 x(0) + \binom{N}{2} \Delta^3 x(0) = \qquad \Delta x(n) - \quad \Delta x(0) \quad .$$

Using the values of $\Delta^2 x(0)$ and $\Delta^3 x(0)$ which satisfy (as close as possible) these equations reduce the errors in x(N) and Δx(N). For backward differences a similar method exists.

(II. 6) Comparison of the Perspective and the Linear Differences Methods for Curve Generation

The LDM (Linear Difference Method) is described in Section III. In this section I have described the PM (Perspective Method) for curve generation.

I compare the two methods, the LDM and the PM (for m=2) according to:

- complexity of points generation,
- variety of curves and speed of generation,
- complexity of the mathematics involved,
- sensitivity to errors.

The complexity of points generation

The PM requires 6 additions and 2 divisions per point. If enough hardware is available then all the additions can be carried out in parallel, and so can the divisions. Hence the time which is consumed by each point is between 1-add-time plus 1-divide-time and 6-add-times plus 2-divide-times.

The LDM requires 4 multiplications and 2 additions. If the hardware is available then the multiplications can be performed in parallel, and so can the additions. Hence the time which is consumed per point is between 1-add-time plus 1-multiply-time and 2-add-times plus 4-multiply-times.

If the curves are generated by software it is preferred to use the LDM. If hardware is used then the comparison depends on the speed and the cost of the components involved.

<u>Variety of curves and speed of generation</u>

The PM can generate only conic segments; ellipses, parabolas, hyperbolas and straight lines. It cannot generate complete ellipses and circles. The LDM can generate complete ellipses and arcs of hyperbolas and generalized parabolas ($y = x^{\alpha}$) which do not pass through the origin or through infinity. In addition the LDM can also generate straight lines, elliptic spirals, stars and various other shapes as described and illustrated in Section III.

The LDM generates the conics always at a "good-speed"[1]. The PM is not always able to generate a conic section at a "good-speed". For example, it is impossible for the PM to generate a circular arc at a uniform speed. Because of its good-speed-property the LDM needs fewer iterations to display a curve than the PM. (See the figures in III. 10).

<u>Complexity of the mathematics involved</u>

There is a simple method for finding the parametric matrix for any conic segment given by its end-points (see II. 7 and [6]). This method can be used for finding the matrices which together represent a complete ellipse. However, although it is relatively simple to find the parametric matrix, as required by the PM, it is not always easy to perform the shape invariant transformations [4] which are required to improve the generation speed.

There is no simple way to find the generating matrix for a given conic segment, as required by the LDM, because finding the matrix is equivalent to finding sines and cosines. However it is relatively very easy to find the generating matrix for a conic (not only a segment) from its implicit form or from its geometrical properties.

---

[1]A curve is said to be generated at a "good-speed" if the distance between successive points decreases when the radius of curvature decreases.

The simple manipulation of curves, such as rotation, translation, stretching and scaling do not have to be performed on each data point. In the PM all these operations can be performed on the parametric matrix. In the LDM only the stretching and rotation (in some cases) have to be performed on the matrix. The other operations do not need any operation on the matrix, and are carried out by the specification of the initial point (and initial difference), because the same generating matrix generates a whole family of curves.

Sensitivity to errors

Both, the PM and LDM may introduce roundoff errors. These roundoffs propagate during the iterations, and might result in a large error after N iterations. We show in this section that the error in evaluation each polynomial for the PM can be reduced to be less than $\frac{N}{2}$ in magnitude at the end points. We can do that because the iteration involve only additions which do not introduce new roundoff errors. The LDM uses multiplications which are rounded off, introducing possibly new truncation errors, of one machine-resolution unit, at each step. These roundoff errors are accumulated and multiplied by T^n. Hence the LDM is more sensitive to errors, then the PM. In some cases the LDM becomes so sensitive that it cannot be used. For example: A definition of a hyperbola by its implicit form and a point which is very close to an asymptote. The reason for this kind of sensitivity is that the same generating matrix generates a family of hyperbolas which all converge to the same asymptotes.

The PM does not have a 1-1 correspondence between curves and matrices, unlike the LDM. It is possible to change the representing matrix of a curve (even by splitting the curve into some sections) in order to make the iteration less sensitive. By this operation the sensitivities due to a small denominator, or to small numerators may be improved.

Unfortunately the generating matrices for the LDM are unique (up to the generation speed) and do not have any freedom which can be used for improving the sensitivity to errors.

(II. 7) Mathematical Justifications

The purpose of this subsection is to justify mathematically some results which are used in the section without proof.

(II. 7. 1) We will define a family of curves, $\sigma_{m,n}$, which consists of curves in R^n, whose components are polynomials of degree m in some parameter t. The perspective projection of $\sigma_{m,n}$, into R^{n-1} is called here $\pi_{m,n}$. We will prove the following results:

(a) If $S \in \pi_{2,3}$ then S is a conic segment;

(b) If S is a conic segment then $S \in \pi_{2,3}$;

(c) There exists $S \in \sigma_{2,3}$ such that for any $P \in \pi_{2,3}$ there exists a linear transformation T, such that P is the perspective projection of TS;

(d) Conics do not intersect themselves.

Let $\sigma_{m,n}$ be the family of curve segments in R^n, with each component being a polynomial in t, with degree not exceeding m. $S \in \sigma_{m,n}$ implies $S = \{s(t)\} = \{x_1(t), x_2(t)... x_n(t)\}$ where $x_i(t) = \sum_{j=0}^{m} a_{ij}t^j$. Let $\pi_{m,n}$ be the perspective projection of $\sigma_{m,n}$ in P^n, the perspective space obtained from R^n by dividing each component x_i by x_n. Note that P^n is isomorphic to the closed R^{n-1}.

If $S = \{s(t)\} = \{x_1(t), x_2(t)... x(t)\} \in \sigma_{m,n}$, then its projection in P^n is:

$$P = \{p(t)\} = \left\{ \frac{x_1(t)}{x_n(t)}, \frac{x_2(t)}{x_n(t)} \cdots \frac{x_{n-1}(t)}{x_n(t)} \right\} \in \pi_{m,n} \quad .$$

(a) We want to show that if $S \in \pi_{2,3}$ then S is a conic segment.

<u>Proof</u> (following L. G. Roberts [6]): Consider $\pi_{2,3}$, with the following notation: $x = x_1$, $y = x_2$ and $w = x_3$. Let

$$x = x(t) = x_2t^2 + x_1t + x_0$$
$$y = y(t) = y_2t^2 + y_1t + y_0$$
$$w = w(t) = w_2t^2 + w_1t + w_0 \quad .$$

In short, this can be written as:

$$p = p(t) = (x, y, w) = (t^2, t, 1) \begin{pmatrix} x_2 & y_2 & w_2 \\ x_1 & y_1 & w_1 \\ x_0 & y_0 & w_0 \end{pmatrix} = \tau A,$$

where $\tau = (t^2, t, 1)$ and A is the above matrix.

If A is a nonsingular matrix, define:

$$C = A^{-1} \begin{pmatrix} 0 & 0 & 1 \\ 0 & -2 & 0 \\ 1 & 0 & 0 \end{pmatrix} A^{*-1} = A^{-1} M A^{*-1}$$

then:

$$p C p^* = (\tau A) C (\tau A)^* = (\tau A)(A^{-1} M A^{*-1})(A^* \tau^*) = \tau \begin{pmatrix} 0 & 0 & 1 \\ 0 & -2 & 0 \\ 1 & 0 & 0 \end{pmatrix} \tau^* = 0$$

for all t. This proves that the arc τA is a conic segment. If A is singular, then there exists a non-zero vector V such that AV = 0, and $\tau AV = pV = (x, y, w) \begin{pmatrix} a \\ b \\ c \end{pmatrix} = ax + by + cw = 0$. Hence τA is a straight line, which is a degenerate conic.

(b) We proved above that if $P \in \pi_{2,3}$ then P is a conic segment (or a line segment, which is a degenerate case of a conic). The converse is more important: any conic segment belongs to $\pi_{2,3}$. We prove this as follows:

<u>Proof</u>: Let $vCv^* = 0$ be the implicit equation of the conic, and let v_0 and v_1 be the start and end points of the segment. Let v_T be the intersection of the two tangents to the conic through v_0 and v_1. Note that the tangents do not necessarily intersect in R^2, but they must intersect in P^3.

Consider the matrix

$$A = J V = \begin{pmatrix} 1 & -2 & 1 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ f v_T \\ v_0 \end{pmatrix}$$

where f is a scalar. It satisfies the following conditions:

(i) for $t = 1$, $\tau A = V_1 : (1,1,1)\,A = (1,1,1)\,J\,V = (1,0,0)\,V = v_1$

(ii) for $t = 0$, $\tau A = V_0 : (0,0,1)\,A = (0,0,1)\,J\,V = (0,0,1)\,V = v_0$.

(iii) τA is a conic arc because $(\tau A)\,C\,(\tau A)^* = 0$ for all t.

This is proved by:

$$V\,C\,V^* = \begin{pmatrix} v_1 \\ fv_T \\ v_0 \end{pmatrix} C \begin{pmatrix} v_1 \\ fv_T \\ v_0 \end{pmatrix}^* = \begin{pmatrix} v_1Cv_1^* & fv_1Cv_T^* & v_1Cv_0^* \\ fv_TCv_1^* & fv_TCv_T^* & fv_TCv_0^* \\ v_0Cv_1^* & fv_0Cv_T^* & v_0Cv_0^* \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & a \\ 0 & b & 0 \\ a & 0 & 0 \end{pmatrix}$$

where $a = v_1Cv_0^*$ and $b = f^2v_TCv_T^*$.

$v_1Cv_1^* = v_0Cv_0^* = 0$ because v_1 and v_0 are on the conic.

$v_TCv_1^* = v_1Cv_T^* = 0$ because v_T is on the tangent to the conic through v_1.

$v_TCv_0^* = v_0Cv_T^* = 0$ because of a similar reason.

With no loss of generality we can assume that C is a positive definite form. In this case, $a = v_1Cv_0^* = v_0Cv_1^* < 0$ and $b = f^2v_TCv_T^* > 0$. Next we consider the product $A\,C\,A^* = J\,V\,C\,V^*\,J^*$:

$$J(V\,C\,V^*)J^* = \begin{pmatrix} 1 & -2 & 1 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & a \\ 0 & b & 0 \\ a & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{pmatrix} = \begin{pmatrix} c & -c & a \\ -c & 4b & 0 \\ a & 0 & 0 \end{pmatrix}$$

where $c = 2a + 4b$. Hence

$$(\tau A)\,C\,(\tau A)^* = \tau(J\,V\,C\,V^*\,J^*)\tau^* = ct^4 - 2ct^3 + ct^2 = ct^2(1-t)^2 .$$

This expression vanishes for all values of t if c = 0. This condition can be satisfied by the proper choice of f, namely

$$f^2 = \frac{-v_1 C v_0^*}{2 v_T C v_T^*} \quad .$$

We showed in (iii) that τA is a part of the conic $VCV^* = 0$, and we showed in (i) and (ii) that the end points of τA are v_0 and v_1.

This completes the proof of the converse claim.

(c) <u>Corollary</u>: Any conic segment can be obtained from any non-degenerate conic segment by a linear transformation.

<u>Proof</u>: Let $v_1 = \tau A_1$ be a non-degenerate conic segment. Let $v_2 = \tau A_2$ be another segment. v_2 is obtained from v_1 by the linear transformation $T = A_1^{-1} A_2$, as $v_1 T = \tau A_1 (A_1^{-1} A_2) = \tau A_2 = v_2$. Note that if v_1 is the "canonical-parabola" $v = (t^2, t, 1)$ then $A_1 = I$, the unit matrix, and $T = A_2$, which means that v_2 is obtained from the canonic-parabola by using the representation matrix of v_2 as the transformation.

(II. 7. 2) The forward and the backward difference schemes need initialization by loading the values of $\Delta^{\ell} x_i(0)$ or $\nabla^{\ell} x_i(0)$ to the appropriate registers. We will find first $\Delta^{\ell} x_i(t)$ then we will find $\Delta^{\ell} x_i(0)$ by substituting t = 0. Later we will find $\nabla^{\ell} x_i(t)$ and substitute t = 0 to get the $\nabla^{\ell} x_i(0)$.

We use the fact that the ∇ and the Δ are linear operators in the evaluation of the differences of the polynomial x(t):

$$\Delta t^3 = 3t^2\delta + 3t\delta^2 + \delta^3$$

$$\Delta t^2 = 2t\delta + \delta^2$$

$$\Delta t = \delta$$

$$\Delta^2 t^3 = 6t\delta^2 + 6\delta^3$$

$$\Delta^2 t^2 = 2\delta^2$$

$$\Delta^3 t^3 = 6\delta^3 \quad .$$

Let $x(t) = a_3t^3 + a_2t^2 + a_1t + a_0$. Then:

$$\Delta x(t) = (a_1\delta + a_2\delta^2 + a_3\delta^3) + t(2a_2\delta + 3a_3\delta^2) + 3a_3\delta t^2$$

$$\Delta^2 x(t) = (2a_2\delta^2 + 6a_3\delta^3) + 6a_3\delta^2 t$$

$$\Delta^3 x(t) = 6a_3\delta^3 \quad .$$

Substitute $t = 0$ and get:

$$x(0) = a_0$$

$$\Delta x(0) = a_1\delta + a_2\delta^2 + a_3\delta^3$$

$$\Delta^2 x(0) = 2a_2\delta^2 + 6a_3\delta^3$$

$$\Delta^3 x(0) = 6a_3\delta^3 \quad .$$

Find the backward differences by:

$$\nabla t^3 = 3t^2\delta - 3t\delta^2 + \delta^3$$

$$\nabla t^2 = 2t\delta - \delta^2$$

$$\nabla t = \delta$$

$$\nabla^2 t^3 = 6t\delta^2 - 6\delta^3$$

$$\nabla^2 t^2 = 2\delta^2$$

$$\nabla^3 t^3 = 6t^3 \quad .$$

Again let $x(t) = a_3t^3 + a_2t^2 + a_1t + a_0$. Then:

$$\nabla x(t) = (a_1\delta - a_2\delta^2 + a_3\delta^3) + (2a_2\delta - 3a_3\delta^2) + 3a_3\delta t^2$$

$$\nabla^2 x(t) = (2a_2\delta^2 - 6a_3\delta^3) + 6a_3\delta^2 t$$

$$\nabla^3 x(t) = 6a_3\delta^3 \quad .$$

Substitute $t = 0$ and get:

$$
\begin{aligned}
x(0) &= a_0 \\
\nabla\, x(0) &= a_1\delta - a_2\delta^2 + a_3 \\
\nabla^2 x(0) &= 2a_2\delta^2 - 6a_3\delta^3 \\
\nabla^3 x(0) &= 6a_3\delta^3 \quad .
\end{aligned}
$$

<u>To summarize</u>: Let the curve be given by

$$[x(t), y(t), w(t)] = [t^3, t^2, t, 1] \begin{pmatrix} a_x & a_y & a_w \\ b_x & b_y & b_w \\ c_x & c_y & c_w \\ d_x & d_y & d_w \end{pmatrix} = \tau A \quad .$$

<u>The Forward Differences Scheme</u>:

We showed in (II. 4) that:

$$F(n) = \begin{pmatrix} x_n \\ \Delta\, x_n \\ \Delta^2 x_n \\ \Delta^3 x_n \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}^n \begin{pmatrix} x(0) \\ \Delta\, x(0) \\ \Delta^2 x(0) \\ \Delta^3 x(0) \end{pmatrix} = S^n F(0)$$

where $x_n = x(n\delta)$. Then we showed that:

$$F(0) = \begin{pmatrix} d_x \\ a_x\delta^3 + b_x\delta^2 + c_x\delta \\ 6a_x\delta^3 + 2b_x\delta^2 \\ 6a_x\delta^3 \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 6 & 2 & 0 & 0 \\ 6 & 0 & 0 & 0 \end{pmatrix}}_{Q} \underbrace{\begin{pmatrix} \delta^3 & & & \\ & \delta^2 & & \\ & & \delta & \\ & & & 1 \end{pmatrix}}_{D} \underbrace{\begin{pmatrix} a_x \\ b_x \\ c_x \\ d_x \end{pmatrix}}_{A} = QDA \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} .$$

Combine together and get:

$$[x(n\delta), y(n\delta), w(n\delta)] = [1, 0, 0, 0]S^n F(0) = [1, 0, 0, 0]S^n QDA$$

which may be verified by checking that:

$$[1, 0, 0, 0]S^n QD = [(n\delta)^3\ (n\delta)^2\ n\delta\ \ 1] \ .$$

The Backward Differences Scheme:

We showed that:

$$B(n) = \begin{pmatrix} x_n \\ \nabla x_n \\ \nabla^2 x_n \\ \nabla^3 x_n \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}^n \begin{pmatrix} x(0) \\ \nabla x(0) \\ \nabla^2 x(0) \\ \nabla^3 x(0) \end{pmatrix} = T^n B(0)$$

where $x_n = x(n\delta)$. Then we showed that

$$B(0) = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & -1 & 1 & 0 \\ -6 & 2 & 0 & 0 \\ 6 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta^3 & & & \\ & \delta^2 & & \\ & & \delta & \\ & & & 1 \end{pmatrix} \begin{pmatrix} a_x \\ b_x \\ c_x \\ d_x \end{pmatrix} = \hat{Q}DA \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} .$$

Combine together and get:

$$[x(n\delta)\ y(n\delta)\ w(n\delta)] = [1\ \ 0\ \ 0\ \ 0]T^n\hat{Q}DA$$

which may be verified by checking that:

$$[1\ \ 0\ \ 0\ \ 0]T^n\hat{Q}D = [(n\delta)^3\ (n\delta)^2\ (n\delta)\ 1] .$$

SECTION III

LINEAR DIFFERENCES CURVES

III. 1 Summary

We are interested in the family of curves of the form:

$$\Pi = \{P(s) \mid P(s) = T^s P(0) \; ; \quad 0 < s < \infty\}$$

where T is a 2x2 real matrix, P(0) is the initial point in 2D space, and s is a continuous variable.

These curves can be displayed by generating the sequence of points $\{P(n)\}$ where n is an integer, and connecting successive points by straight lines. The sequence $\{P(n)\}$ can be generated incrementally by using:

$$P(n+1) = T\,P(n) \quad .$$

The simplicity of this iteration makes it very attractive for digital systems involving either a special hardware or conventional programming.

There are several different definitions for these linear-differences-curves. The main ones are:

(a) $P(n+1) = TP(n)$ or $P(n) = T^n P(0)$.

(b) $\Delta P(n+1) = T_1 \Delta P(n)$ where $\Delta P(n) = P(n+1) - P(n)$.

(c) $\Delta P(n) = T_2 P(n)$.

(d) $\frac{dP(n)}{dn} = T_3 P(n)$.

All these definitions are equivalent. In (III. 2) below we prove this, and show the connection between T_1, T_2, T_3 and T. Although we use only the first of these definitions we want to point out that, for some applications, the other definitions might be more appropriate, (or even still other definitions).

In (III. 3) below we prove that any origin-centered-conic[1] may be generated by the above process. The proof is constructive and gives a

[1]An origin-centered-conic is defined by $ax^2 + 2bxy + cy^2 = d$.

"recipe" for getting the generating matrix T, for any conic specified by its implicit form. As corollaries from this theorem we get the generating matrices for circles and hyperbolas. Each of the generating matrices obtained by this method belongs to a one-parameter family of matrices, all of which generate the same curves but at different speeds. The free parameter can be used to control the generating speed, for example, by specifying P(1) on the curve; (with P(0) already specified).

In (III. 4) we show that if two curves are obtained from each other by a linear transformation, then their generating matrices are similar (in the usual mathematical sense). As a result we have a method for constructing the generating matrices for ellipses and hyperbolas according to their geometrical properties.

In (III. 5) we show that the condition $\det(T) = 1$ implies that T generates:

(a) an ellipse if $|\operatorname{trace}(T)| < 2$

(b) a straight line if $|\operatorname{trace}(T)| = 2$

(c) an hyperbola if $|\operatorname{trace}(T)| > 2$.

Next we are concerned with the "speed" of the conic generation, where the "speed" is defined as the distance between successive computer-generated points. In (III. 6) we show that unlike the perspective method, the LDM makes it possible to generate a circle with a uniform speed (i. e., equally spaced points) and hyperbolas with the right kind of speed (i. e., the speed decreases down as the radius of curvature decreases). As a corollary from the equally spaced circle generation it follows that it is possible to generate ellipses with the right kind of speed. Next we show the connection between the area of successive triangles A_n[1] and det(T). In particular we show that the areas of all the triangles of a conic are constant. This implies that the spacing on a conic is necessarily good.

---

[1] A_n is the triangle whose vertices are P(n), P(n+1) and the origin.

It is important to mention that the family of conics includes two kinds of straight lines, the ones which pass through the origin, and the ones which do not. In (III. 7) we discuss these lines.

In (III. 8) we discuss the curves which are generated by matrices with a non-unit determinant. We show that for any α, the curve $y = kx^{\alpha}$ can be generated by the iteration, as well as elliptic spirals.

Finally, in (III. 9) we discuss some programming aspects of coding the iteration.

(III. 2) <u>The Equivalence of the Various Definitions</u>

We will show that the 4 following definitions of linear-differences curves, are essentially equivalent, in the sense that they define the same family of curves.

(a) $P_n = TP_{n-1}$

(b) $\Delta P_n = T_1 \Delta P_{n-1}$

(c) $\Delta P_n = T_2 P_n$

(d) $\frac{dP_n}{dn} = T_3 P_n$.

We will show the equivalence of each definition to (a).

(III. 2. a) (a) implies (b), i. e., if a curve belongs to the family which is defined by (a), then it also belongs to the family of curves which is defined by (b).

$$P_{n+1} = TP_n ; \quad P_n = TP_{n-1}$$

$$\Delta P_n = P_{n+1} - P_n = TP_n - TP_{n-1} = T(P_n - P_{n-1}) = T\Delta P_{n-1} \quad \text{(QED)}$$

(b) does not imply (a), but:

$$P_{n+1} = \Delta P_n + P_n = \Delta P_n + \Delta P_{n-1} + P_{n-1} = \dots = \Delta P_n + \dots + \Delta P_1$$

$$+ \Delta P_0 + P_0 = T^n \Delta P_0 + T^{n-1} \Delta P_0 + \dots + \Delta P_0 + P_0$$

$$= (T^n + T^{n-1} + \dots I)\Delta P_0 + P_0 \ .$$

Assume that 1 is not an eigenvalue of T, then we can write:

$$\Delta P_0 = (T - I)\hat{P}_0 ,$$

then

$$P_{n+1} = (T^n + T^{n-1} + \ldots + I)(T - I)\hat{P}_0 + P_0 = T^{n+1}\hat{P}_0 + (P_0 - \hat{P}_0)$$

which shows that (b) defines the same curves as (a) but they might be off-centered by $E = P_0 - \hat{P}_0$. The reason for this possible displacement is that (a) requires only an initial P_0, but (b) requires initial P_0 and ΔP_0. If $\Delta P_0 = TP_0 - P_0 = (T - I)P_0$ then $E = 0$, and there is no center displacement.

If 1 is an eigenvalue of T, then T generates a straight line, which obviously can be generated by (a).

(III. 2. b) (a) implies (c):

$$\Delta P_n = P_{n+1} - P_n = (T - I)P_n = T_2 P_n \quad .$$

(c) implies (a):

$$P_{n+1} = P_n + \Delta P_n = P_n + T_2 P_n = (T_2 + I) P_n \quad .$$

(III. 2. c) (a) implies (d):

$$P(n) = T^n P(0)$$

$$\frac{dP(n)}{dn} = (\ell g\, T)T^n P(0) = (\ell g\, T)P(n) = T_3 P(n) \quad .$$

(d) implies (a):

$$\frac{dP(n)}{dn} = T_3 P(n)$$

$$P(n) = e^{nT_3}A \; ;$$

substitute n = 0 and get A = P(0),

$$P(n) = (e^{T_3})^n P(0) = T^n P(0) \quad .$$

Note that if T has non-positive eigenvalues then there does not exist a real matrix $T_3 = \ell g\, T$, and (a) does not imply (d).

For displaying an off-centered curve, one can generate the $\{P(n)\}$ using (a), and add some displacement to each point, or position the initial point $P(0)$, and generate the $\{\Delta P(n)\}$ using (b). The latter scheme saves the addition of the displacement to each point, and is, therefore, preferred if a special-purpose hardware is not available.

(III. 3) Obtaining the Generating Matrix for a Conic (Implicit Form)

Theorem: For any given origin-centered non-degenerate conic, there exists a one parameter family of matrices $\{T(k)\}$ which generate the conic, and $\det[T(k)] = 1$ for all k.

Proof: Let the conic be

$$P^*CP = P^* \begin{pmatrix} a & b \\ b & c \end{pmatrix} P = d \ .$$

We construct a matrix T, such that if P_i is on the conic, then so is $P_{i+1} = T P_i$. Consider $P_{i+1} = P_i + \Delta P_i$. Define:

$$E = (P_i + \Delta P_i)^* \, C \, (P_i + \Delta P_i) - P_i^* \, C \, P_i \ .$$

Expand it:

$$\begin{aligned} E &= 2ax(\Delta x) + a(\Delta x)^2 + 2by(\Delta x) + b(\Delta y)(\Delta x) \\ &\quad + 2bx(\Delta y) + b(\Delta x)(\Delta y) + 2cy(\Delta y) + c(\Delta y)^2 \\ &= (\Delta x)\cdot[2ax + 2by + a(\Delta x) + b(\Delta y)] + (\Delta y)\cdot[2bx + 2cy + b(\Delta x) + c(\Delta y)]. \end{aligned}$$

For Δx, Δy and any k which satisfy the following:

$$\Delta x = k[2bx + 2cy + b(\Delta x) + c(\Delta y)]$$

$$\Delta y = -k[2ax + 2bx + a(\Delta x) + b(\Delta y)]$$

E vanishes, which means that P_{i+1} is on the conic. Separate Δx and Δy:

$$\begin{aligned} (1 - kb)\Delta x \quad -kc \quad \Delta y &= 2k[bx + cy] \\ ka \, \Delta x + (1 + kb)\Delta y &= -2k[ax + by] \end{aligned}$$

which is in matrix form:

$$\left(I - k\begin{pmatrix} b & c \\ -a & -b \end{pmatrix}\right)\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = 2k\begin{pmatrix} b & c \\ -a & -b \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} .$$

Introduce:

$$G = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} a & b \\ b & c \end{pmatrix} = \begin{pmatrix} b & c \\ -a & -b \end{pmatrix} ,$$

then,

$$(I - kG)\Delta P = 2kGP .$$

The matrix (I - kG) is invertible for all (except two, at most) values of k:

$$\Delta P = 2k(I - kG)^{-1}GP$$

$$P_{i+1} = P_i + \Delta P_i = [I + 2k(I - kG)^{-1}G]P_i .$$

Hence $T(k) = I + 2k(I - kG)^{-1}G$. Substitute a, b, c and define $e = b^2 - ac$. Then for $k^2 \neq e^{-1}$:

$$T(k) = \frac{1}{1-k^2e}\begin{pmatrix} 1+2kb+k^2e & 2kc \\ 1-k^2e & 1-2kb+k^2e \end{pmatrix} .$$

It is easy to verify that $\det(T) \equiv 1$, for all k.

Consider the trace of T(k):

$$\mathrm{trace}(T) = 2\,\frac{1+k^2e}{1-k^2e} ;$$

$$\mathrm{trace}(T) = \left\{\begin{matrix} < 2 & \text{if} & e < 0 \\ = 2 & \text{if} & e = 0 \\ > 2 & \text{if} & e > 0 \end{matrix}\right\} \quad \text{for all } k .$$

It is well-known that e < 0 implies an ellipse, e > 0 implies an hyperbola, and e = 0 implies straight lines.

Corollary: The generating matrix for the circle $x^2 + y^2 = \alpha$ is obtained by substituting $a = c = 1$ and $b = 0$ $(e = -1)$:

$$T_C = \frac{1}{1+k^2}\begin{pmatrix} 1-k^2 & 2k \\ -2k & 1-k^2 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} .$$

For the hyperbola $x^2 - y^2 = \beta$, substitute $a = -c = 1$ and $b = 0$ $(e = +1)$:

$$T_H = \frac{1}{1-k^2}\begin{pmatrix} 1+k^2 & -2k \\ -2k & 1+k^2 \end{pmatrix} = \begin{pmatrix} \mathrm{ch}\,\phi & \mathrm{sh}\,\phi \\ \mathrm{sh}\,\phi & \mathrm{ch}\,\phi \end{pmatrix}$$

where $\theta = \operatorname{arctg} \frac{-2k}{1+k^2}$ and $\theta = \operatorname{argth} \frac{-2k}{1-k^2}$.

(III. 4) Obtaining the Generating Matrix for a Conic (Geometric Approach)

Theorem: If T generates the curve Π, and the linear transformation H, maps Π into the curve Σ, such that $\Sigma = H\Pi$ then the generating matrix of Σ is $S = HTH^{-1}$.

Proof: Let $\Pi = \{P(n)\}$ and $\Sigma = \{S(n)\}$ such that $S(n) = HP(n)$ for all n, then:

$$S(n+1) = HP(n+1) = HTP(n) = HTH^{-1}S(n) . \qquad \text{QED}$$

Namely, S and T are "similar matrices", and have the same eigenvalues (and therefore the same determinant and trace).

This theorem suggests another method for finding the generating matrices for ellipses and hyperbola. Consider the ellipse, whose axes are parallel to the X-Y axes, the length of the horizontal axis is 2λ, and the length of the vertical axis is 2μ, as shown in Figure III. 4. 1. This ellipse is obtained from the unit circle by the transformation:

$$D = \begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix} .$$

If this ellipse was tilted by the angle α then it could be obtained from the unit circle by the transformation $R(\alpha)D$, where

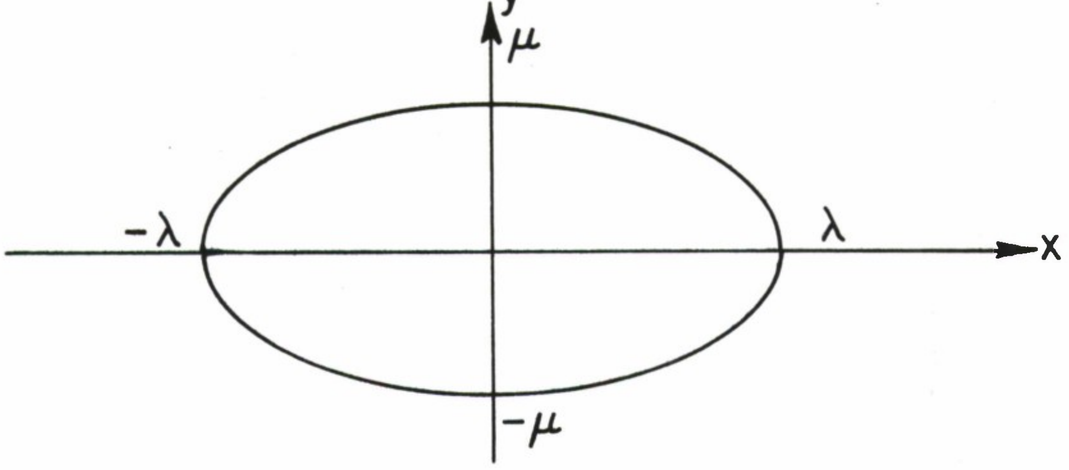


Figure III. 4. 1: The ellipse $\frac{x^2}{\lambda^2} + \frac{y^2}{\mu^2} = 1$

$$R(a) = \begin{pmatrix} \cos a & -\sin a \\ \sin a & \cos a \end{pmatrix} .$$

Hence the generating matrix of the ellipse is $E = R(a)\,D\,R(\theta)D^{-1}R(-a)$.

The generating matrix of $x^2 - y^2 = 1$ is

$$T_H(\phi) = \begin{pmatrix} \mathrm{ch}\,\phi & \mathrm{sh}\,\phi \\ \mathrm{sh}\,\phi & \mathrm{ch}\,\phi \end{pmatrix} .$$

Consider the hyperbola in Figure III. 4. 2, which is obtained from $x^2 - y^2 = 1$ by the transformation

$$H = R(a)D = \begin{pmatrix} \cos a & -\sin a \\ \sin a & \cos a \end{pmatrix} \begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix} .$$

Hence its generating matrix is:

$$T = R(a)DT_H(\phi)\,D^{-1}\,R(-a) .$$

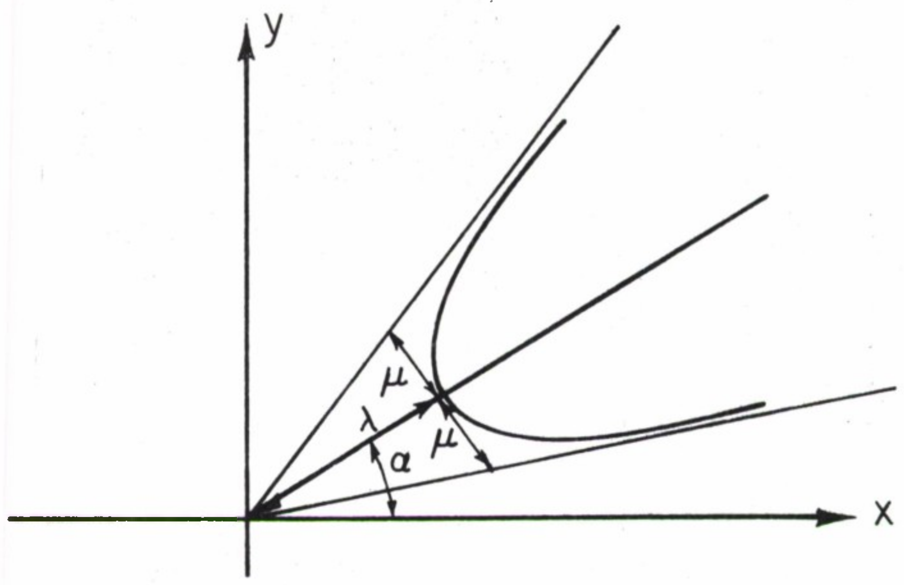


Figure III. 4. 2: A tilted hyperbola

(III. 5) Characterization of Conics by the Generating Matrices

Theorem: If $\Pi = \{P(s) | P(s) = T^s P_0\}$ and $\det(T) = 1$, then Π is a conic, whose type depends on trace(T).

Proof: Assume that $T \neq \pm I$ as these cases generate either P_0 repeatedly or P_0 and $-P_0$ alternately. Let

$$T = \begin{pmatrix} a & b \\ c & d \end{pmatrix} .$$

Consider the following cases:

(a) $|a+d| > 2$ (b) $|a+d| = 2$ (c) $|a+d| < 2$.

Case (a): If $|a+d| > 2$. Consider λ, an eigenvalue of T,

$$\lambda = \frac{a+d}{2} + \sqrt{\frac{1}{4}(a+d)^2 - 1} .$$

T can be written as:

$$T = QDQ^{-1} \text{ where } Q = \begin{pmatrix} b & \frac{1}{\lambda} - d \\ \lambda - a & c \end{pmatrix} \text{ and } D = \begin{pmatrix} \lambda & 0 \\ 0 & \frac{1}{\lambda} \end{pmatrix} .$$

D can be written as

$$D = ST_HS^{-1} \quad \text{where} \quad S = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \text{and} \quad T_H = \begin{pmatrix} \text{ch}\,\phi & \text{sh}\,\phi \\ \text{sh}\,\phi & \text{ch}\,\phi \end{pmatrix}$$

where $\text{ch}\,\phi = \frac{1}{2}(\lambda + \frac{1}{\lambda})$ and $\text{sh}\,\phi(\lambda - \frac{1}{\lambda})$. Hence,

$$T = QDQ^{-1} = (QS)T_H(QS)^{-1} \quad .$$

T_H generates the hyperbola H, and T generates the hyperbola (QS)H.

Case (b): If $a+d = 2$ and $c \neq 0$ then:

$$T = \begin{pmatrix} a-1 & 1 \\ c & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a-1 & 1 \\ c & 0 \end{pmatrix}^{-1} = QLQ^{-1} \quad .$$

The matrix $L = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ generates lines since

$$P_\ell = L^\ell P_0 = \begin{pmatrix} 1 & \ell \\ 0 & 1 \end{pmatrix} P_0 = P_0 + \ell \begin{pmatrix} y_0 \\ 0 \end{pmatrix} \quad .$$

Because L generates a line so does T. If $a+d = -2$ and $c \neq 0$ then

$$T = \begin{pmatrix} a+1 & -1 \\ c & 0 \end{pmatrix} \begin{pmatrix} -1 & 1 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} a+1 & -1 \\ c & 0 \end{pmatrix}^{-1} \quad .$$

The matrix $\begin{pmatrix} -1 & 1 \\ 0 & -1 \end{pmatrix}$ generates a line, or two lines, and so does T.

If $c = 0$ then:

$$T = \begin{pmatrix} -1 & b \\ 0 & -1 \end{pmatrix}$$

which generates one or two lines.

Case (c): If $|a+d| > 2$, consider λ, an eigenvalue of T,

$$\lambda = \frac{a+d}{2} + i\sqrt{1 - \frac{1}{4}(a+d)^2} = e^{i\theta}$$

where $\cos\theta = \frac{1}{2}(a+d)$ and

$$\sin\theta = \sqrt{1 - \frac{1}{4}(a+d)^2} \quad .$$

The sign of $\sin\theta$ is chosen to be like the sign of b, because of a reason that becomes clear later. Note that $b \neq 0$ because $b = 0$ implies $ad = 1$, which implies $|a+d| > 2$. We will show later the existence of a real matrix Q such that

$$T = QT_C Q^{-1} = Q \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} Q^{-1} \quad .$$

As T_c generates the circle C, T generates a linear transformation of it, which is the ellipse QC. We will construct the matrix Q. As T_c and T do not determine Q uniquely, we can expect some freedom in the solution for Q.

$$QT_c Q^{-1} = \begin{pmatrix} x & y \\ z & w \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} w & -y \\ -z & x \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} = T \; .$$

Equate components:

$$(E_1):\; a = (xw - yz)\cos\theta - (xz + yw)\sin\theta$$

$$(E_2):\; b = (x + y)\sin\theta$$

$$(E_3):\; c = -(z + w)\sin\theta$$

$$(E_4):\; d = (xw - yz)\cos\theta + (xz + yw)\sin\theta$$

$$(E_5):\; 1 = xw - yz \; .$$

The identity $(xw - yz)^2 + (xz + yw)^2 = (x^2 + y^2)(z^2 + w^2)$ shows that one of the equations E_1 through E_4 can be discarded, and we can impose another external condition on the system. We choose $y = 0$, then

$$x = \sqrt{\frac{b}{\sin\theta}} = \frac{b}{\sqrt{b\sin\theta}}$$

$$w = \frac{1}{x} = \frac{\sin\theta}{\sqrt{b\sin\theta}}$$

$$z = \frac{d-a}{2x \sin\theta} = \frac{d-a}{2\sqrt{b \sin\theta}} \quad .$$

Note that $b \sin\theta > 0$. We get:

$$Q = \frac{1}{2\sqrt{b \sin\theta}} \begin{pmatrix} 2b & 0 \\ d-a & 2\sin\theta \end{pmatrix} \quad .$$

This is the real matrix Q such that $T = QT_cQ^{-1}$, hence

$$\begin{pmatrix} 2b & 0 \\ d-a & 2\sin\theta \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 2b & 0 \\ d-a & 2\sin\theta \end{pmatrix}^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad .$$

This completes the proof of the theorem.

(III. 6) On the Spacing Between Successive Points

Theorem: Let us consider the circle $x^2 + y^2 = 1$, and the hyperbola $x^2 - y^2 = 1$, both passing through the point $P_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. The circle is generated by the matrix T_C, and the hyperbola by T_H where

$$T_C = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} , \text{ and } T_H = \begin{pmatrix} \mathrm{ch}\,\phi & \mathrm{sh}\,\phi \\ \mathrm{sh}\,\phi & \mathrm{ch}\,\phi \end{pmatrix} \quad .$$

The point P_n on the circle is

$$P_n = T_C{}^n P_0 = \begin{pmatrix} \cos(n\theta) \\ \sin(n\theta) \end{pmatrix}$$

and the point P_n on the hyperbola is

$$P_n = T_H{}^n P_0 = \begin{pmatrix} \mathrm{ch}(n\phi) \\ \mathrm{sh}(n\phi) \end{pmatrix} \quad .$$

Let d_n be the distance between P_n and P_{n+1}. We want to show that d_n is constant for the circle, but is an increasing function of $|n|$ for the hyperbola.

Proof: For the circle:

$$P_n = \begin{pmatrix} \cos(n\theta) \\ \sin(n\theta) \end{pmatrix}$$

$$\begin{aligned} d_n &= (x_{n+1} - x_n)^2 + (y_{n+1} - y_n)^2 \\ &= [\cos(n+1)\theta - \cos n\theta]^2 + [\sin(n+1)\theta - \sin n\theta]^2 \\ &= [-2 \sin\frac{2n+1}{2}\theta \sin\frac{\theta}{2}] + [2\cos\frac{2n+1}{2}\theta \sin\frac{\theta}{2}]^2 \\ &= 4\sin^2\frac{\theta}{2}[\sin^2\frac{2n+1}{2}\theta + \cos^2\frac{2n+1}{2}\theta] = 4\sin^2\frac{\theta}{2} \end{aligned}$$

which is independent of n. QED

For the hyperbola $P_n = \begin{pmatrix} ch(n\phi) \\ sh(n\phi) \end{pmatrix}$

$$\begin{aligned} d_n^2 &= (x_{n+1} - x_n)^2 + (y_{n+1} - y_n)^2 \\ &= [ch(n+1)\phi - ch\, n\phi]^2 + [sh(n+1)\phi - sh\, n\phi]^2 \\ &= [2\, sh\frac{2n+1}{2}\phi\, sh\frac{\phi}{2}]^2 + [2\, ch\frac{2n+1}{2}\phi\, sh\frac{\phi}{2}]^2 \\ &= 4\, sh^2\frac{\phi}{2}[sh^2\frac{2n+1}{2}\phi + ch^2\frac{2n+1}{2}\phi] \\ &= 4\, sh^2\frac{\phi}{2}\, ch(2n+1)\phi \end{aligned}$$

which is an increasing function of $|n|$. QED

This theorem is illustrated in Figures III.10.1 through III.10.4.

Theorem: The Areas Law: Let $P_n = T^n P_0$ and let $\tau = \det(T)$. Let S_i be the triangle whose vertices are the origin and the points P_n and P_{n+1}. Let A_n be the area of S_n. Then, $A_n = \tau^n A_0$.

Proof:

$$A_0 = \frac{1}{2}(x_0y_1 - x_1y_0) = \frac{1}{2}(x_0 y_0) \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \frac{1}{2}P_0^* G P_1 .$$

Substitute $P_1 = TP_0$ and get $A_0 = \frac{1}{2}P_0^* GTP_0$. Similarly

$$A_n = \frac{1}{2}P_0^*(T^n)^*(GT)T^nP_0 \ .$$

Consider

$$T^*GT = \begin{pmatrix} a & c \\ b & d \end{pmatrix}\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} -c & a \\ -d & b \end{pmatrix}\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$$= \begin{pmatrix} -ac + ac & -bc + ad \\ -ad + bc & -bd + bd \end{pmatrix} = \begin{pmatrix} 0 & \tau \\ -\tau & 0 \end{pmatrix} = \tau G$$

and $(T^*)^nGT^n = \tau^nG$. Substitute above and get

$$A_n = \frac{1}{2}P_0^*[(T^n)^*GT^n]TP_0 = \frac{1}{2}\tau^nP_0^* GTP_0 = \tau^nA_0 \ . \qquad \text{QED}$$

The reader should notice that although there is a similarity this is not the Kepler area law.

Corollary: On a conic all the areas $\{A_n\}$ are equal.

Corollary: The distance between successive points on ellipses gets smaller when the distance of the points from the origin gets longer.

It is clear that by stretching a uniformly spaced circle into an ellipse the uniform spacing is changing into a good spacing.

The first theorem shows that it is possible to generate conics in good spacing, but because of the second theorem any matrix which generates a conic, must generate it in good spacing.

(III. 7) Straight Lines as Degenerate Conics

There are two kinds of straight lines which are degenerate conics. The ones which pass through the origin, and the ones which do not.

The lines which pass through the origin are asymptotes to hyperbolas, and are generated by the same matrices which generate the hyperbolas when the initial points are eigenvectors of the matrices.

The matrices T, which generate hyperbolas satisfy det(T) = 1 and trace(T) > 2. These conditions guarantee the existance of two distinct eigenvectors, which introduce the two asymptotes.

The lines which do not pass through the origin, are arcs of an ellipse whose major axis is infinite. For example, the ellipse whose axes are of length infinity and of length μ, is the two parallel lines $y = \pm\mu$. The implicit form of the ellipse is $y^2 = \mu^2$. Using the method of (III. 3), and substituting a = b = 0; c = 1 (e = -1) we get:

$$T(k) = \frac{1}{1+k^2}\begin{pmatrix} 1+k^2 & 2k \\ 0 & 1+k^2 \end{pmatrix} = \begin{pmatrix} 1 & \hat{k} \\ 0 & 1 \end{pmatrix} .$$

As expected, trace[T(k)] = 2.

This matrix, with the initial point $P_0 = (x_0, y_0)^*$ generates the line $y = y_0$, in the positive direction along the ellipse, which is to the right for $y_0 > 0$ and to the left for $y_0 < 0$. All the points of the form $P_0 = (x_0, 0)^*$ are eigenvectors of T, corresponding to the eigenvalue 1, and therefore are not changed by the iteration. It is easy to show that T does not have any other eigenvectors.

The generating matrices of proper ellipses [det(T) = 1, |trace(T)| < 2] do not have any real eigenvectors, and cannot generate any lines.

(III. 8) <u>On Curves Created by Matrices with a Non-unit Determinant</u>

This section suggests, implicitly, building special hardware for the iteration $P(n+1) = T\,P(n)$ or $\Delta P(n+1) = T\,\Delta P(n)$. This hardware can, as shown before, generate conics. However it is interesting to know what happens if one iterates with a matrix with non-unit determinant (which is a necessary condition for conics). In this subsection we answer this question.

It should be noted that if T has negative eigenvalues then the function T^s is not well defined and required an extra care for getting continuity. However by observing only integer powers of T, most of this danger is bypassed.

(III. 8. 1) Consider the cases $\tau = \det(T) > 0$. The matrix $S = \tau^{-1/2}T$ satisfies $\det(S) = 1$, and S generates some conic. If S generates an ellipse then T generates an elliptic spiral which winds outward to infinity if $\tau > 1$, or winds inward to zero if $\tau < 1$. See Figures III. 10. 5 through III. 10. 8.

If S generates an hyperbola, then T has two distinct eigenvalues, λ and μ. If both are positive then we can define α by $\lambda^{\alpha} = \mu$. Then

$$T = H \begin{pmatrix} \lambda & 0 \\ 0 & \lambda^{\alpha} \end{pmatrix} H^{-1}$$

which shows that T generates a linear transformation of $y = kx^{\alpha}$. If both are negative then $-T$ has two positive eigenvalues. T generates points which alternate between the curve which is generated by $-T$ from P_0, and the curve which is generated by $-T$ from $-P_0$. Note that $\alpha = -1$ implies an hyperbola, and $\alpha = 0$ (i. e., 1 is an eigenvalue of T) implies a straight line. See Figures III. 10. 9 through III. 10. 11.

If S generates a straight line then if $\tau \neq 1$, T generates lines which belong to a family of curves, all of which pass through the origin, and go to infinity. The shape of these curves is illustrated in Figure III. 10. 12. T generates these curves (or a linear transformation of them). Note that one straight line belongs to this family.

(III. 8. 2) Consider $\tau = \det(T) = 0$. If T is a non-zero matrix then $\dim[\mathrm{Range}(T)] = 1$, which means that all $\{P(s)\}$ belongs to a 1-dimensional space. Hence T generates a straight line through the origin.

(III. 8. 3) Consider $\tau = \det(T) < 0$. If $\det(T) < 0$ then $\det(T^2) > 0$. Hence T^2 generates one of the curves discussed before. Every second point $\{P(2n)\}$ which is generated by T is on the curve which is generated by T^2. The other points $\{P(2n+1)\}$ are on the curve which is generated by T^2 through $P(1)$. Hence T generates a sequence of points which oscillate between two curves of the same type.

(III. 9) Programming Aspects of Coding the Linear Difference Scheme

This section suggests an incremental method for generating curves. We pointed out in (III. 2) that the scheme which is used for generating the $\{P(n)\}$ can be used for generating the $\{\Delta P(n)\}$.

This iteration can be implemented either by conventional programming or by hardware. We give in this subsection some "coding-tips" for programming the linear differences scheme.

(III. 9. 1) When $P(n+1) = T\,P(n)$ is coded there is no need to store the array of $\{P(n)\}$, as one can do only with the current value of P, i.e., the values of x and y. The straightforward coding of the iteration is:

$$\begin{aligned} ax + by &\longrightarrow \text{temp} \\ cx + dy &\longrightarrow y \\ \text{temp} &\longrightarrow x \quad . \end{aligned}$$

The need for the temporary storage "temp" rises because x cannot be changed before it is used for the y calculation. However, the iteration can be defined such that x(n+1) is expressed by means of x(n) and y(n), but y(n+1) is expressed by x(n+1) and y(n).

Consider the following identity:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ c/a & (ad-bc)/a \end{pmatrix} \begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix} \quad .$$

Multiply by P(n):

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ c/a & (ad-bc)/a \end{pmatrix} \begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ c/a & (ad-bc)/c \end{pmatrix} \begin{pmatrix} x_{n+1} \\ y_n \end{pmatrix} \quad .$$

This may be coded as:

$$\begin{aligned} ax + by &\longrightarrow x \\ \alpha x + \beta y &\longrightarrow y \end{aligned}$$

where $\alpha = \frac{c}{a}$ and $\beta = \frac{ad-bc}{a}$. If $a = 0$ but $d \neq 0$, a similar scheme works. This eliminates the need for the temporary storing.

(III. 9. 2) The well-known iteration:

$$x - \delta \cdot y \longrightarrow x$$
$$y + \delta \cdot x \longrightarrow y$$

generates an ellipse, because y uses the "new" x as set by the first statement. This iteration can be formulated as:

$$x_{n+1} = x_n - \delta \cdot y_n$$

$$y_{n+1} = y_n + \delta \cdot x_{n+1} = (1 - \delta^2)y_n + \delta x_n$$

or

$$P(n+1) = \begin{pmatrix} x \\ y \end{pmatrix}_{n+1} = \begin{pmatrix} 1 & -\delta \\ \delta & 1-\delta^2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}_n = E\,P(n)$$

$\det(E) = 1$, $\operatorname{trace}(E) = 2 - \delta^2 < 2$, hence E generates an ellipse.

(III. 9. 3) The iteration:

$$x + \delta \cdot y \longrightarrow x$$
$$y + \delta \cdot x \longrightarrow y$$

generates a hyperbola, because it is equivalent to:

$$P(n+1) = \begin{pmatrix} 1 & \delta \\ \delta & 1+\delta^2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}_n = H\,P(n)$$

and $\det(H) = 1$, $\operatorname{trace}(H) = 2 + \delta^2 > 2$.

(III. 10) Examples

All the pictures (except III. 10. 2) which are appended to this subsection were taken from a PDP-1 program which generates curves using the method which is discussed in this section.

Figure III. 10. 1: A circle generated by 16 segments. Note the uniform spacing.

Figure III. 10. 2: A circle which was generated according to the method which is discussed in Section II. Note the non-uniform spacing.

Figure III. 10. 3: 2 ellipses. The generated points are marked by asterisks. Note the "good" spacing, regardless of the starting point.

Figure III. 10. 4: A family of hyperbolas, all of which were generated by the same matrix. Note the good spacing.

Figure III. 10. 5: A circular spiral which is generated by a circle-generating matrix, multiplied by a scalar, which has a magnitude less than 1.

Figure III. 10. 6: An elliptic spiral which is generated by an ellipse-generating matrix, multiplied by a scalar whose magnitude is less than 1.

Figure III. 10. 7: A star is generated by a rotation matrix, with $\theta = 2\pi/5$.

Figure III. 10. 8: A star spiral is generated by the matrix of picture 7, multiplied by a scalar whose magnitude is less than 1.

Figure III. 10. 9: The parabola $y = x^2$ is generated by the matrix $T = \text{diag}\{a, a^2\}$. Each part of the parabola has to be generated separately.

Figure III. 10. 10: The cubic $y = x^3$ is generated by the matrix $T = \text{diag}\{a, a^3\}$. Each part of the cubic has to be generated separately.

Figure III. 10. 11: The sequence of points 1-2-3-4-5-6-7-8-9-10 was generated by the matrix -H, where H is the generating matrix of the hyperbolas 1-3-5-7-9 and 2-4-6-8-10.

Figure III. 10. 12: A family of curves which is generated by a matrix which has two equal eigenvalues, but only one independent eigenvector. As discussed before, this family contains one straight line, which is in this example the X-axis.

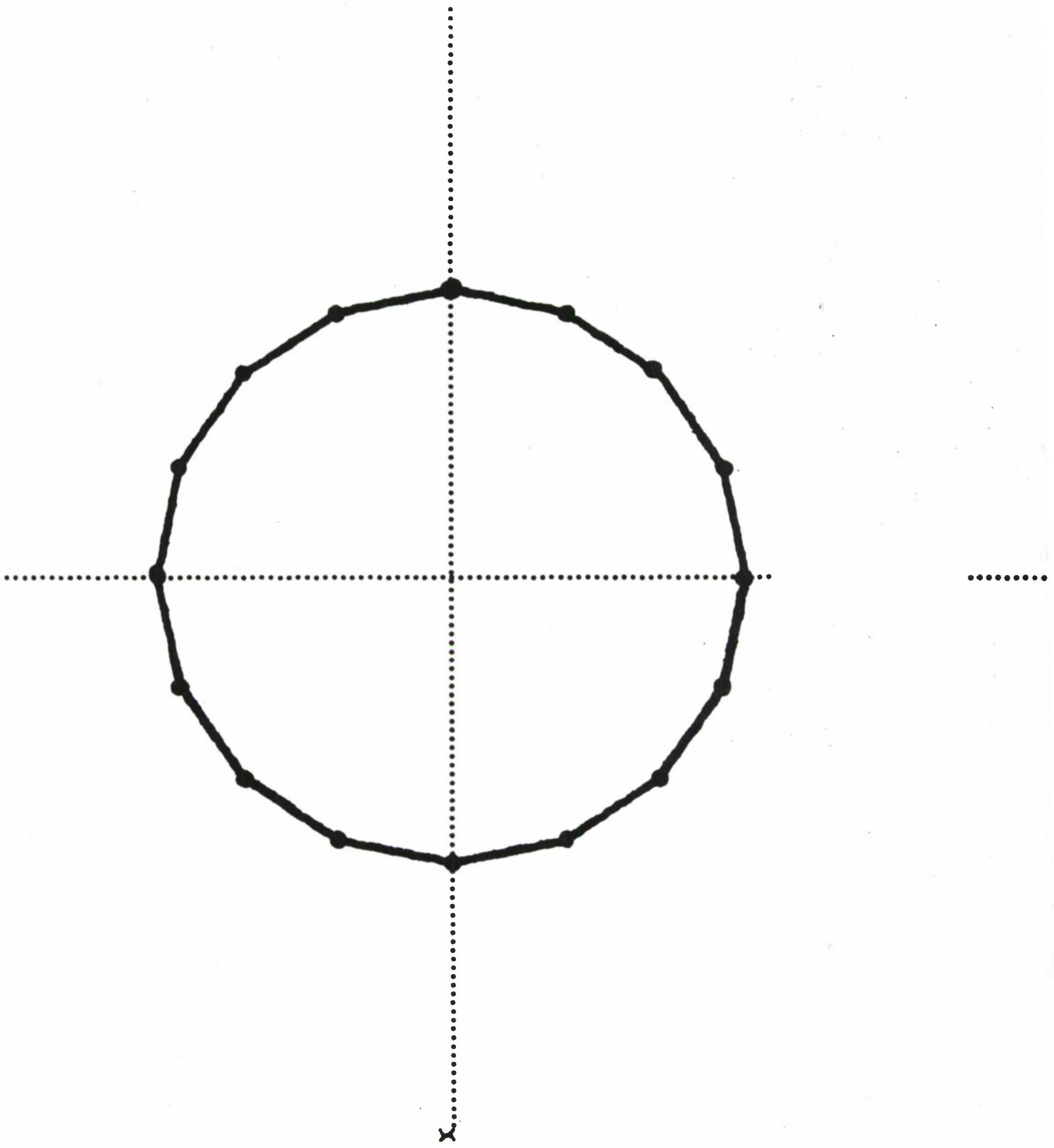


Figure III.10.

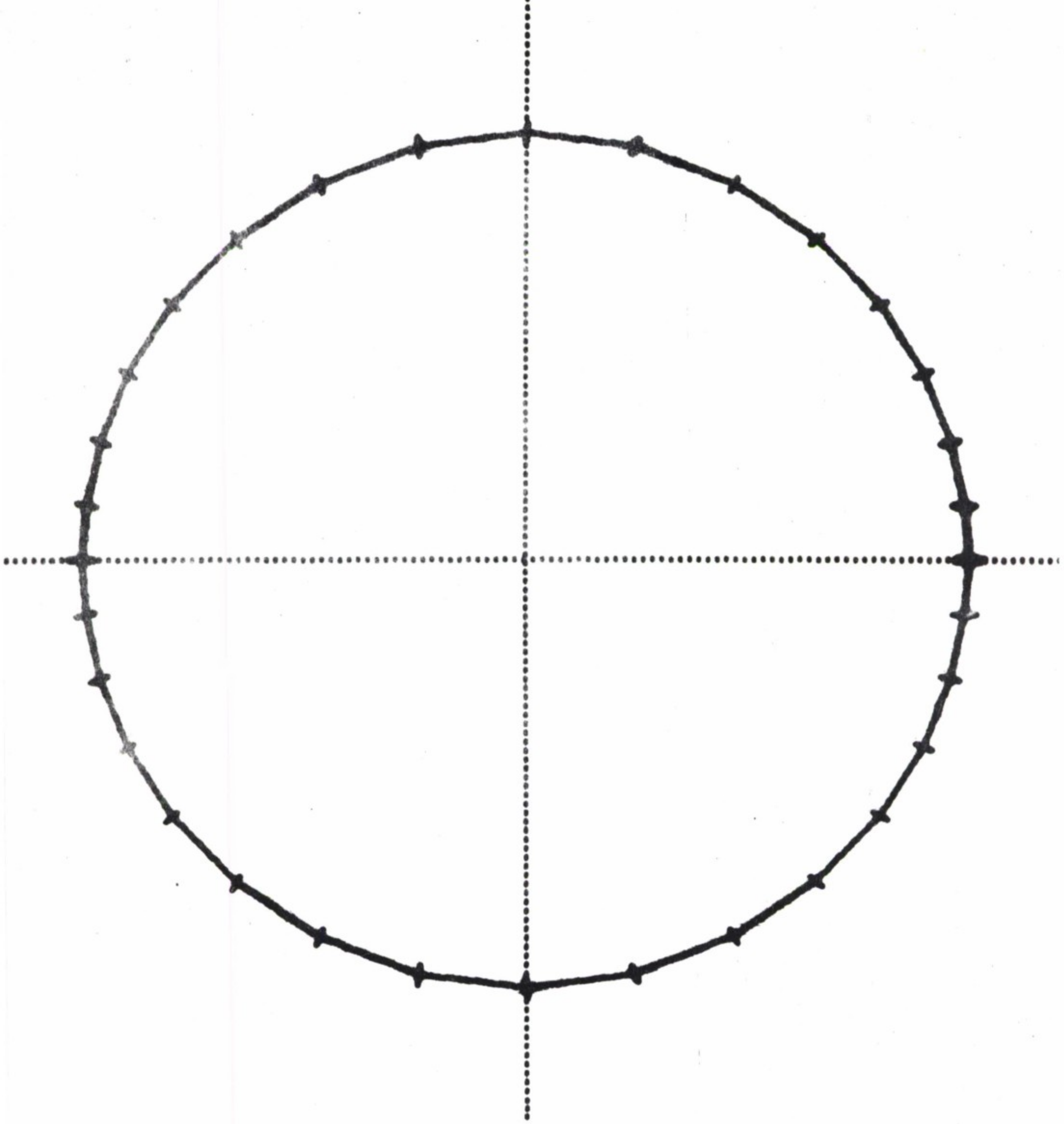


Figure III.10.2

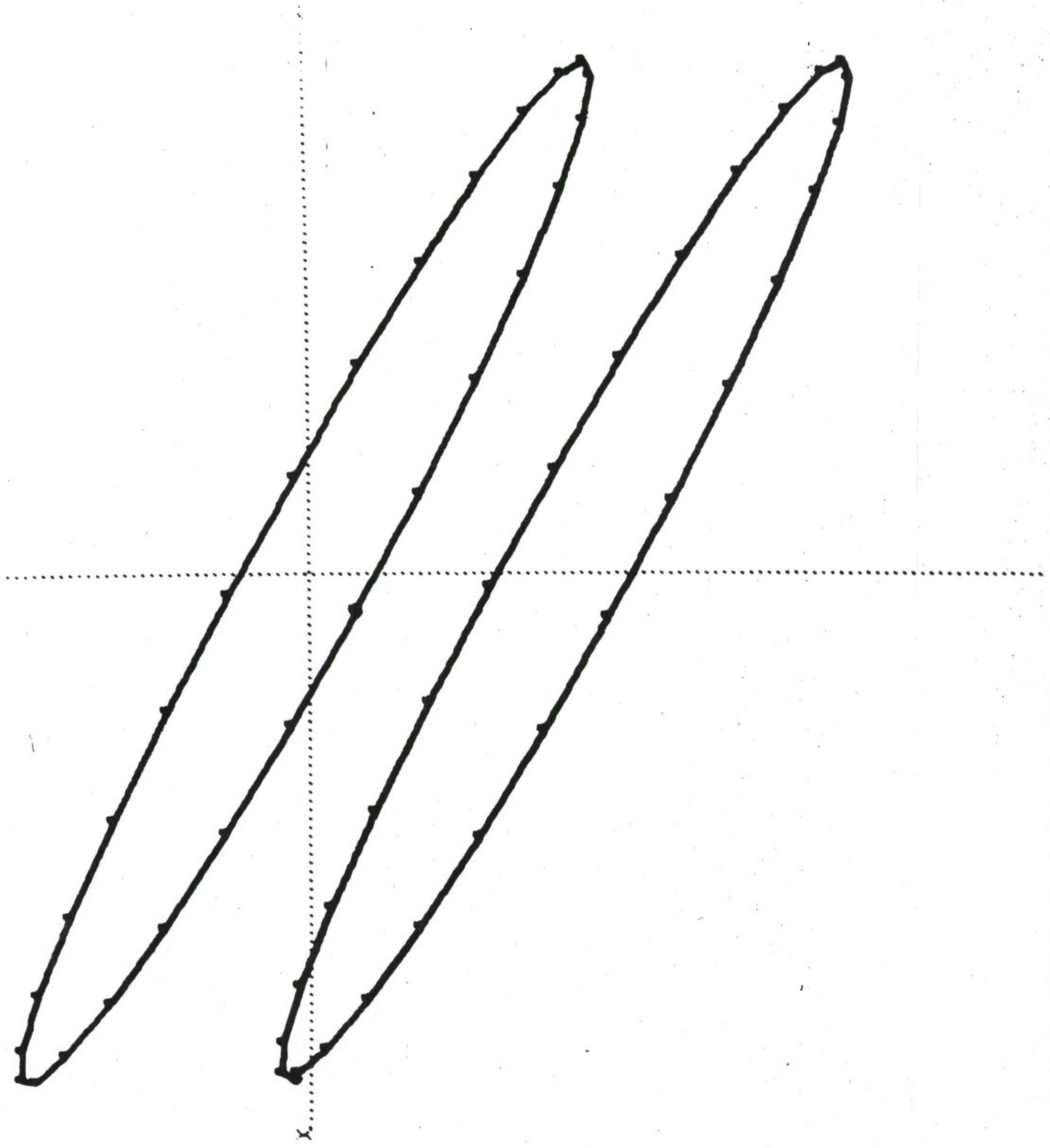


Figure III.10

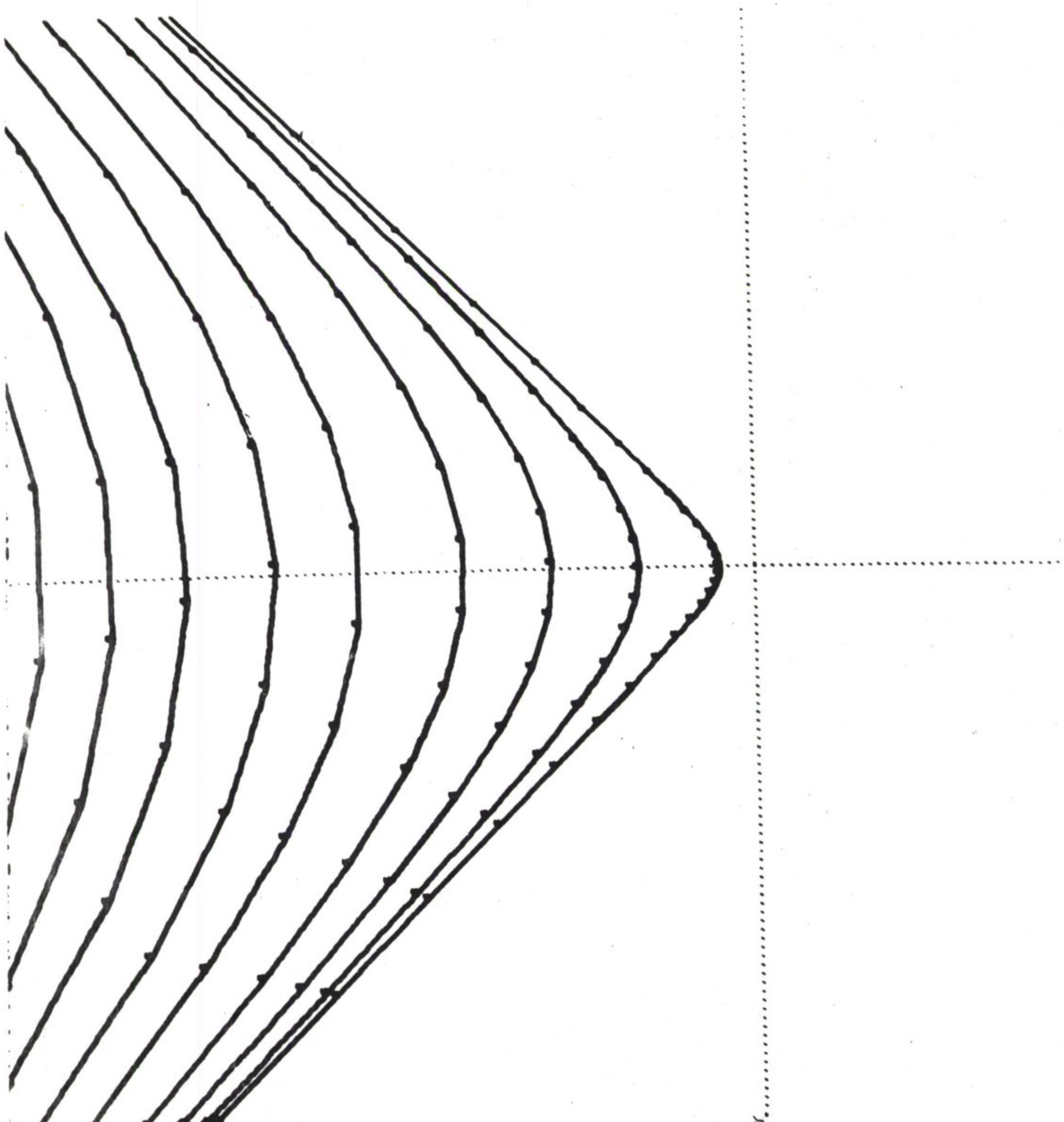


Figure III.10.4

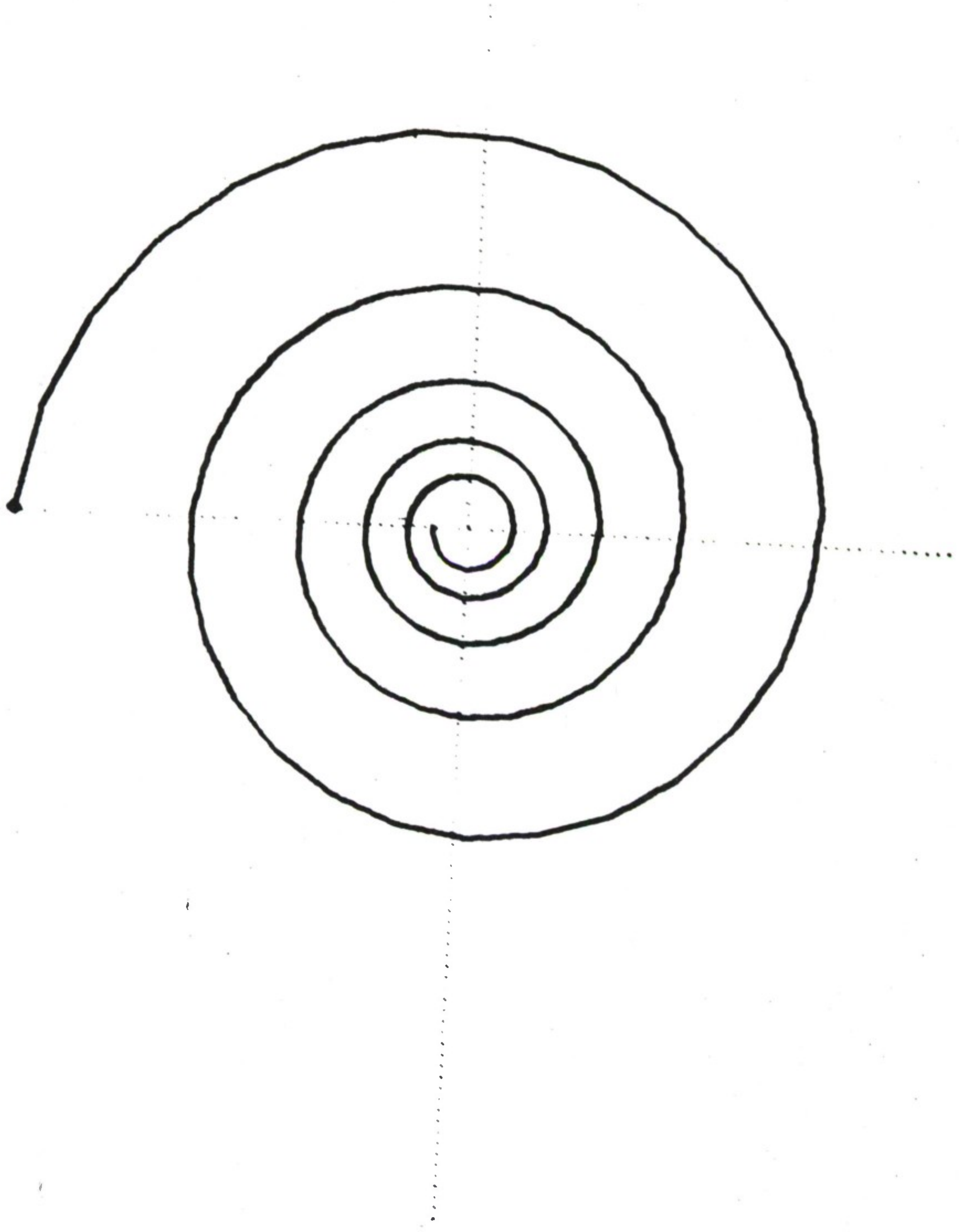


Figure III.10.5

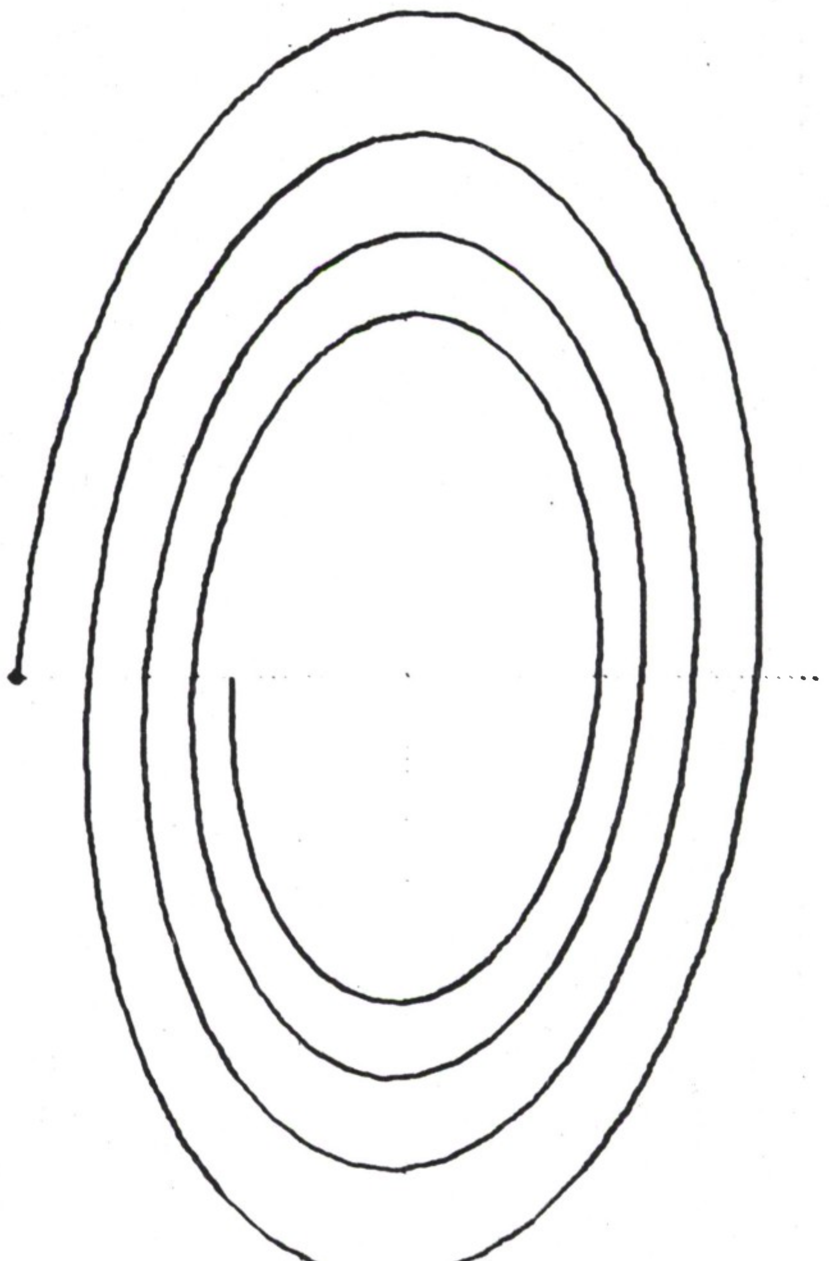


Figure III.10.6

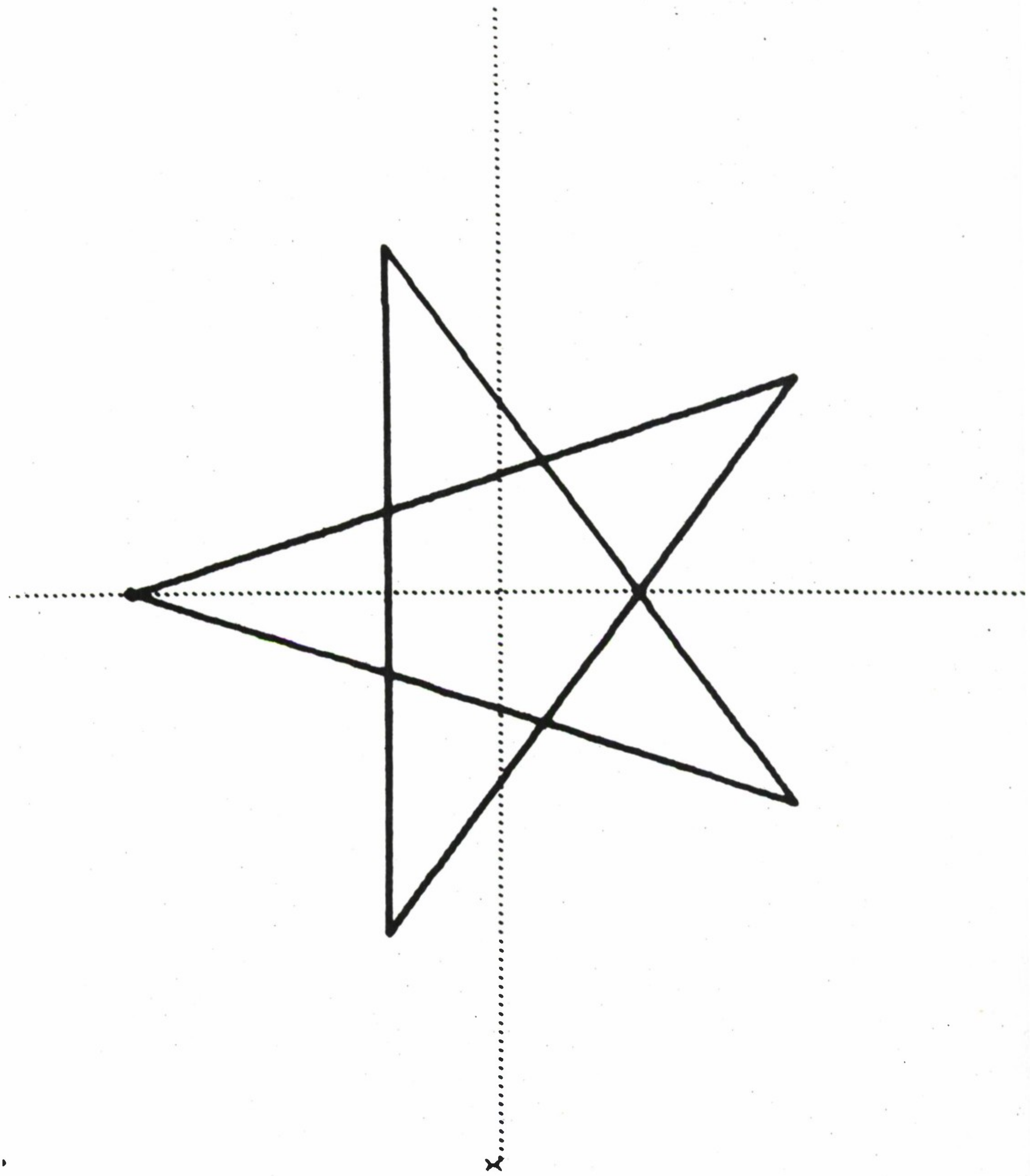


Figure III.10.7

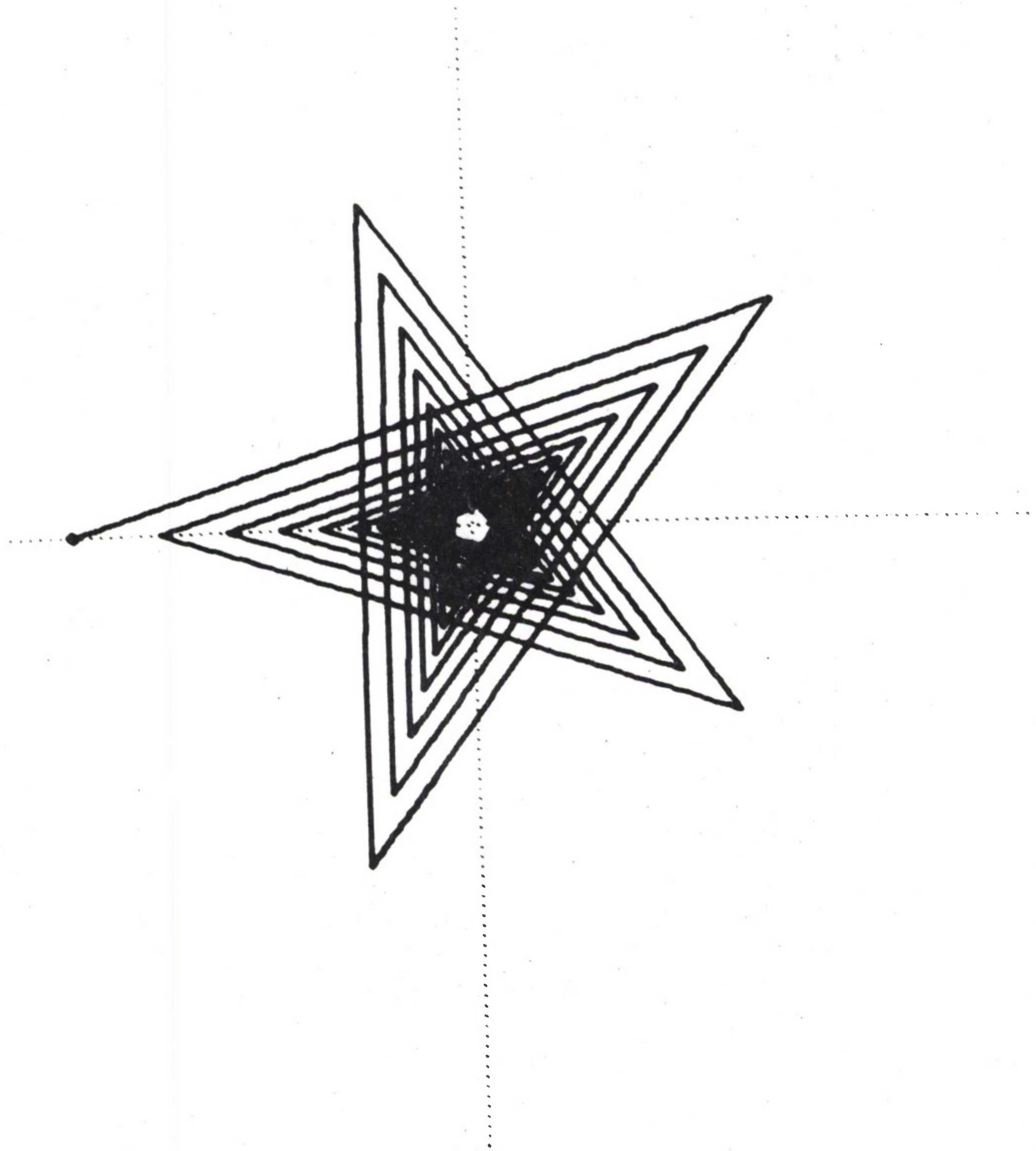


Figure III.10.8

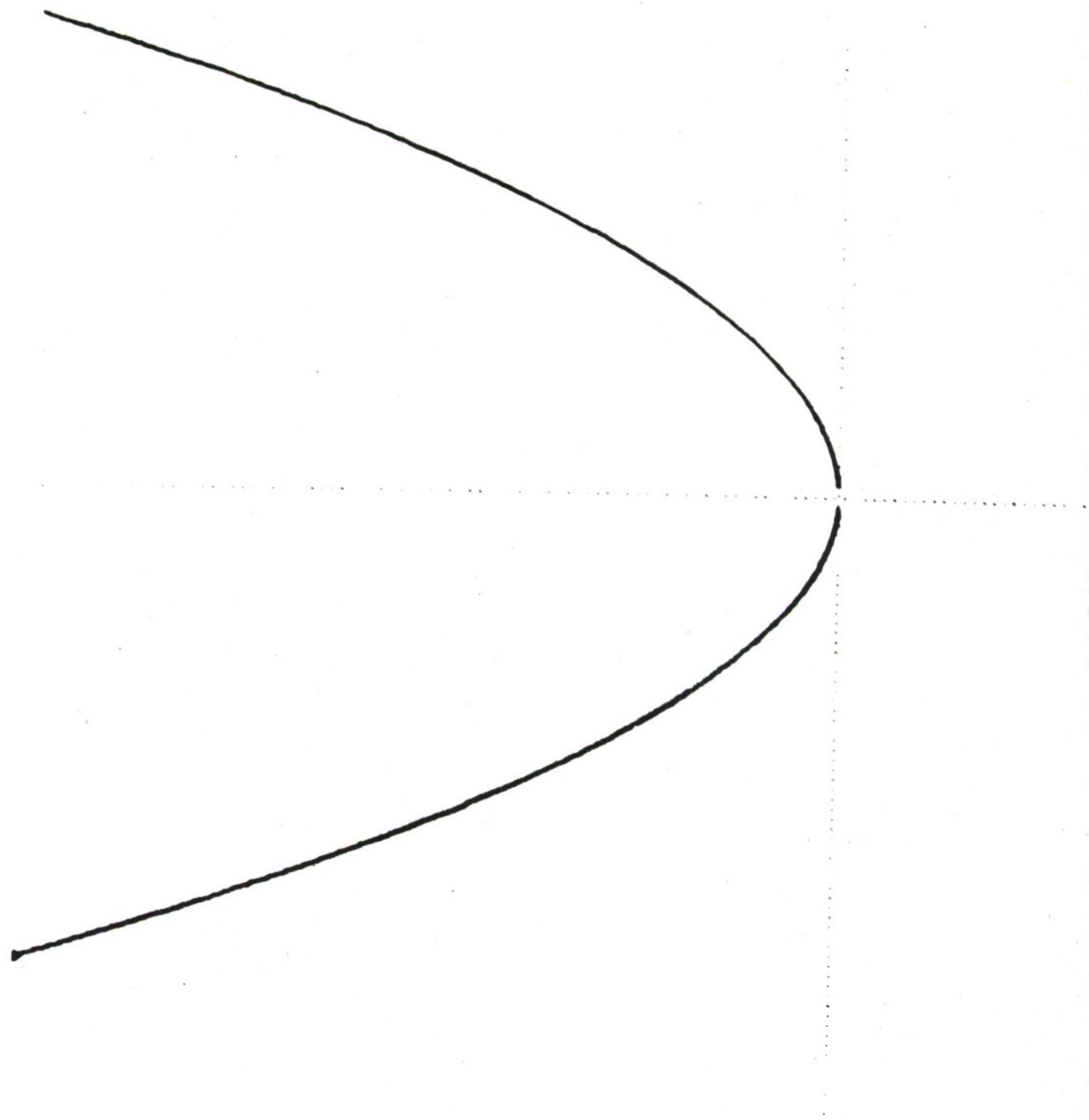


Figure III.10.9

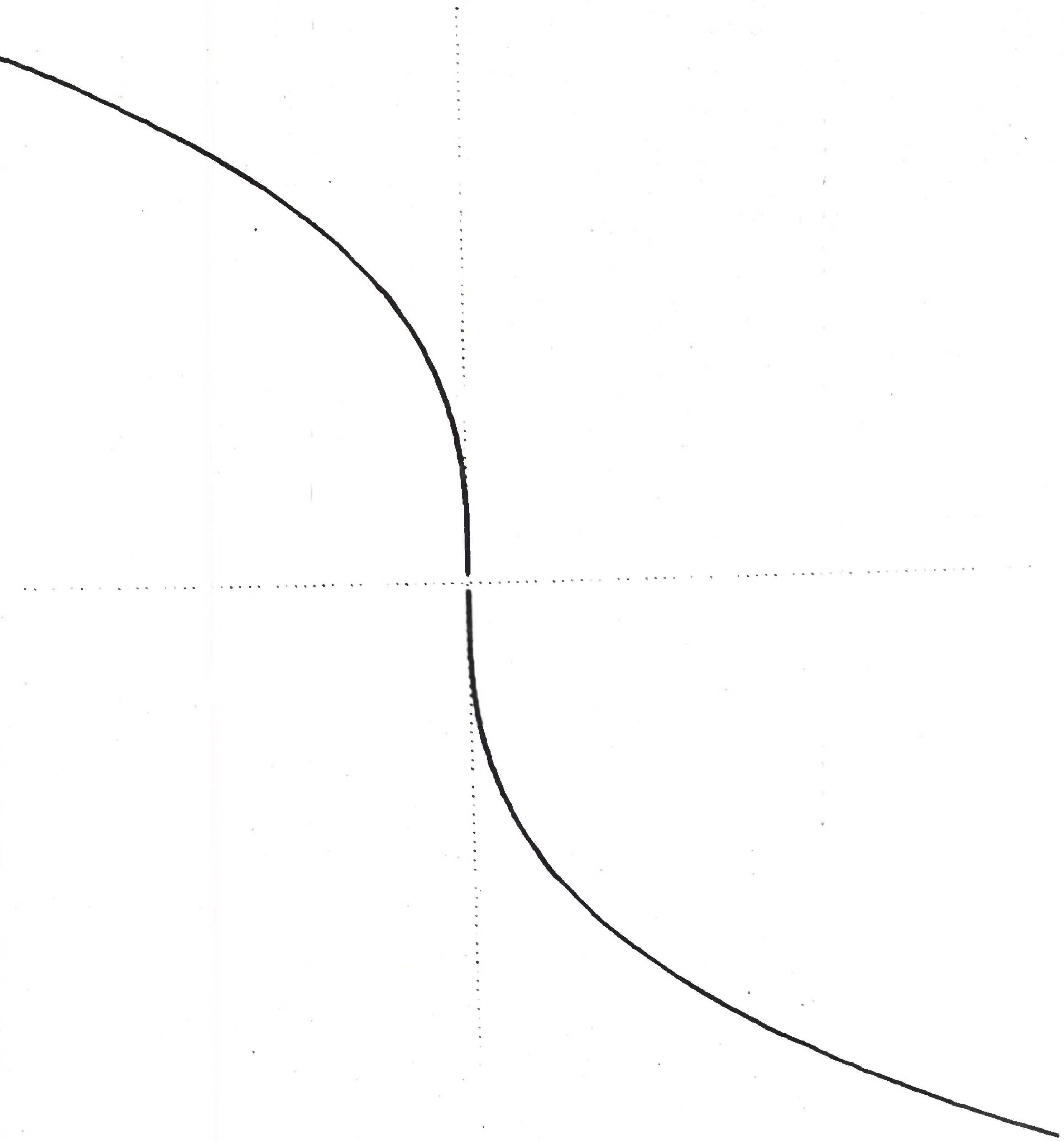


Figure III.10.10

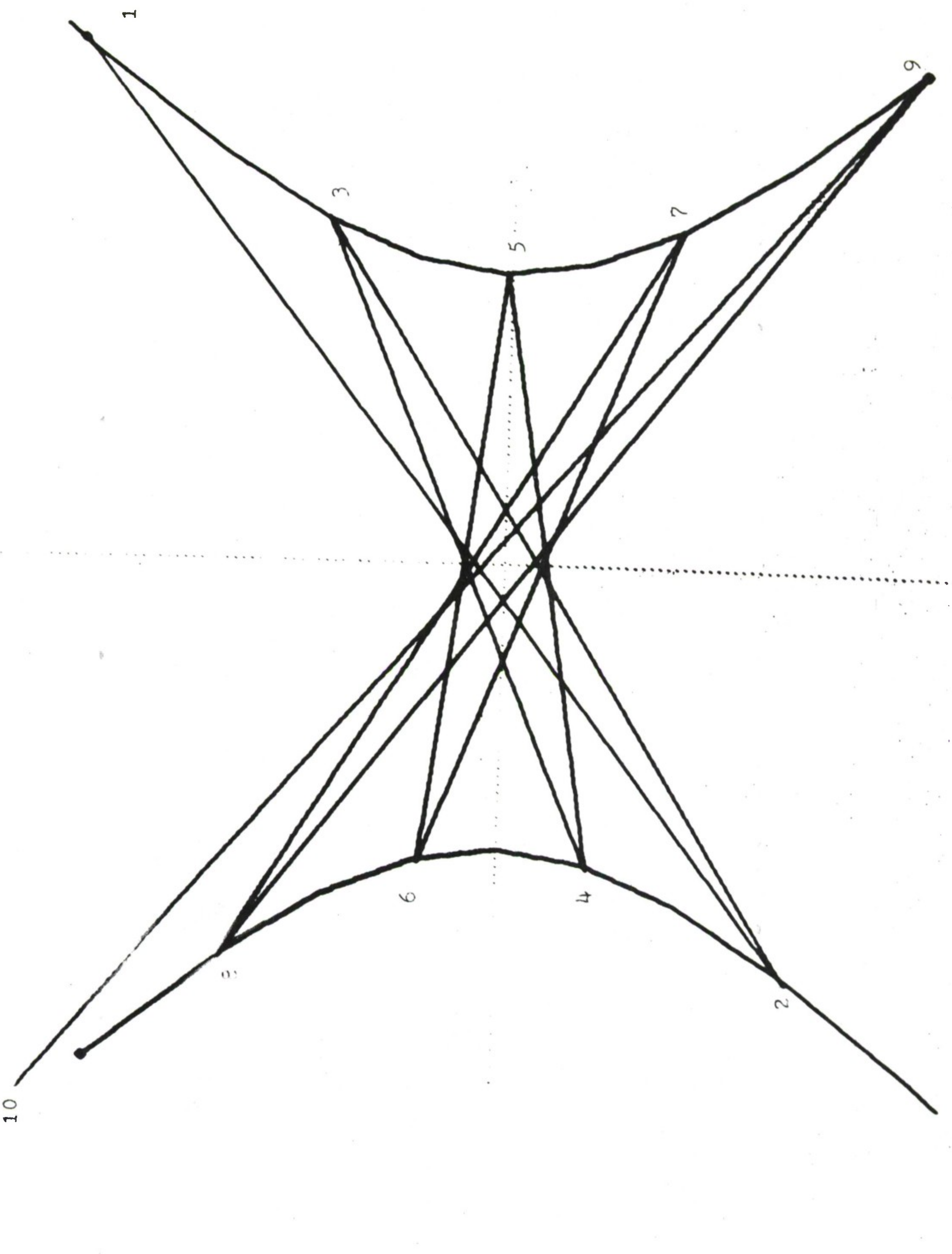


Figure III.10.11

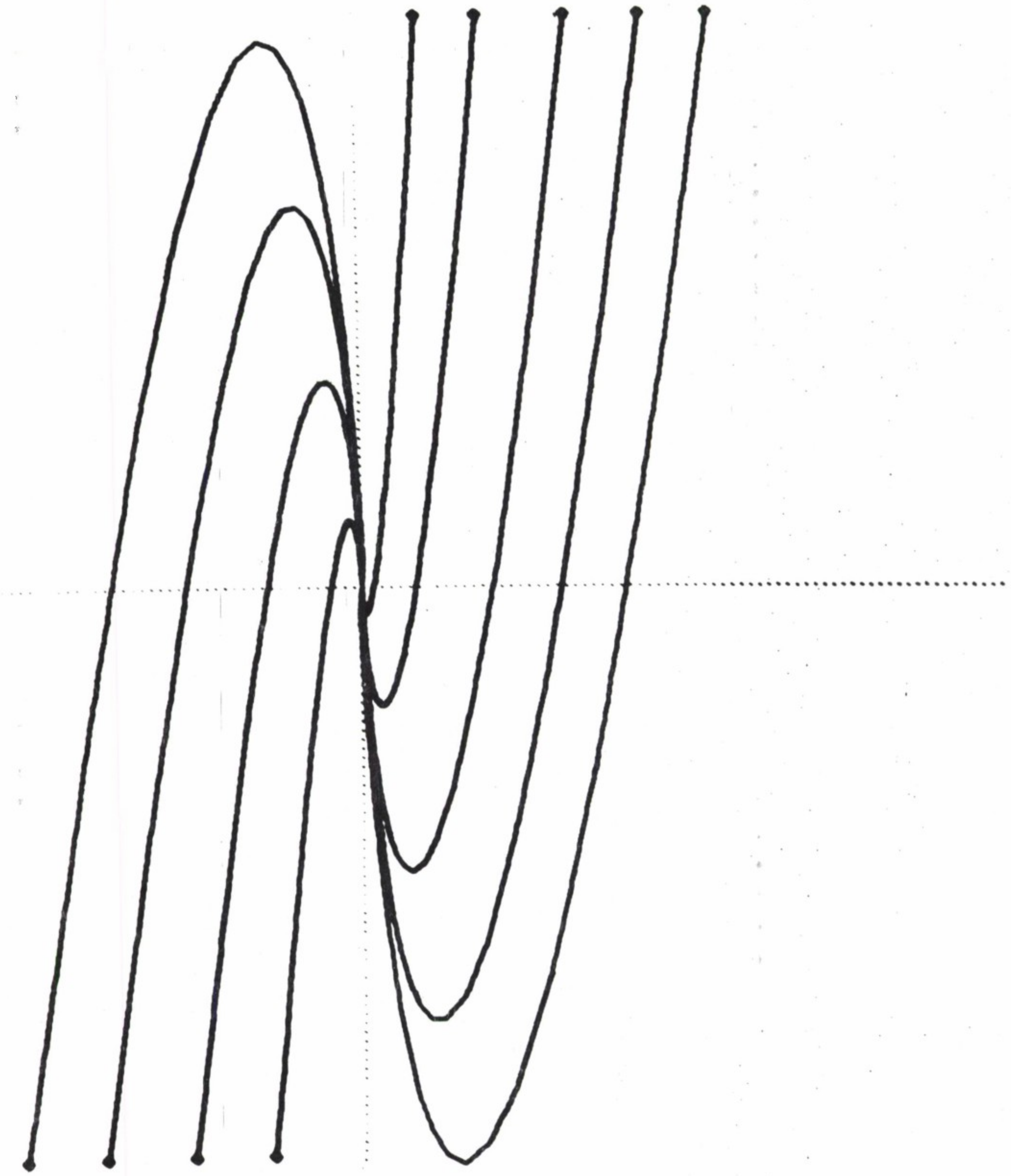


Figure III.10.12

SECTION IV

INCREMENTAL METHODS FOR HIDDEN-LINE ELIMINATION

(IV. 1) Introduction

Producing pictures with hidden lines elimination (HLE) in real time is one of the biggest challenges in computer graphics. For some years there have existed some programs for generating pictures with HLE, like [7], [8], [9], [10], [11], [12], [13], and [14]. All of them are many orders of magnitude away from real time computation. There exists only one system which produces images, with HLE in real time. This is a special-purpose system which was designed and built by GE for NASA, [15], at a cost of about $3, 000, 000.

Recently, John Warnock of the University of Utah devised an algorithm [16], which for the first time brings some hope for economical real time HLE. The programs mentioned before use a straightforward brute force algorithm which checks each possibly-seen entity against each possibly-hiding entity. This checking of "all against all" makes the required amount of computation to be proportional to the square of the number of defined objects. The Warnock algorithm (WA) deals with the objects according to the order in which they are located in the picture, not according to their arbitrary order in the data structure. The amount of computation required by the WA grows at a rate less then the square of the complexity. In (IV. 2) we describe briefly the WA. In (IV. 3) we show an incremental approach to the WA which eliminates redundant computation by organizing the computation in such a way which saves at any step the information which can be used in later steps. It is estimated[1] that this approach can cut the required amount of computation for a typical simple figure (like the picture of a house), by an order of magnitude. A most time-consuming problem, which is at the core of most HLE programs, is finding whether a given point is inside of a given polygon. In (IV. 4) we show an incremental method for solving this problem.

---

[1]By Mr. Warnock and others.

(IV. 2) A Brief Discussion of the Warnock Algorithm

The WA has two basic logical units, a "control-unit" and a "looking-unit". The control unit chooses a portion of the picture, which I call a window, and tells the looking-unit to work on it. The looking unit considers this specified subpicture (the "window") and finds what is seen in this window, or announces a failure to find it, due to a too complex situation. In case of success, the control unit outputs the results to some display system. In case of failure, the window is put on a list of "unsolved-windows". Later each unsolved window is subdivided into some smaller windows, each of which is given to the looking-unit for consideration. The looking-unit never announces failure when the size of the window has been reduced to a single resolution unit of the display. This guarantees that the algorithm is always completed in a finite number of steps. The process continues until all the portions of the picture are solved.

There are many possible versions of the algorithm, depending on the complexity of the looking-unit and the control-unit. The simplest looking-unit is one which announces failure if any vertex (point) or edge (line) is seen in the window, i. e., if its projection lies inside the window, and no opaque surface is between it and the observation point. If there is some surface which hides everything else in the given window, then the looking-unit tells the display system that the "color" of the window is the "color" of this polygon. If nothing is seen in the window, then the looking-unit announces the window to have the "color" of the background (mostly black). The simplest control-unit is the one which always divides windows into four identical quarters.

A more sophisticated looking-unit can handle more complex situations, like a window with a single line in it. Handling these windows, without announcing failure saves a considerable amount of computation; the more sophisticated the looking-unit is, i. e., the more complex situations can be handled without announcing failure, the less subdivision is required.

The simplest control-unit always subdivides windows in the middle; however, it is advantageous to subdivide windows, at some complex point, like a vertex. By subdivision at a complex point, the complexity is most often reduced to a simpler case.

There is a wide range of looking-units and control-units which can be used with the algorithm. A beautiful property of the algorithm is the independence of the structure of the control-unit and the looking-unit.

(IV. 3) The Incremental Approach to the Warnock Algorithm

The basic idea of the incremental approach is reordering the computational procedure in such a way that essential information can be saved, and used in later steps. Storing this information eliminates the need to ask the most time-consuming questions more than once.

Let $W = \{W_1, W_2, W_3, W_4\}$ be a window which is divided into the 4 windows: W_1, W_2, W_3, W_4 as shown below:

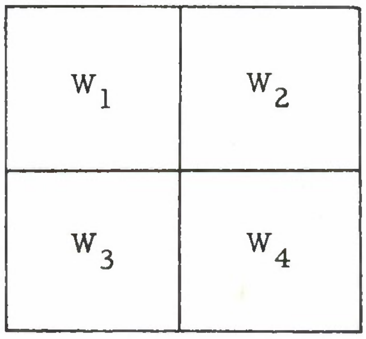


Figure IV. 3. 1: A window subdivided into 4 subwindows

Let the index 1 always denote the upper left corner, 2 for the upper right corner, 3 for the lower left and 4 for the lower right one. If W_i is subdivided we have $W_i = \{W_{i1}, W_{i2}, W_{i3}, W_{i4}\}$. If W_{ij} is subdivided we have $W_{ij} = \{W_{ijk} | k = 1, 2, 3, 4\}$ and so on.

Let us consider the following window, W:

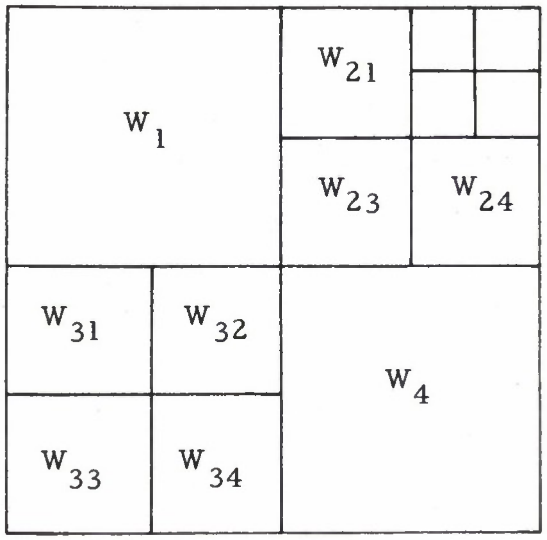


Figure IV. 3. 2: Subwindows divided into subwindows

The window W, with its subdivisions, as shown in the above figure, can be represented by:

$$W = \{W_1, W_2, W_3, W_4\}$$
$$W_2 = \{W_{21}, W_{22}, W_{23}, W_{24}\}$$
$$W_3 = \{W_{31}, W_{32}, W_{33}, W_{34}\}$$
$$W_{22} = \{W_{221}, W_{222}, W_{223}, W_{224}\} \quad .$$

By substitution we get the following representation:

$$W = \{W_1, \{W_{21}, \{W_{221}, W_{222}, W_{223}, W_{224}\}, W_{23}, W_{24}\},$$
$$\{W_{31}, W_{32}, W_{33}, W_{34}\}, W_4\} \quad .$$

The graphical representation of the above is as follows:

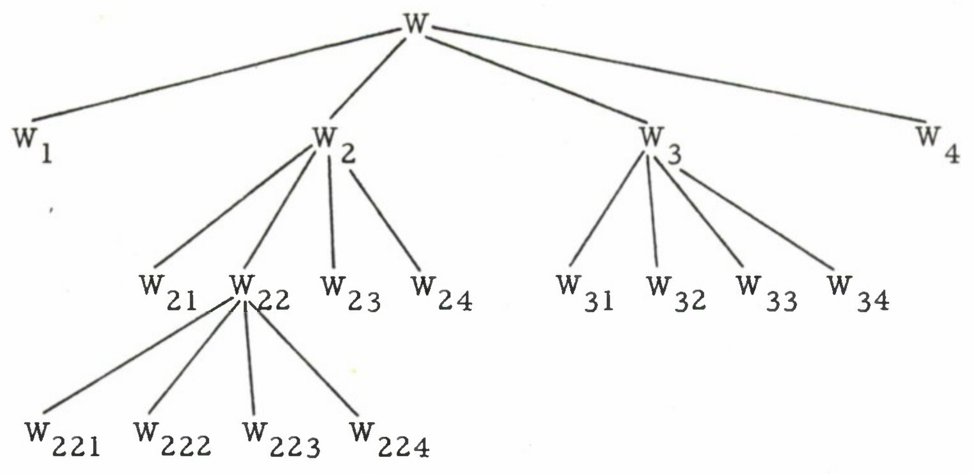


Figure IV. 3. 3: A tree representation of Figure IV. 3. 2

Each node $\{W_{ij\ldots}\}$ corresponds to a window. A window which is declared to be too complicated is represented by a node with 4 sons, a solved window is represented by a terminal node. The task of the control-unit is to decide which node should be considered next. Tree scanning can be done in many ways. One of these is the "constant-depth walk"[1]. In this example, a constant-depth-walk visits the nodes in the following order:

$$\{W, W_1, W_2, W_3, W_4, W_{21}, W_{22}, W_{23}, W_{24}, W_{31}, W_{32}, W_{33}, W_{34}, W_{221}, W_{222}, W_{223}, W_{224}\} .$$

This order is achieved by starting with W, and keeping a FIFO (First In, First Out) stack for the unsolved windows. This ordering implies that windows are processed in the order of their sizes.

The incremental approach is to scan the problems-tree by a prefix-walk[2], which is achieved by keeping the unsolved problems in

---

[1]See [17].

[2]See [17].

a LIFO (Last In, First Out) stack, instead of a FIFO stack. In our example, the prefix walk is:

$$\{W, W_1, W_2, W_{21}, W_{22}, W_{221}, W_{222}, W_{223}, W_{224}, W_{23}, W_{24},$$

$$W_3, W_{31}, W_{32}, W_{33}, W_{34}, W_4\} \quad .$$

The length of this LIFO stack cannot exceed N, the maximal number of subdivisions which is the base-2 logarithm of the scope size (e.g., N = 10 for a 1024x1024 scope). There are never more than N open, unsolved problems, which reduces the amount of storage required.

The most time-consuming problem is finding the relation between a given polygon and a window. This relation is one of the following:

(a) the polygon is outside the window,

(b) the polygon surrounds the window,

(c) some edges of the polygon intersect the window, but no vertex is inside the window,

(d) some vertices of the polygon are inside the window.

If the polygon P is outside the window W (property (a)) then P is outside any subwindow of W.

If P surrounds W, it also surrounds any subwindow of W.

If P has no vertex inside W, it has no vertex in any W_i, but P may be outside some W_i, or surrounds some W_j.

If P has any vertex inside W, then all the above relations can hold between P and W_i.

This can be represented in the following diagram:

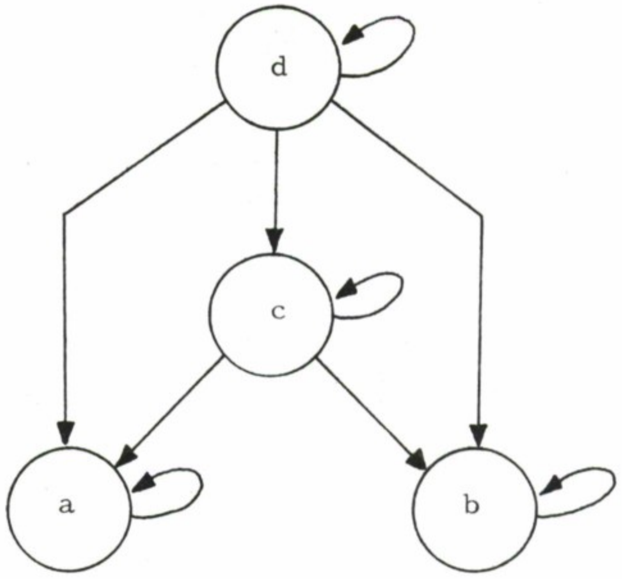


Figure IV. 3. 4: The relations between the polygon/window relations

Outsideness and surroundedness are terminal because they do not change for descendants. When any terminal-relation exists between a window W and a polygon P, this relation is known to hold between P and any subwindow of W. The classification of the relations into terminal and non-terminal is the key to the incremental approach, and to the prefix-walk. When a relation is found between a window W and a polygon P, the relation is stored. If this is a terminal relation, then it is known to hold between P and any descendant of W (i. e., subwindow). This saves the repeated checking for the relation between polygons and windows. If, when a polygon is considered, there is no terminal relation between it and any ancestor of the window, then the program should look for it. The relation between windows and polygons are stored in a stack which is associated with the polygons. The length of the stack is N (as defined before) and each entry can be expressed by 2 bits. Hence, by associating a stack of size 2N bits with each polygon we can convey its relation with any window, to all of its descendants. This works only for prefix walk, and not for constant-depth walk, because the number of open-problems on constant-depth walk can be as high as 4^{N-1}, and associating $2\cdot 4^{N-1}$ bits to each polygon is very likely to be infeasible. The prefix walk contains never more than N open-windows. A flow chart of the algorithm, using the simplest looking-unit and the simplest control-unit is shown in Figure IV. 3. 5.

(IV. 4) <u>The Incremental Solution for the In/Out Problem for Polygons</u>

In the core of all HLE programs there is a solution for the in/out problem. The computation required for solving this problem is a great portion of the total computation required for HLE. It is very important, therefore, to improve the technique for solving this problem. We will show an incremental solution for this problem, and outline the hardware implementation of this technique.

The In/Out problem for polygons can be stated by:

(Q1): "Is the point P_0, inside the polygon P?"[1]

Another form of this problem is:

(Q2): "Does the polygon P surround the point P_0?"

A generalization of (Q2) is the following:

(Q3): "How many times does the polygon P surround the point P_0?"

If the edges of the polygon P do not intersect themselves, then (Q2) and (Q3) are equivalent. But this is not necessarily the case.

There are two techniques for solving this problem. One is based on angles summation and the other on intersections of the polygon and a line from the point P_0 to infinity. The angle summation technique works as follows. Define: $\theta_i = \measuredangle P_iP_0P_{i+1}$ where $-\pi < \theta_i \leq \pi$. Compute

---

[1] An N-vertices polygon is given by the ordered set of its vertices $\{P_i | i = 1, 2...N\}$. We define $P_{N+1} = P_1$, such that the edges are $E_i = \overline{P_iP_{i+1}}$ for i = 1, 2. . . N.

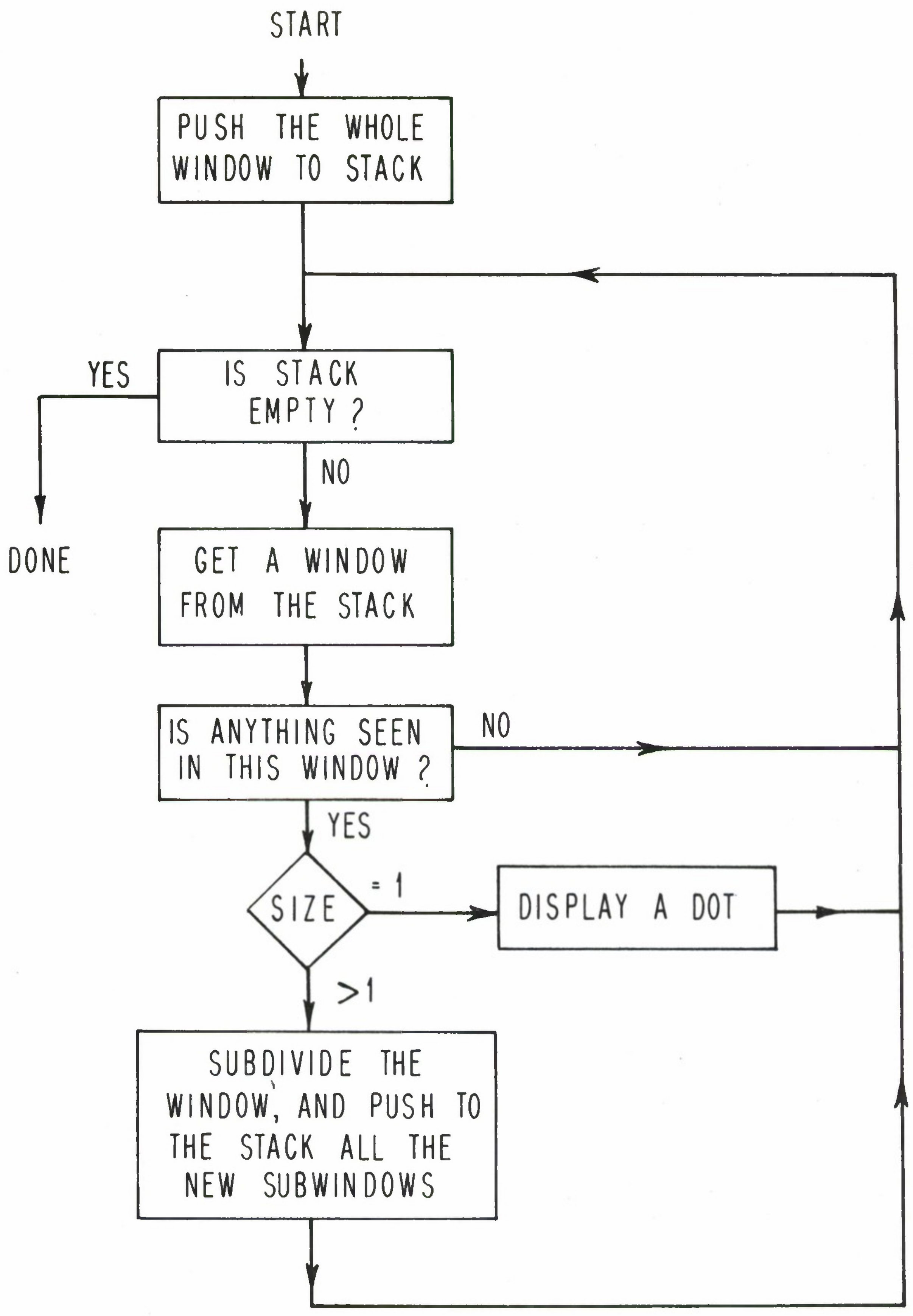

START
PUSH THE WHOLE WINDOW TO STACK
IS STACK EMPTY ?
YES
DONE
NO
GET A WINDOW FROM THE STACK
IS ANYTHING SEEN IN THIS WINDOW ?
NO
YES
SIZE
= 1
DISPLAY A DOT
>1
SUBDIVIDE THE WINDOW, AND PUSH TO THE STACK ALL THE NEW SUBWINDOWS

Figure IV.3.5 : The simplest Warnock algorithm for line drawing.

$$\sigma = \sum_{i=1}^{N} \theta_i .$$

Here σ is the total angular change of the polygon P around the point P_0. If P surrounds the point n times then $\sigma = 2\pi n$. Note that n and σ may be positive or negative, depending on the sense of the polygon. Note that according to Euler's theorem

$$n = \frac{\sigma}{2\pi} = \frac{1}{2\pi i} \oint_P \frac{dz}{z} .$$

P_0 is outside P if $n = 0$, and inside otherwise. Hence $\sigma = 0$ if P_0 is inside, and $|\sigma| > 2\pi$ if P_0 is outside.

In implementing this technique one computes

$$\sigma = \sum_{i=1}^{N} \theta_i ,$$

and announces insideness if $|\sigma| \leqslant \pi$, and outsideness otherwise. The computation of the angles $\{\theta_i\}$, is very expensive timewise although each θ_i has to be computed only with accuracy of $\epsilon = \frac{\pi}{N}$.

The other technique, the ray-method works as follows: we define a ray from P_0 to infinity. For simplicity in this discussion we assume that $P_0 = (0, 0)$. Let the ray be $R = \{0 < x < \infty, y = 0\}$. We check all the edges of P to find if they intersect R. Let n be the count of intersections. P_0 is announced to be outside, if n is even, and inside otherwise.

The angles-summation method can answer the problem (Q3). The ray method can answer only (Q2) for certain cases, but is much faster since intersecting the $\{E_i\}$ with R is simpler than finding the $\{\theta_i\}$.

Consider, for example, a polygon which describes the digit "4", as illustrated in Figure IV.4.1. Consider the points A, B, C and D. Applying both methods to these points shows that A is inside but B and D are outside. However the two methods do not agree about the point C. According to the angle-summation method C is inside (twice) but it

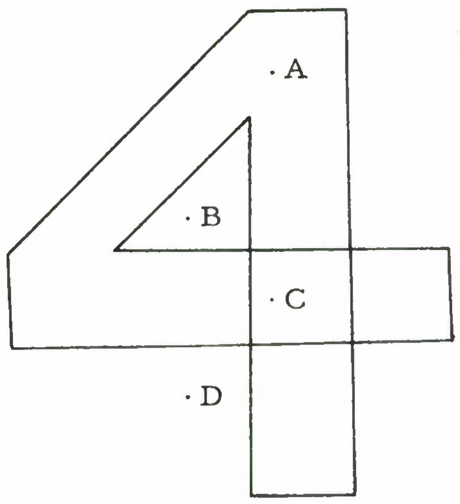


Figure IV.4.1: A polygon describing the digit "4"

is outside according to the ray method. The technique described below is as powerful as the angle-summation method, and is at least as fast as the ray method.

The order of the vertices in the definition of P implies a direction (clockwise or counterclockwise) for the polygon. This direction is used to associate a value (plus or minus one) to each intersection of edges and the ray R. The algebraic sum of the values of the intersections is the number of times in which P surrounds the point P_0. For simplicity again let $P_0 = (0, 0)$. We choose the ray to be $R = \{0 < x < \infty, y = 0\}$. The edge E_i intersects R only if y_i and y_{i+1} have different signs[1]. The value of the intersection is defined to be +1 if $y_i \geq 0$, and -1 if $y_i < 0$. If E_i does not intersect R the value of their intersection is defined to be zero.

By considering only the signs of (x_i, y_i) and (x_{i+1}, y_{i+1}) we can obtain the value of the intersections for the following cases:

[1]Note that zero is always considered as a positive number.

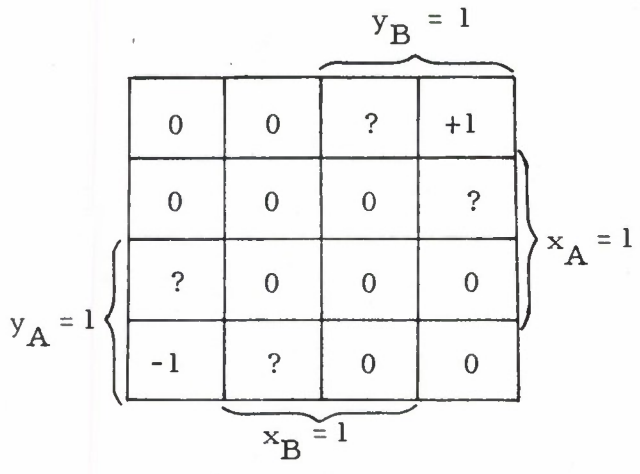
Case	x_i	y_i	x_{i+1}	y_{i+1}	V(V = value)
(a)	-	Φ[1]	-	Φ	0
(b)	Φ	+	Φ	+	0
(c)	Φ	-	Φ	-	0
(d)	+	+	+	-	+1
(e)	+	+	+	-	-1
(f)	+	-	-	-	+1 or 0
(g)	-	+	+	-	+1 or 0
(h)	-	+	+	+	-1 or 0
(i)	+	-	-	+	-1 or 0

Cases (a)-(e) cover 12 out of the 16 possible sign combinations. The other 4 combinations are (f)-(i). The intersection values in these cases have to be found by using arithmetic operation. A possible set of rules which resolves these cases is the following.

Cases (f) and (g): if $x_1y_2 < x_2y_1$ then V = +1, otherwise V = 0.

Cases (h) and (i): if $x_1y_1 > x_2y_2$ then V = -1, otherwise V = 0.

For the general case, where P_0 is not the origin, these cases are described by the following Karnaugh map:



0	0	?	+1
0	0	0	?
?	0	0	0
-1	?	0	0

$x_A = 1$ if $x_i < x_0$

$y_A = 1$ if $y_i < y_0$

$x_B = 1$ if $x_{i+1} < x_0$

$y_B = 1$ if $y_{i+1} < y_0$

Figure IV.4.2: A Karnaugh map for the signs conditions

[1] "Φ" for don't care.

The (?) indicate the cases in which more computation is required for finding V.

The In/Out problem as described before deals with the relations between a polygon and a point. The WA deals with relations between polygons and windows. Since a point may be considered as a degenerate window, solving the relations between windows and polygons is a more general problem then solving the relations between polygons and points, as described before. The technique in this case is to check all the polygon vertices $\{P_i\}$ for intersections of the edges $\{E_i\}$ and the window sides, and for being inside the window, W. If some vertices are inside W then this is a case of complexity (d) as defined in II. 3. If no vertices are inside W, but some edges intersect W then this is a case of complexity (c). If no vertex is inside W, and no edge intersects W then this is either case (a) of the window being outside the polygon, or case (b) of the polygon surrounding the window W. It is still to be resolved whether this is case (a) or (b).

The same relation (outsideness or surroundedness) which holds between the polygon and the window W holds also between the polygon and each of the window corners. Hence, it can be found for W by finding it for any one corner using the method described earlier. However, this computation which scans all the $\{E_i\}$ can be combined with checking the $\{E_i\}$ for intersecting the window sides, as both computations require common calculations and comparisons.

It is feasible to build a special-purpose hardware, using this technique, which scans once all the $\{p_i\}$ and announces which portions of the $\{E_i\}$ are inside the window W, if any. If none, it announces an outsideness or surroundedness relation. This "box" can get the required information $\{P_i\}$ by using a direct channel to some memory and work in parallel and independent of the host computer.

This hardware is primarily a clipping-divider [19] which clears some n-register when it starts working on a new polygon, and updates it when working on the polygon edges. After the last edge has been processed, the n-register contains the number of times in which the polygon surrounds some window corner. If there are no intersections then the n-register shows how many times P surrounds W. For each edge, E_i, the clipping divider compares the coordinates of P_i and P_{i+1} with the window coordinates. These comparisons are the inputs needed for updating the n-register according to the table of Figure IV. 4. 2. In the 4 complicated cases (which are indicated by - ?- in the table) the clipping-divider cannot make a trivial decision either and finds a point P_x, on the edge E_i, which together with each vertex is used for two simple decisions. This point P_x, enables the In/Out logic to make two decisions according to the table. This may be illustrated by the example in Figure IV. 4. 3.

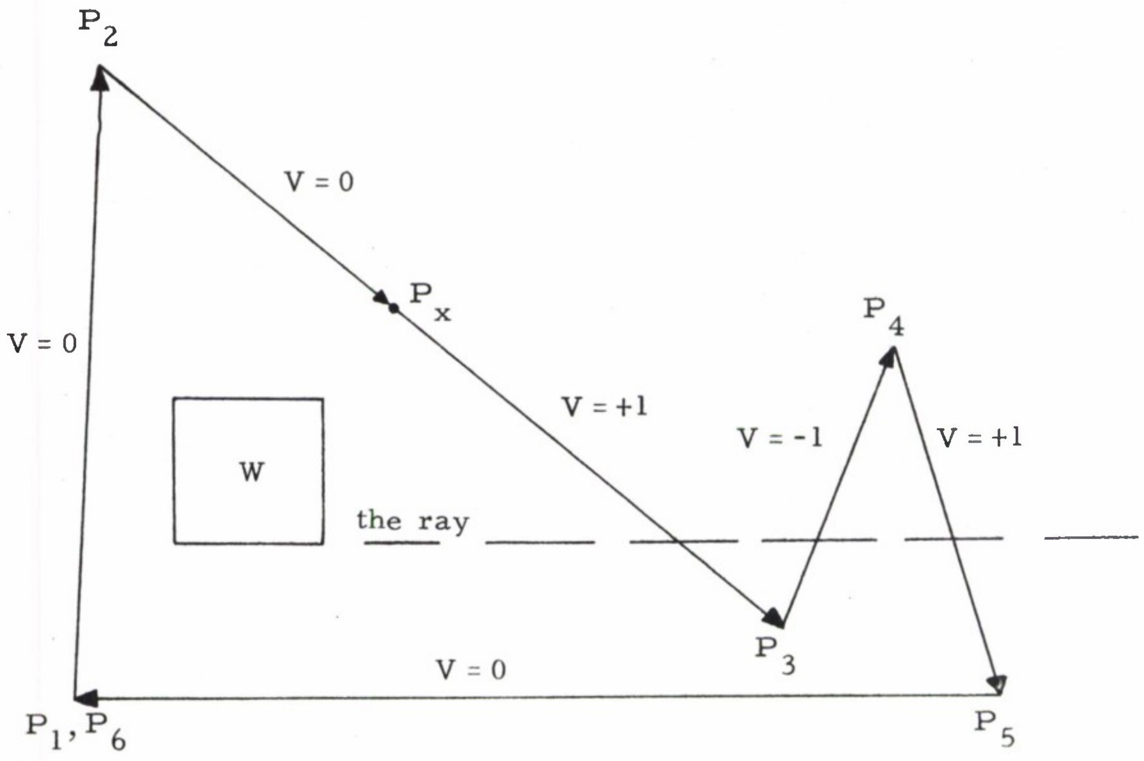


Figure IV. 4. 3: Examples for window/line relations

SECTION V

PRODUCTION OF HALF-TONE IMAGES IN REAL TIME

(V. 1) Introduction

Most computer display systems use calligraphic (line drawing) scopes. Real world pictures (like the ones produced by cameras and TV) are half-tone images. Production of real time half-tone images, at a feasible cost[1] has been for a long time, a challenge for computer graphics.

The reason why this challenge was not met for a long time, is the huge number of points[2] which have to be intensified for displaying a half-tone image. The bigger this number is, the less time is available per point. The short time which is available per point, might be enough for intensity calculation, but not for random beam positioning. For this reason, any half-tone image production must be based on some standard scanning pattern with the beam intensity-modulated according to the image to be displayed.

Using a scan technique for displaying introduces a sorting problem. The scanning requires a certain order in which the data must be arranged, while the data is generated by the computer according to some order which does not necessarily agree with the scan order. This need for sorting creates what is known as the "scan-conversion" problem. In this section we describe two methods for producing a half-tone image. One, which is based on a TV scan by using Cartesian coordinates, is described in (V. 3). The other method, which is based on a radar scan using polar coordinates, is described in (V. 4).

---

[1]There exists a system in the MSC (NASA) in Houston, Texas, which was built by GE for about $3,000,000. [15].

[2]For a 512×512 raster scope, with refreshing rate of 30 times per second, about $7.5 \cdot 10^6$ points have to be intensified per second, which is about 133 νsec per point.

Both techniques use the same two functional units, a line-controller (LC) and a line-controller-interface (LCI). The LC generates the scanning[1] and modulates the beam intensity according to its position. The LCI prepares information for the LC. There is a trade-off between the complexity of the LCI and program which feeds it. The simpler the LCI is, the more has to be computed by the program, and vice versa. The main feature which makes these techniques feasible is supplying information only about the points in which the intensity rule changes along scan lines, instead of supplying information about each point.

In (V. 2) below we describe the general LC, which might work for any 2D-coordinates system.

(V. 2) <u>Line Controller for Generalized Coordinates</u>

As the scope face is a 2D surface, there are many ways to map it by using some 2D-coordinate system, which is not necessarily cartesian.

Let (u, v) be the coordinate system which is used to cover the scope face. Without loss of generality we can assume that the displayed area is:

$$\{(u, v),\ 0 \leqslant u \leqslant 1,\ 0 \leqslant v \leqslant 1\}\ .$$

This coordinate system is useful only if there exists a fast technique for generating the scan lines: $\{(u, v_j),\ 0 \leqslant u \leqslant 1\}$ for all values of v_j. Let us assume that this is the case.

A (u, v)-scan image is the following sequence of intensities:

$$\{\{i(u, v) \mid 0 \leqslant u < 1\}\ ,\quad 0 < v < 1\}$$ [2]

The image is generated by generating the sequence

$$\{v_0 = 0,\ v_1,\ v_2 \ldots,\ v_N = 1\}$$

synchronized with displaying a frame, and supplying the scan lines:

---

[1] If the scanning is generated in an analog technique, the LC is synchronized with the scanning.

[2] i hereafter is for 'intensity'.

$\{i(u, v_n),\ 0 < u < 1\}$

for each v_n.

The LCI prepares a list of positions at which the intensity rules change along the scan line $v = v_n$. For simplicity we assume that all intensity rules are just constant intensity in some area. We will describe other intensity rules below. An intensity profile like the following:

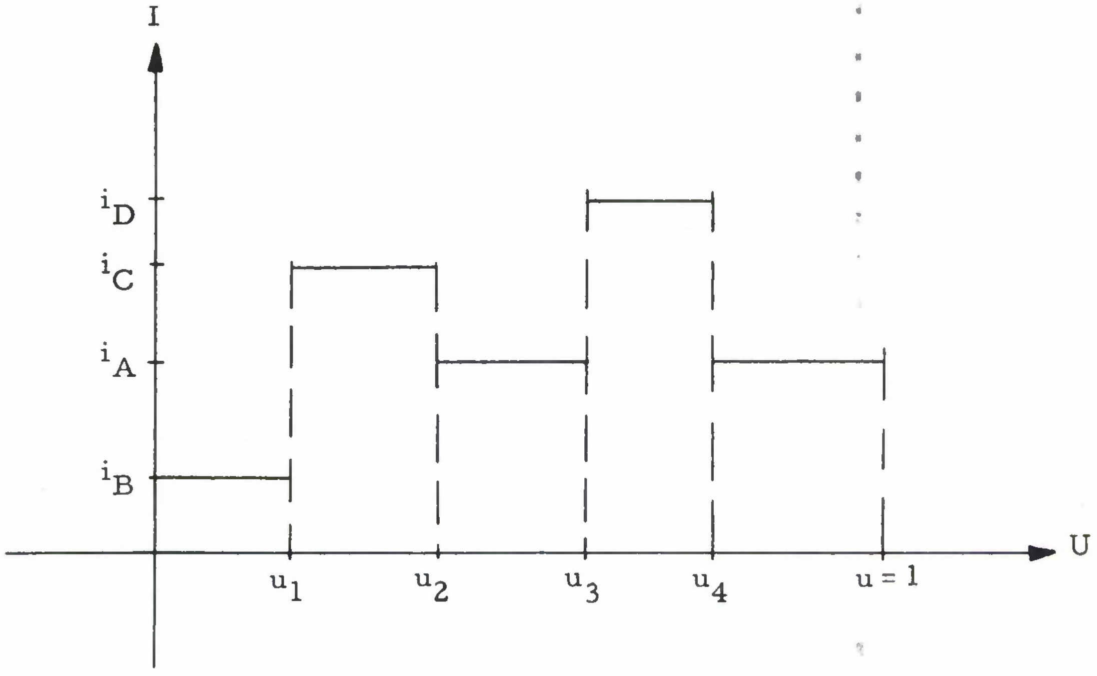


Figure V. 2. 1: Intensity profile

is described by the following list:

$$(0, i_B)(u_1, i_c)(u_2, i_A)(u_3, i_D)(u_4, i_A) \ .$$

The intensity which is associated with each u_j is displayed between $u = u_j$ and $u = u_{j+1}$ (if there is no u_{j+1}, then $u_{j+1} = 1$).

When the LC starts displaying a frame it clears its v-register (VR) its u-register (UR) and loads its intensity register (IR) from the first piece of data. The UR and the VR are used for controlling the beam position and generating the scan, the IR modulates the displayed intensity.

The next data (u_i, i_j) is loaded into the next-u-register (NUR) and into the next-intensity-register (NIR). For displaying each point, the UR is incremented by some δU, and the IR sent to the scope. If the incremented UR is equal to the content of the NUR, then the NIR is jammed into the IR for the next point, and new data is entered into the NUR, and the NIR. When the UR reaches the value $u = 1$, the VR is incremented by δV, and the UR and the IR are loaded from the NUR (which contains $u = 0$) and the NIR, while new data is placed in them. When the VR reaches $V = 1$ it is the end of the frame. Note that the magnitude of δu determines the resolution along scan lines, as well as the scan velocity, and the magnitude of δv determines the number of lines per frame.

The organization of this LC is illustrated in Figure V.2.2. If the shade between successive data points $\{u_i\}$ is not constant, but changes linearly with u, (i.e., $\frac{\partial i}{\partial u}$ is constant) then 2 more registers are needed, the $\frac{\partial i}{\partial u}$ - register (IUR) and the next - $\frac{\partial i}{\partial u}$ - register (NIUR). The organization of this LC is illustrated in Figure V.2.3.

The generalization for higher degrees of shading rules is obvious, and would follow the polynomial generation scheme described in (II.4).

Note that if δU is of the form 2^{-M}, then adding δU is only a counting operation, and the $U = 1$ condition is recognized by a carry from the most significant bit, or by the change of the UR to zero. When this condition is detected, there is no need to clear the UR by loading the NUR to it, as the binary representation of $u = 1$ can be $u = 0$. The above remarks hold also for δV.

Note also that the process of scanning which involves repetitive addition of the δU to the UR, and of the δV to the VR is a simple integration process which can be carried out by analog means.

(V.3) Production of TV-scan Images

The technique described below was first suggested by Professor D. Evans of the University of Utah. The version which is described below, and which was simulated on a PDP-1 computer is a result of many discussions with Professor I. Sutherland.

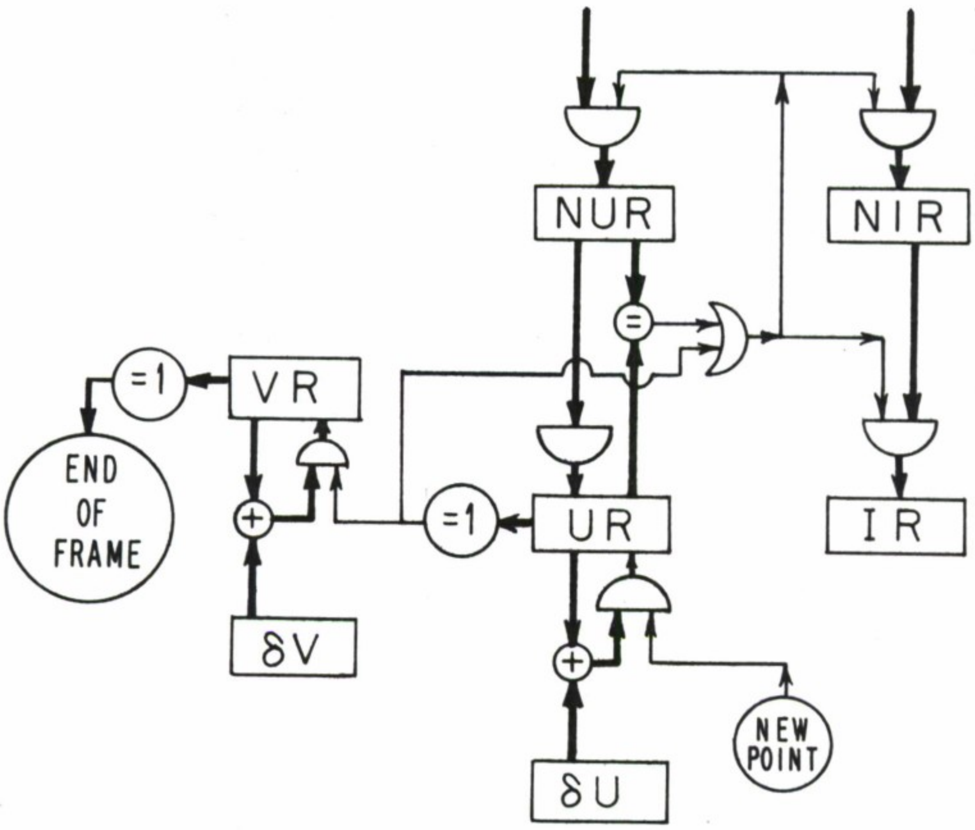


Figure V.2.2 : A shader for constants intensities

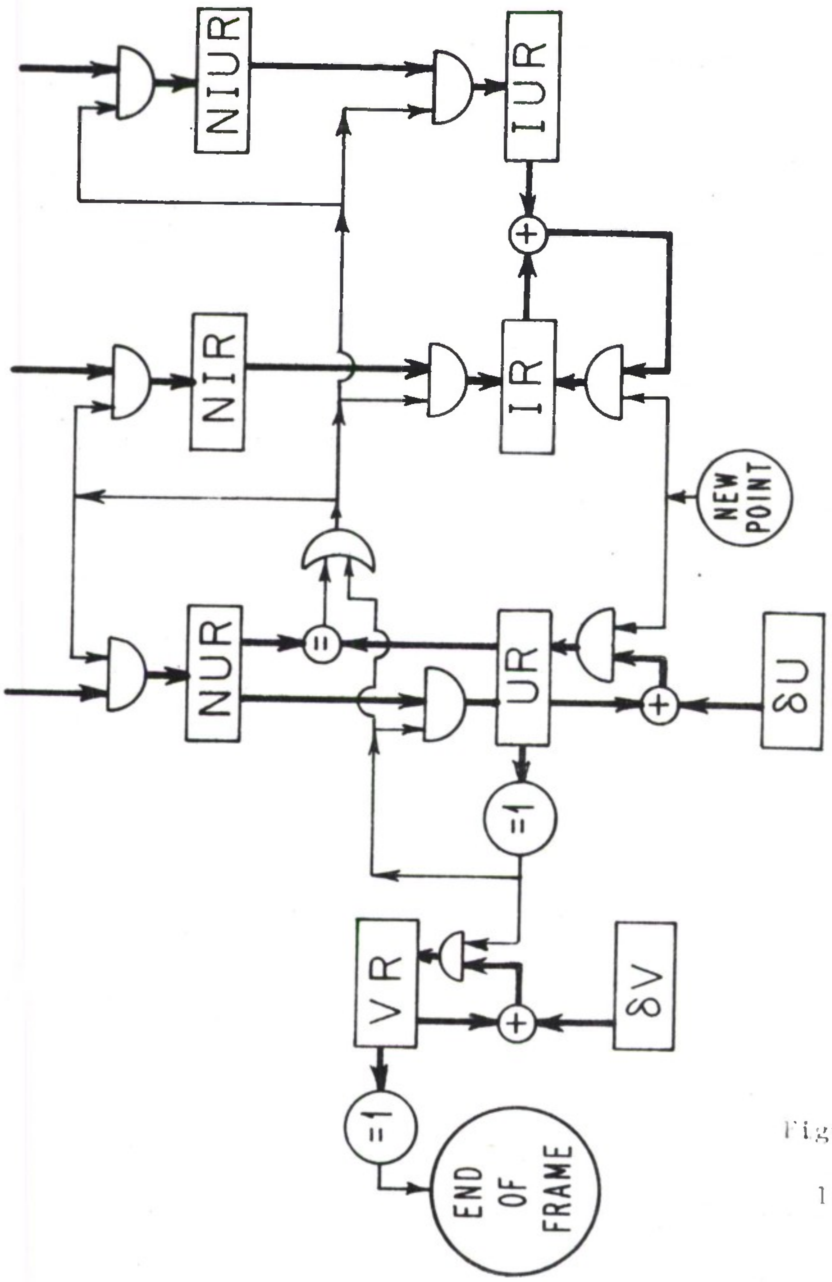


Figure V.2.3 : A shader for linear intensities.

The TV-scan is based on cartesian coordinates where the (u, v) of (IV. 2) are replaced by (x, y). The LC is exactly as described there, however, the LCI can take advantage of the properties of cartesian coordinates, and perform some operations which otherwise have to be done by the program.

The LCI is able to accept a description of a linear shading rule line, and produce by itself the data about all the intersections of this line with scan lines. This is done by the following scheme; consider the line $\overline{P_1P_2}$ which initiates[1] the following linear shading rule:

$$i(x, y) = i(x_1, y_1) + (x - x_1)\frac{\partial i}{\partial x} + (y - y_1)\frac{\partial i}{\partial y} \quad .$$

For the scan line $y = y_n$ where $y_1 \leq y_n \leq y_2$ the LC needs the following information:

$$(x_n, i_n, \frac{\partial i}{\partial x})$$

where $x_n = x_1 + (y_n - y_1)\frac{x_2 - x_1}{y_2 - y_1}$ which is the x coordinate of the intersection of $\overline{P_1P_2}$ with the scan line $y = y_n$, and i_n is:

$$i_n = i(x_n, y_n) = i(x_1, y_1) + (x_n - x_1)\frac{\partial i}{\partial x} + (y_n - y_1)\frac{\partial i}{\partial y}$$

which is the intensity at (x_n, y_n), the intersection point.

For the next scan line $y = y_{n+1} = y_n + \delta y$ the LC needs the following information:

$$(x_{n+1}, i_{n+1}, \frac{\partial i}{\partial x})$$

where

$$x_{n+1} = x_1 + (y_{n+1} - y_1)\frac{x_2 - x_1}{y_2 - y_1} = x_n + \delta y \cdot \frac{dx}{dy} \quad ,$$

and

[1] i. e., this shading rule applies on the right of this line, while the scanning goes left to right.

$$i_{n+1} = i(x_1, y_1) + (x_{n+1} - x_1)\frac{\partial i}{\partial x} + (y_{n+1} - y_1)\frac{\partial i}{\partial y}$$

$$= i_n + (\frac{\partial i}{\partial x}\frac{dx}{dy} + \frac{\partial i}{\partial y})\delta y = i_n + \frac{di}{dy}\delta y \quad .$$

Hence, the LCI can generate the $\{x_n\}$ and the $\{i_n\}$ for successive scan lines if it is given $\delta y \cdot \frac{dx}{dy}$ and $\delta y \cdot \frac{di}{dy}$, in addition to the data about P_1, the first intersection with the scan lines. It needs also some information which indicates where P_2 is, to terminate this shading rule.

To summarize: The LCI can feed the LC with all the needed data for a shading rule boundary if it has the following information for each shading rule boundary:

$$(x_1, y_1, \frac{dx}{dy} \cdot \delta y, \frac{dI}{dy} \cdot \delta y, i(x_1, y_1), \frac{\partial I}{\partial x}, n)$$

where (x_1, y_1) is the first intersection of the shading boundary with the scan lines.

$\frac{dx}{dy}$	indicates the slope of the boundary
$\delta y \frac{dI}{dy}$	the intensity change <u>along the boundary</u>, as a result of changing y
$I(x_i, y_i)$	the intensity at (x_1, y_1)
$\frac{\partial I}{\partial x}$	the intensity change <u>along the scan line</u>, as a result of changing x
$N = \frac{y_2 - y_1}{\delta y}$	the number of scan lines which intersect this boundary.

From this information it can generate for each scanning line the $(x_n, i_n, \frac{\partial i}{\partial x})$ which are needed by the LC.

The usefulness of this LCI depends on the nature of the picture. If it has many long lines which are intensity rule boundaries it pays off to build such an LCI, otherwise a simple output channel from the computer storage can serve as a LCI.

(V.4) Production of Radar Scan Images

The radar scan is generated by using the polar coordinates (r, ϕ) over the domain $0 < r \leqslant 1$, $0 \leqslant \phi < 2\pi$. The scan lines are the radii $\{(r,\phi),\ 0 < r \leqslant 1\}$, for all ϕ.

We will describe below a LC which uses the (r^2, ϕ) coordinates, but generates the (x, y) coordinates for the scan, as required by the orthogonal deflection mechanism of the CRT's. It uses r^2 rather than r, because mathematically it is easier to find the square of a distance between two given points rather than finding the distance itself.

The operation of this LC is along the lines described in (V.2) before, where u is replaced by r^2, and V by $\phi/2\pi$.

The (x, y) coordinates for (r, ϕ) are:

$$x = r \cos\phi$$
$$y = r \sin\phi \quad .$$

The $\cos\phi$ and $\sin\phi$ define the direction of the scan line, and are supplied by the LCI for each scan line. The 2 multiplications which are needed for the x and y calculation can be replaced by two additions because r is incremented linearly. The (x, y) generation can be generated by a scheme such as the one shown in Figure V.4.1. Another possible implementation is shown in Figure V.4.2.

Note that the r-register has no function in this implementation. Note also that if the r-register has m bits, and the $\cos\phi$ and $\sin\phi$ have n bits then the x and y registers need m + n bits.

Next we have to generate r^2 for the comparison of the u-register with the next-u-register (see V.2). The $\{r^2\}$ can be generated, of course, by a multiplication. But this multiplication can be replaced by an addition, by using:

$$r = \sum_{n=0}^{r-1} (2n + 1) \quad .$$

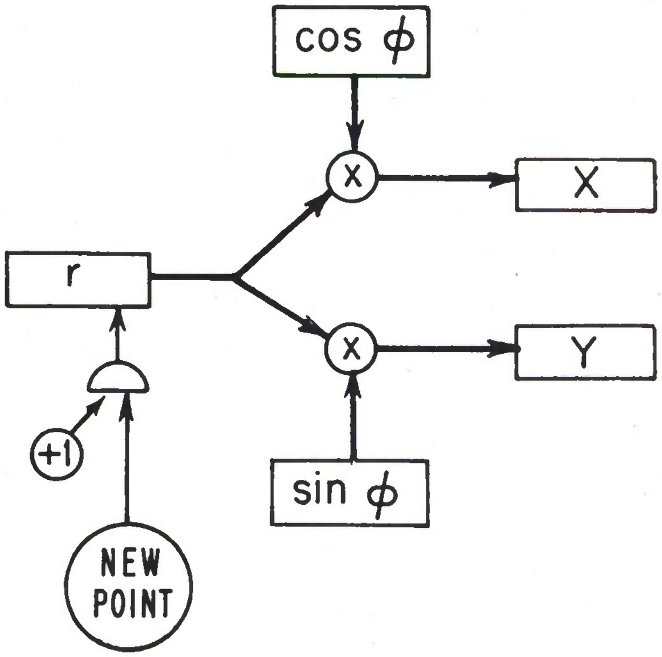


Figure V.4.1 : A scheme for generating X and Y.

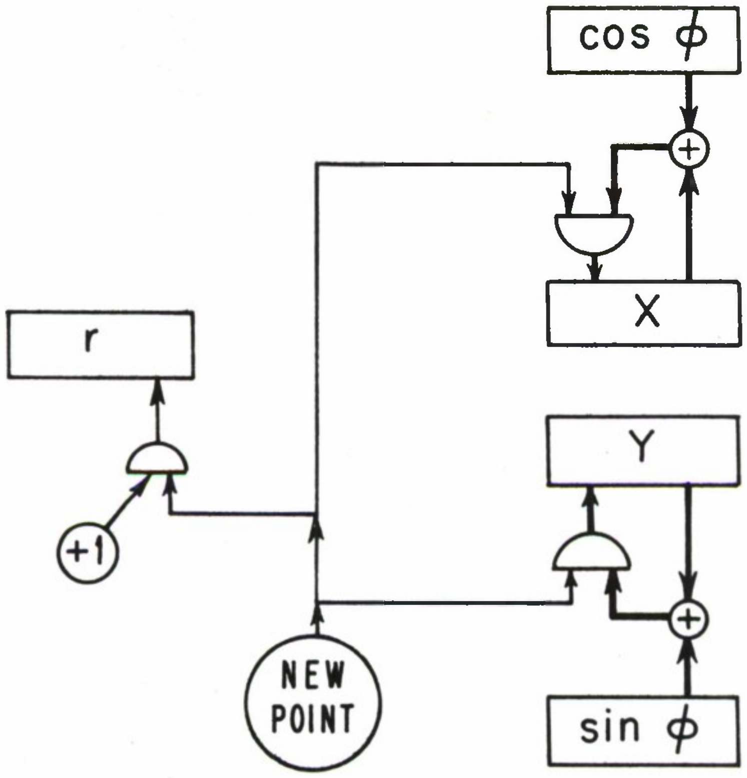


Figure V.4.2 : A multiplication free scheme for computing X and Y.

Note that adding 2n + 1 is implemented by adding n, shifted left one bit, and adding 1 to the least-significant bit.

Incorporating all the above to one system suggests the implementation shown in Figure V. 4. 3.

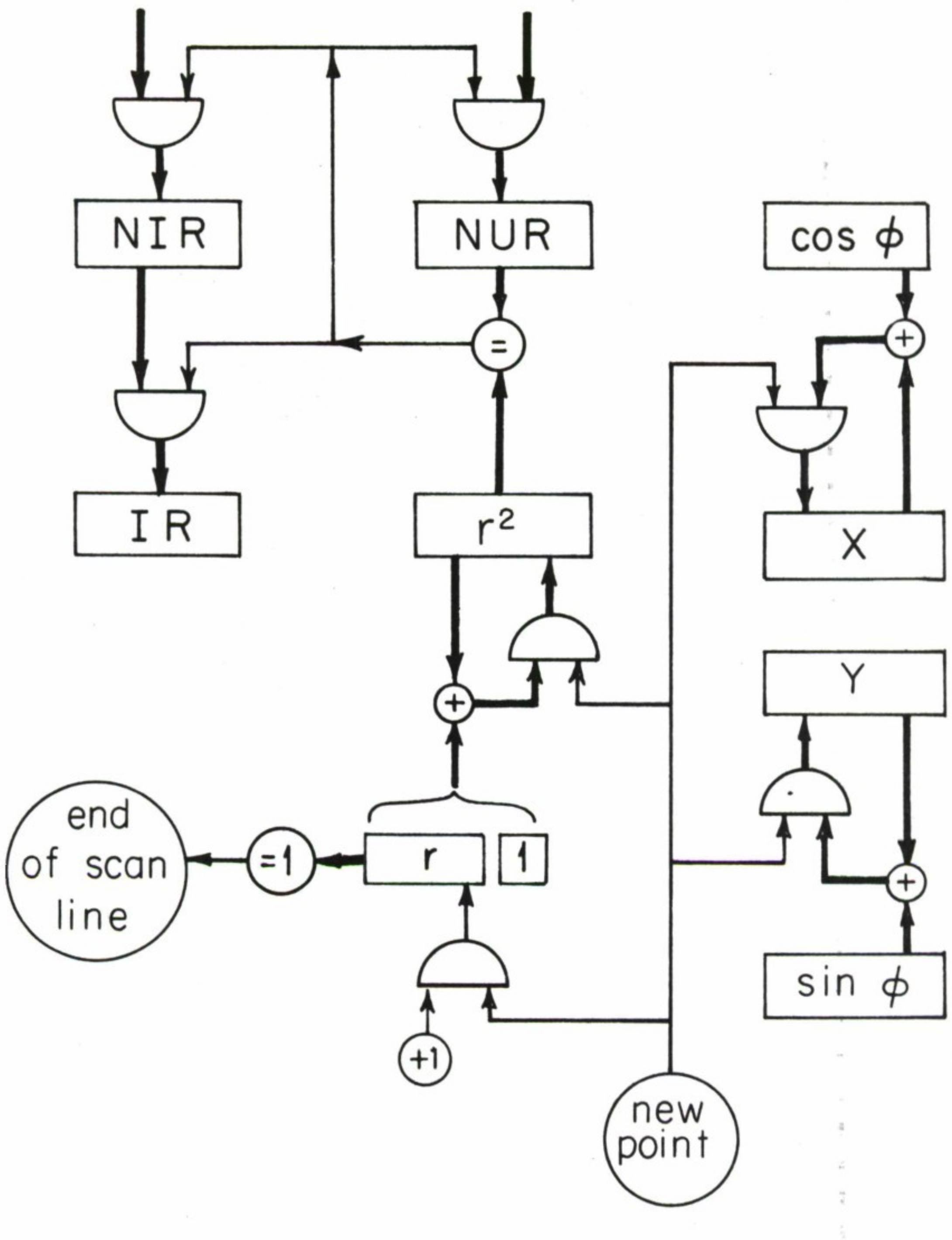


Figure V.4.3 : An $\{r^2, \phi\}$ shading system.

SECTION VI

FAST INTERSECTING OF LINES

(VI. 1) Introduction

Another problem in computer graphics is the line intersecting problem, i.e., finding if two given line segments intersect, and if so, where.

The mathematics required for solving this problem directly and classically is simple, in principle, and requires only some additions, multiplications and a division. However for some applications, the time required to do these operations precludes solving this problem in real time.

Our goal is to achieve a method for solving this problem very quickly which can be implemented by special-purpose hardware. It is impossible to separate the method from its hardware implementation.

In (VI. 2) below we show and describe the operation of the "Sutherland interpolator" which is the core of the line intersecting. A description of the Sutherland interpolator can be found in [19].

In (VI. 3) we show how the Sutherland interpolators are used for intersecting line segments.

In (VI. 4) we describe a device which can get (from some memory, via a channel) a string of contiguous line segments, and finds the intersections of these segments with a given segment.

(VI. 2) The Sutherland Interpolator

The basic function of the Sutherland Interpolator (SI) is finding the intersection of a given segment with one of the coordinates axes.

Let the segment be $\overline{P_A P_B}$, where $P_A = (x_A, y_A)$ and $P_B = (x_B, y_B)$. Finding the intersection of this segment with the y-axis is finding the value y_0, such that the point $P_0 = (0, y_0)$ is on the segment $\overline{P_A P_B}$, as shown in Figure VI. 2. 1:

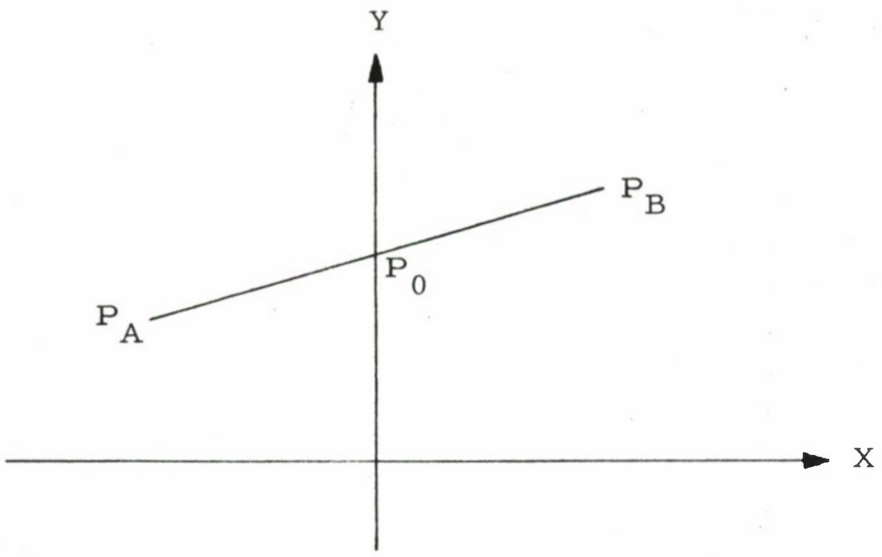


Figure VI. 2. 1: A line intersecting the y-axis

The solution P_0 satisfies:

$$\frac{y_B - y_0}{y_A - y_0} = \frac{x_B}{x_A} \quad .$$

It is self-evident that P_0 is between P_A and P_B or $P_0 \in (P_A, P_B)$ if and only if x_A and x_B have different signs, or $x_A x_B < 0$.

Eliminating y_0 from the above condition yields:

$$y_0 = \frac{x_B y_A - x_A y_B}{x_B - x_A} \quad .$$

Note that $x_A x_B < 0$ implies that $x_B - x_A \neq 0$.

The basic operation of the SI is finding P_m, the midpoint, of the given segment:

$$P = (x_m, y_m) \text{ where } x_m = \frac{1}{2}(x_A + x_B) \text{ and } y_m = \frac{1}{2}(y_A + y_B) \; .$$

If $x_m = 0$ then $y_0 = y_m$ and the problem is solved. Otherwise: if x_m has the same sign as x_A then the solution, P_0, is between P_m and $P_B : P_0 \in (P_m, P_B)$. If x_m has the same sign as x_B then $P_0 \in (P_m, P_A)$.

Note that $x_A x_B < 0$ implies that either $x_m x_A < 0$ or $x_m x_B < 0$, but not both.

To summarize: after one iteration of finding midpoint we either solve the problem (if $x_m = 0$) or reduce the problem to a similar one with the segment (P_m, P_A) or (P_m, P_B) instead of (P_A, P_B). Note that the width of the new segment is half of the width of the original segment.

By iterating this operation M times, replacing at each step one of the end-points by the midpoint, the width of the segment is reduced by the factor 2^{-M}. For N-bit numbers this process cannot last more than N steps until the problem is solved, either by hitting P_0 or by default when the width of the segment which contains P_0 is reduced to one unit.

Each step consumes one <u>add-time</u>. No time is required for the division by 2, which is achieved by the way the registers are connected, ignoring the least significant bit of the adder, and reproducing its most significant bit (the sign bit).

The total duration of this process for N-bits number is bounded by <u>N-add-times</u> (plus a little overhead) which is equivalent to <u>one-divide-time</u>.

Before describing the logic and the control of the hardware implementation of the SI, we make some remarks:

$$\text{Define } S\{x_1, y_1, x_2, y_2\} = \begin{cases} \text{if } x_1 x_2 > 0: & \text{undefined} \\ \text{if } x_1 = x_2 = 0: & \text{undefined} \\ \text{otherwise:} & \dfrac{x_2 y_1 - x_1 y_2}{x_2 - x_1} \end{cases}$$

From the definition it follows that the solution of

$$\frac{y_1 - y}{y_2 - y} = \frac{x_1}{x_2} \quad \text{and} \quad y \in (y_1, y_2)$$

is

$$y = S\{x_1, y_1, x_2, y_2\} \quad .$$

It is easy to verify that for any $\alpha \neq 0$:

$$S\{\alpha x_1, y_1, \alpha x_2, y_2\} = S\{x_1, y_1, x_2, y_2\} \quad ,$$

As illustrated in Figure VI. 2. 2.

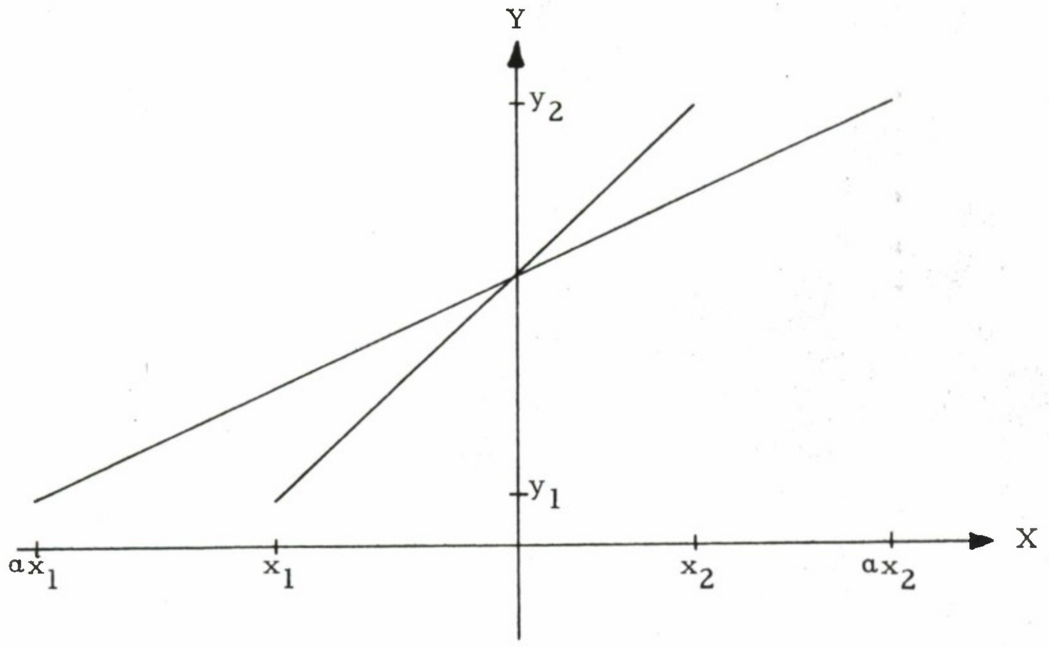


Figure VI. 2. 2: X scaling does not change the intersection

The accuracy of the solution depends on the slope of the segment. The flatter the segment is, the less sensitive the solution is. Therefore it is advised to normalize the x's, by scaling them up, before loading the X-registers.

It is also easy to verify that:

$$S\{-1, 0, b-1, a\} = \frac{a}{b} \quad (\text{for } b > 1)$$

which suggests a way of using the SI for division. This identity can be illustrated by the following figure:

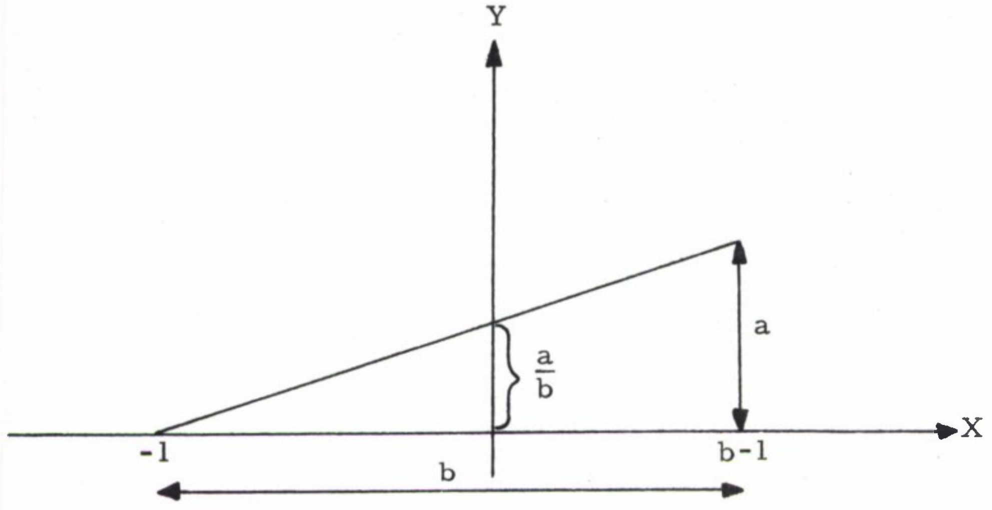


Figure VI. 2. 3: Using the Sutherland interpolator as a divider

If $b < 1$ but $2^{\ell}b > 1$, then:

$$S\{-1, 0, 2^{\ell}b-1, a\} = \frac{a}{2^{\ell}b} = 2^{-\ell}\frac{a}{b}$$

and

$$2^{\ell}S\{-1, 0, 2^{\ell}b-1, a\} = 2^{\ell}S\{-2^{-\ell}, 0, b-2^{-\ell}, a\} = \frac{a}{b} \quad .$$

The Operation of the Hardware Implementation

After loading the x_1, y_1, x_2, y_2 registers, and verifying that $x_1x_2 < 0$, the iterations can start.

Step (a): $x_1 + x_2 \longrightarrow \Sigma x$, and $y_1 + y_2 \longrightarrow \Sigma y$. Then continue with steps (b), (c), and (d) simultaneously.

Step (b): If $\Sigma x = 0$ then $y_0 = \frac{1}{2}\Sigma y$, stop the iterations.

Step (c): If sign (Σx) = sign(x_1) then: $\frac{1}{2}\Sigma x \longrightarrow x_1$; $\frac{1}{2}\Sigma y \longrightarrow y_1$. Then go to step (a).

Step (d): If sign(Σx) = sign(x_2) then: $\frac{1}{2}\Sigma x \longrightarrow x_2$; $\frac{1}{2}\Sigma y \longrightarrow y_2$.

The organization of this logic is illustrated in Figure VI. 2. 4.

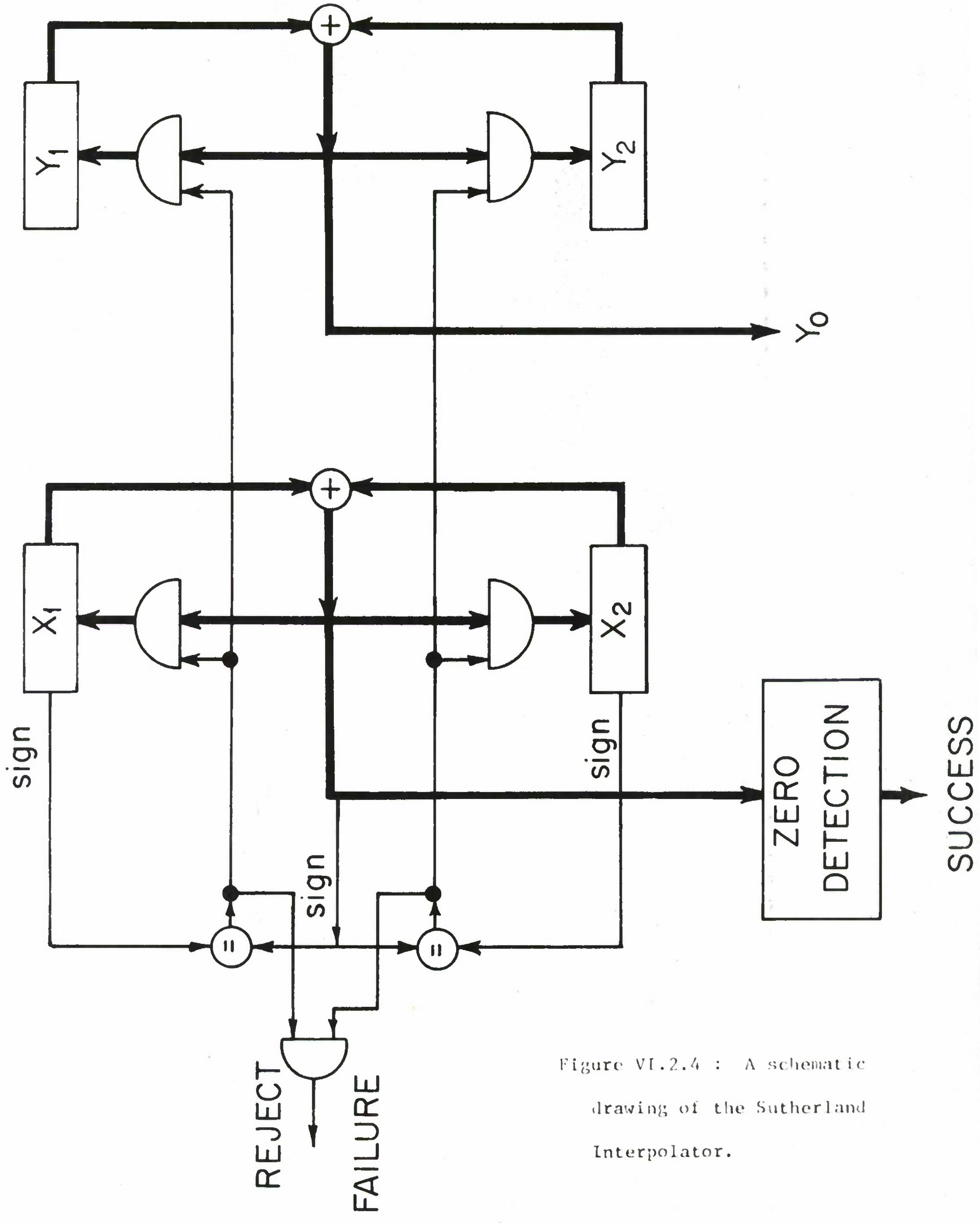


Figure VI.2.4 : A schematic drawing of the Sutherland Interpolator.

Remarks

(a) The division by 2 is achieved by ignoring the LSB (least significant bit) of the sums, and reproducing the sign bits. In 2's complement arithmetic negative numbers do not converge to zero, by repeated arithmetic right shift, but to -1. Hence in step (b) it is important to check also for $\Sigma x = -1$.

(b) If N-bits adders are used for N-bits numbers it is necessary to add the logic for detecting overflow. However, as the least significant sum bit is ignored, and only the least significant carry-bit is used, the last adder stage can be replaced by an "AND" gate, freeing one adder stage for taking care of the overflow which can be done by treating the numbers as (N+1)-bits numbers, with the sign bit reproduced, which do not overflow.

(c) Checking for signs equality between x_1 and x_2 can be done during the iterations, instead of before them. If $\text{sign}(x_1) = \text{sign}(x_2)$ then

$$[\text{sign}(\Sigma x) = \text{sign}(x_1)] \cdot \text{AND} \cdot [\text{sign}(\Sigma x) = \text{sign}(x_2)] \quad ,$$

which is an easily detected condition.

The Control of the SI

The SI can be in one of the following states:

(A) waiting for input in the registers;

(B) iterating;

(C) waiting for output from the y-adder, to be recorded.

State (A) is always followed by state (B).

State (B) is followed by state (C) only in case of success (intersection) or by state (A) in case of failure (no intersection).

State (C) is always followed by state (A).

This can be represented by the graph shown in Figure VI. 2. 5.

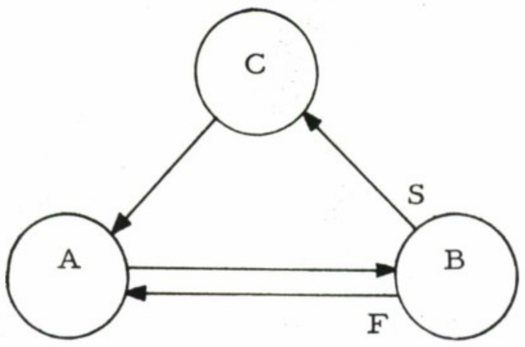


Figure VI. 2. 5: The states of the Sutherland interpolator

Let us assign 2-bits codes to the states:

(0, 0) - State (A)

(0, 1) - State (B)

(1, 1) - State (C) .

The remaining code (1, 0) will be used as a transit state between (C) and (A) to insure that (B) does not occur accidentally. The control logic can be implemented as shown in Figure VI. 2. 6.

(VI. 3) Lines Intersector

The SI as described in (VI. 2) is capable of intersecting segments with the Y-axis. In this section we will describe how to intersect any two segments, given by their end-points.

For intersecting $\overline{P_A P_B}$ with a line which is parallel to the Y-axis: $x = x_0$, a simple coordinates translation is required. It is easy to verify that if $P_0 = (x_0, y_0)$ is the intersection of $\overline{P_A P_B}$ and $x = x_0$ then:

$$x_0 = S\{x_A - x_0, y_A, x_B - x_0, y_B\} \quad .$$

Intersections of segments with the X-axis or with lines parallel to it can be found by changing the roles of x and y in the SI. Example: the intersection of $\overline{P_A P_B}$ with $y = y_0$ is $P_0 = (x_0, y_0)$ where:

$$y_0 = S\{y_A - y_0, x_A, y_B - y_0, y_B\} \quad .$$

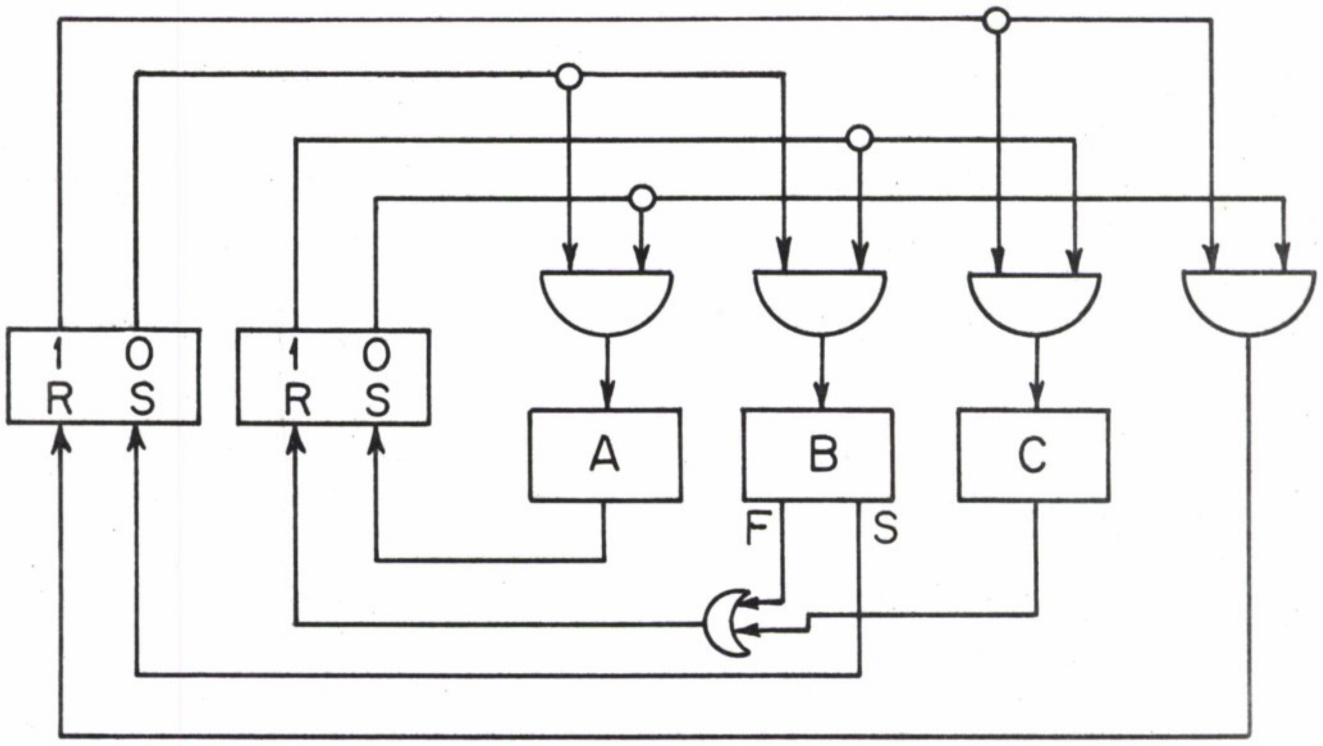


Figure VI.2.6 : The control logic for the Sutherland Interpolator.

Next we consider the problem of intersecting $\overline{P_AP_B}$ and $\overline{P_1P_2}$, where neither segment is parallel to the coordinate axes. The solution $P_0 = (x_0, y_0)$ satisfies:

$$\frac{x_A - x_0}{y_A - y_0} = \frac{x_B - x_0}{y_B - y_0} \quad \text{and} \quad \frac{x_1 - x_0}{y_1 - y_0} = \frac{x_2 - x_0}{y_2 - y_0} \quad .$$

Eliminating x_0 and y_0:

$$x_0 = \frac{(x_Ay_B - x_By_A)(x_2 - x_1) - (x_1y_2 - x_2y_1)(x_B - x_A)}{(y_B - y_A)(x_2 - x_1) - (y_2 - y_1)(x_B - x_A)}$$

$$y_0 = \frac{(x_Ay_B - x_By_A)(y_2 - y_1) - (x_1y_2 - x_2y_1)(y_B - y_A)}{(y_B - y_A)(x_2 - x_1) - (y_2 - y_1)(x_B - x_A)} \quad .$$

Needless to say, these equations are not the basis of our technique.

A possible solution, is reducing the problem into a solved one, by rotating the plane until the line $\overline{P_1P_2}$ is parallel to the Y-axis, then using the SI to find the intersection in the rotated plane, and rotating back to get the true intersection point.

The disadvantage of this technique is twofold. First it requires rotation of the 4 given points plus the solution point. Rotating each point requires 4 multiplications. Hence, all together 20 multiplications are needed. However, all the multiplication required for rotating the 4 given points can be done in parallel consuming only one multiply-time (but 16 different hardware multipliers). Second, for rotating points, we need the sine and the cosine of the angle between the line $\overline{P_1P_2}$ and the coordinate axes.

We shall show how to avoid these difficulties by using only 4 to 6 multiplications, avoiding the back-rotation and avoiding the need for sines and cosines. We will do it by the following scheme:

Consider the example in Figure VI. 3. 1:

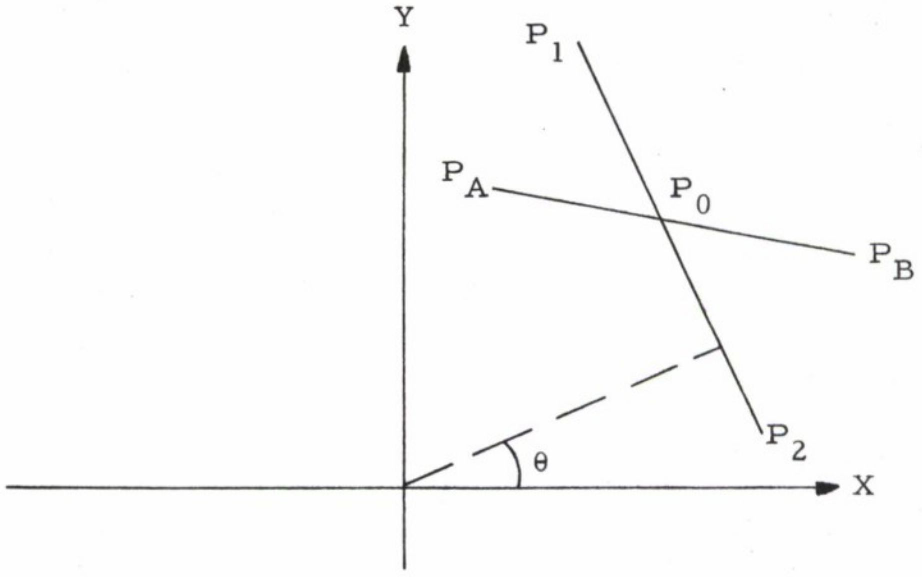


Figure VI. 3. 1: Intersection of segments, which are not parallel to the axes

where θ is the angle between the X-axis and the normal to $\overline{P_1P_2}$ from the origin. Rotate the plane by $-\theta$, and denote the new quantities by primes, as shown in Figure VI. 3. 2.

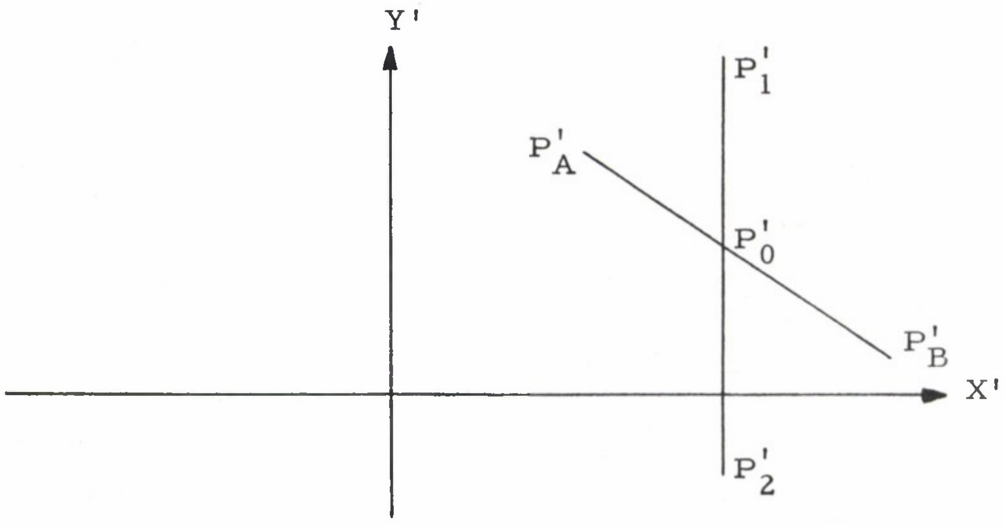


Figure VI. 3. 2: Figure VI. 3. 1 after the rotation

The relation between P' and P is:

$$(x', y') = (x, y) \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} .$$

P_0' is found easily as $\overline{P_1'P_2'}$ is parallel to the Y-axis:

$$P_0' = (x_0', y_0') \text{ where } x_0' = x_1' \text{ and } y_0' = S\{x_A' - x_0', y_A', x_B' - x_0', y_B'\}$$

Note that 6 multiplications are saved, (out of the 20) as y_1', x_2', y_2' are not needed.

From triangle similarity we get:

$$\frac{x_A' - x_0'}{x_B' - x_0'} = \frac{y_A' - y_0'}{y_B' - y_0'} = \frac{|\overline{P_0'P_A'}|}{|\overline{P_0'P_B'}|} ,$$

and

$$\frac{x_A - x_0}{x_B - x_0} = \frac{y_A - y_0}{y_B - y_0} = \frac{|\overline{P_0P_A}|}{|\overline{P_0P_B}|}$$

where $|\overline{P_iP_j}|$ denotes the length of the segment $\overline{P_iP_j}$. As

$$|\overline{P_0'P_A'}| = |\overline{P_0P_A}| \qquad \text{and} \qquad |\overline{P_0'P_B'}| = |\overline{P_0P_B}|$$

we get:

$$\frac{x_A - x_0}{x_B - x_0} = \frac{y_A - y_0}{y_B - y_0} = \frac{x_A' - x_0'}{x_B' - x_0'} .$$

By using only 6 multiplications we can get x_A', x_B', x_0' and then finding P_0 by:

$$x_0 = S\{x_A' - x_0', x_A, x_B' - x_0', x_B\}$$

and

$$y_0 = S\{x_A' - x_0', y_A, x_B' - x_0', y_B\} .$$

This eliminates the need for back-rotation. Note that both SI's above have the same values in their X-registers, and can be combined to one unit, having 6 registers, and 3 adders.

Next, we want to show how we can do without $\sin\theta$ and $\cos\theta$. Consider $\overline{P_1P_2}$:

$$\left.\begin{aligned} \Delta x &= x_2 - x_1 = -\lambda \sin\theta \\ \Delta y &= y_2 - y_1 = \lambda \cos\theta \end{aligned}\right\} \quad \text{where } \lambda = \pm|\overline{P_1P_2}| \ .$$

From $x' = x\cos\theta + y\sin\theta$, one gets:

$$\lambda x' = x\cdot\Delta y - y\cdot\Delta x \ .$$

The SI is homogeneous in the x coordinates, so we have:

$$x_0 = S\{\lambda x'_A - \lambda x'_0, x_A, \lambda x'_B - \lambda x'_0, x_B\}$$

$$y_0 = S\{\lambda x'_A - \lambda x'_0, y_A, \lambda x'_B - \lambda x'_0, y_B\}$$

where:

$$\lambda x'_A = x_A\cdot\Delta y - y_A\cdot\Delta x$$
$$\lambda x'_B = x_B\cdot\Delta y - y_B\cdot\Delta x$$
$$\lambda x'_0 = x_1\cdot\Delta y - y_1\cdot\Delta x \ .$$

The SI rejects the case in which the intersection is outside $\overline{P_AP_B}$, but does not reject if the intersection is outside $\overline{P_1P_2}$. If one cares for this condition as well, he should check for it when the result is announced.

If many segments have to be intersected with $\overline{P_1P_2}$, the problem is solved using only 4 parallel multiplications per segment, as

$$\lambda x_0 = x_1\Delta y - y_1\Delta x$$

depends only on P_1 and P_2.

Note that the above technique does not treat $\overline{P_AP_B}$ and $\overline{P_1P_2}$ symmetrically, and is oriented to consider $\overline{P_AP_B}$ as a segment but $\overline{P_1P_2}$ as a line which causes the need for the extra checking for $P_0 \in (P_1, P_2)$. If time is critical, and if we wish to consider both $\overline{P_AP_B}$ and $\overline{P_1P_2}$ as segments, then we use 2 similar units which compute in parallel both $(x''_A - x''_0, x''_B - x''_0)$ and $(x'_1 - x'_0, x'_2 - x'_0)$[1]. The trivial rejection can be done simultaneously for both cases.

Another trivial rejection can be used by checking, prior (or in parallel) to any other computation the condition:

$$
\begin{aligned}
& (x_A > x_1).\text{AND}.(x_B > x_1).\text{AND}.(x_A > x_2).\text{AND}.(x_B > x_2) \\
.\text{OR}.\quad & (x_A < x_1).\text{AND}.(x_B < x_1).\text{AND}.(x_A < x_2).\text{AND}.(x_B < x_2) \\
.\text{OR}.\quad & (y_A > y_1).\text{AND}.(y_B > y_1).\text{AND}.(y_A > y_2).\text{AND}.(y_B > y_2) \\
.\text{OR}.\quad & (y_A < y_1).\text{AND}.(y_B < y_1).\text{AND}.(y_A < y_2).\text{AND}.(y_B < y_2) \quad .
\end{aligned}
$$

If this condition holds, then obviously there is no intersection. In the example of Figure VI. 3. 3 we illustrate all the trivial rejections.

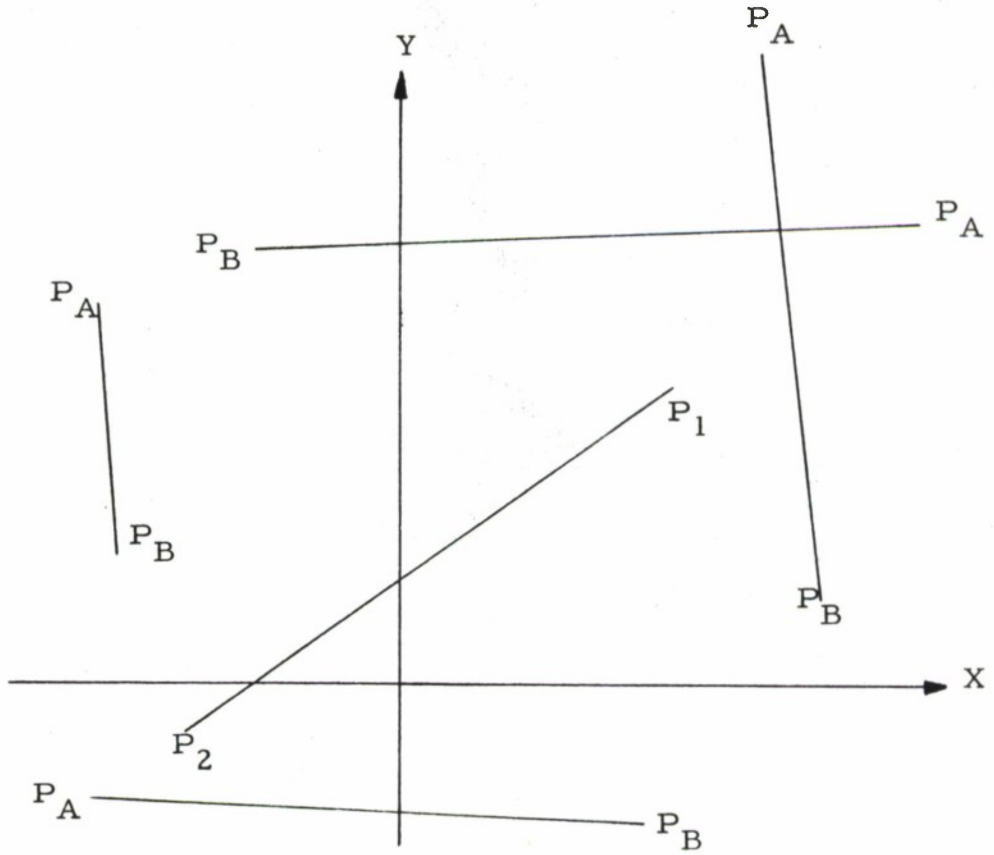


Figure VI. 3. 3: The segments $\{P_A, P_B\}$ are trivially rejected

[1]The double prime quantities are in a system which was rotated such that $\overline{P''_1 P''_2}$ is parallel to the Y-axis.

(VI. 4) On String-Line Intersecting

In many real time applications it is essential to find in a very short time, all of the intersections of a given line with some sets of contiguous segments. We call such contiguous segments set "strings".

We describe below a device which can perform this task. This device is initialized by loading information about the line, then it is given the data about the end-points of the segments, and it finds all the intersections of these segments with the line specified earlier. The time consumed by each point is between one multiplication-time plus two addition times to one divide-time. With today's technology one divide time is about the time required for fetching information from memory.

The operation of this device is based on the method described in (VI. 3). The device described here is able to handle some operations in parallel. All the time-consuming operations, (namely fetching the data, multiplication and interpolation) are organized such that each of them starts as soon as possible, taking advantage of cases in which the other operations are fast. Because these operations can work completely in parallel the time required per point is the duration of the longest operations.

Let the strings be represented in memory by a sequence of triples $\{b_i, x_i, y_i\}$ where (x_i, y_i) are the coordinates of P_i, the i-th point, and b_i is a bit which is set to 'one' if there is a segment between P_{i-1} and P_i, and it is 'zero' otherwise. Consider the example in Figure VI. 4. 1:

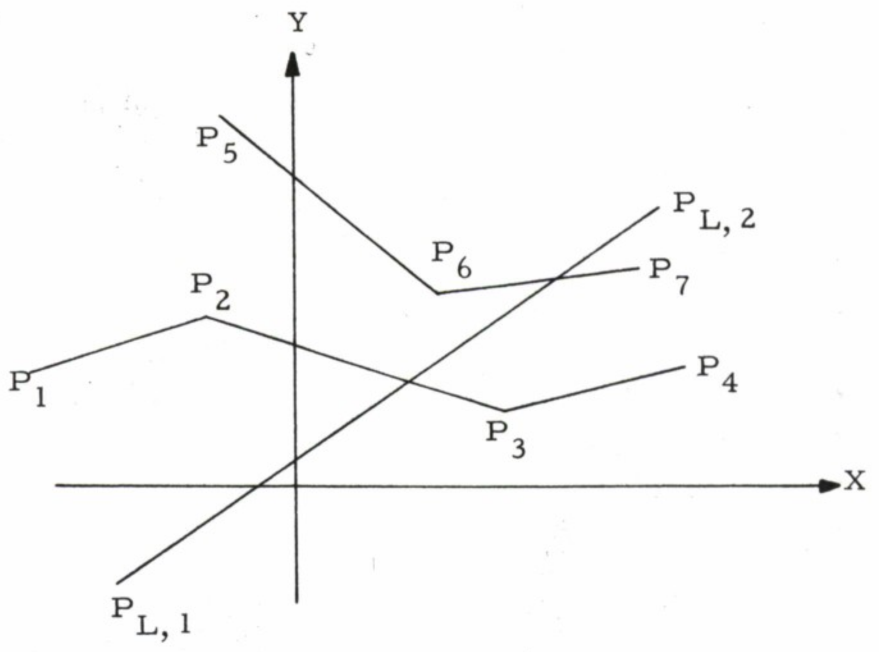


Figure VI. 4. 1: A line and a string

These segments are stored as:

$$(0, x_1, y_1)(1, x_2, y_2)(1, x_3, y_3)(1, x_4, y_4)(0, x_5, y_5)(1, x_6, y_6)(1, x_7, y_7) .$$

The line $\overline{P_1P_2}$ to be intersected is defined by

$$C = \lambda(y_2 - y_1)$$

$$S = \lambda(x_1 - x_2)$$

$$\delta = -(Cx_1 + Sy_1)$$

where λ is a normalization factor. These quantities are calculated, once per line, and loaded to the C, S and δ registers.

After the initialization of the device the first triple is given to it, which starts to process the data. When it is ready to accept the next triple it calls its input channel. When an intersection is found it calls the output channel to record the intersection point. The organization of the hardware is oriented toward a high degree of parallel processing. The data about each point has to pass 2 multiplications in parallel and 2 additions in sequence, before it is given to the SI, together with the information about the previous point. If the b-bit is found to be 'zero', the SI rejects the segment, otherwise it starts its operation which might reject the segment if there is no intersection, or find the intersection point. In this case, the SI announces success and calls the output channel to record the intersection.

The hardware organization is shown in Figure VI. 4. 2 and its operation is described by the graph in Figure VI. 4. 3.

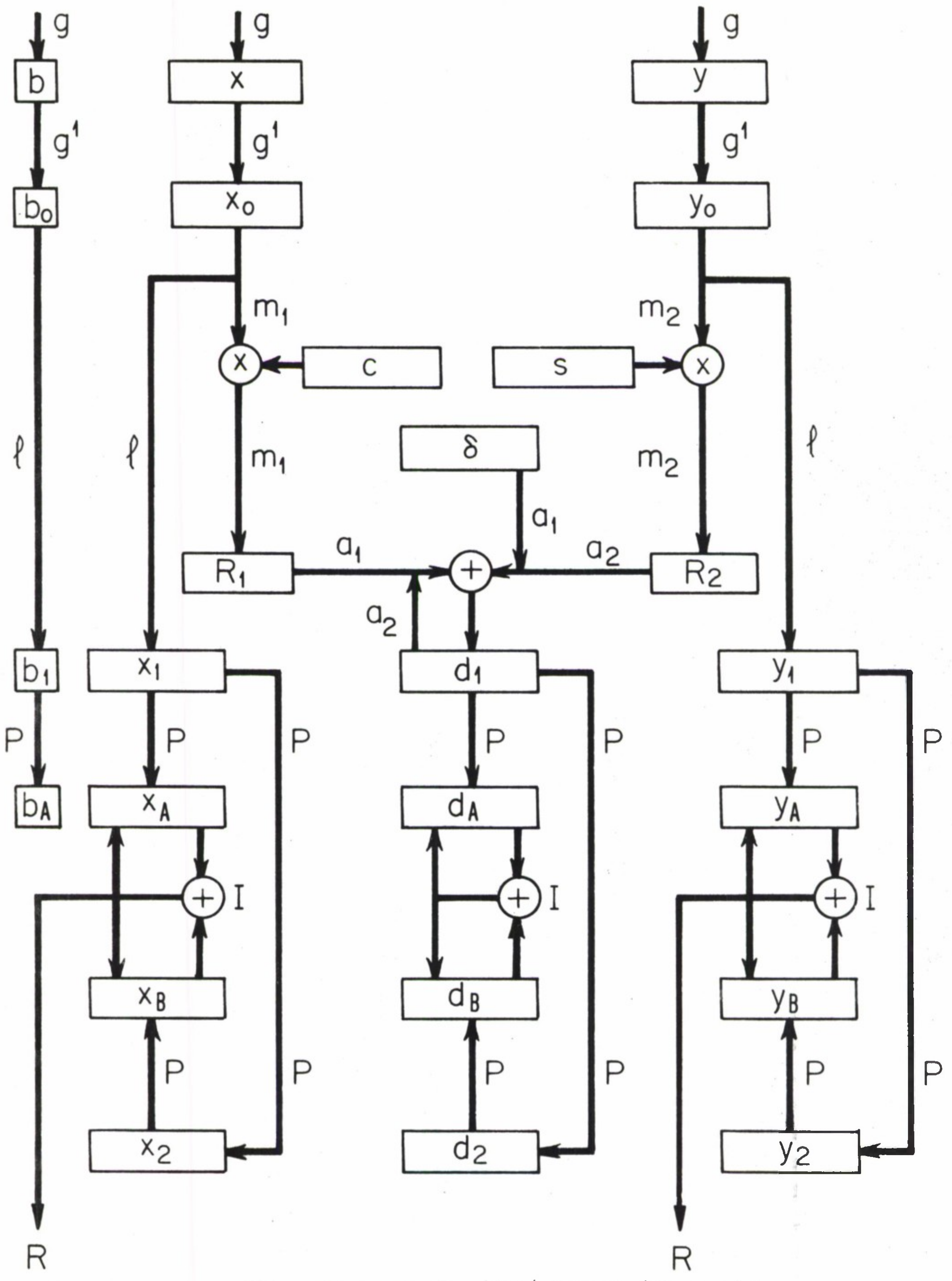


Figure VI.4.2 : The line/segments intersector.

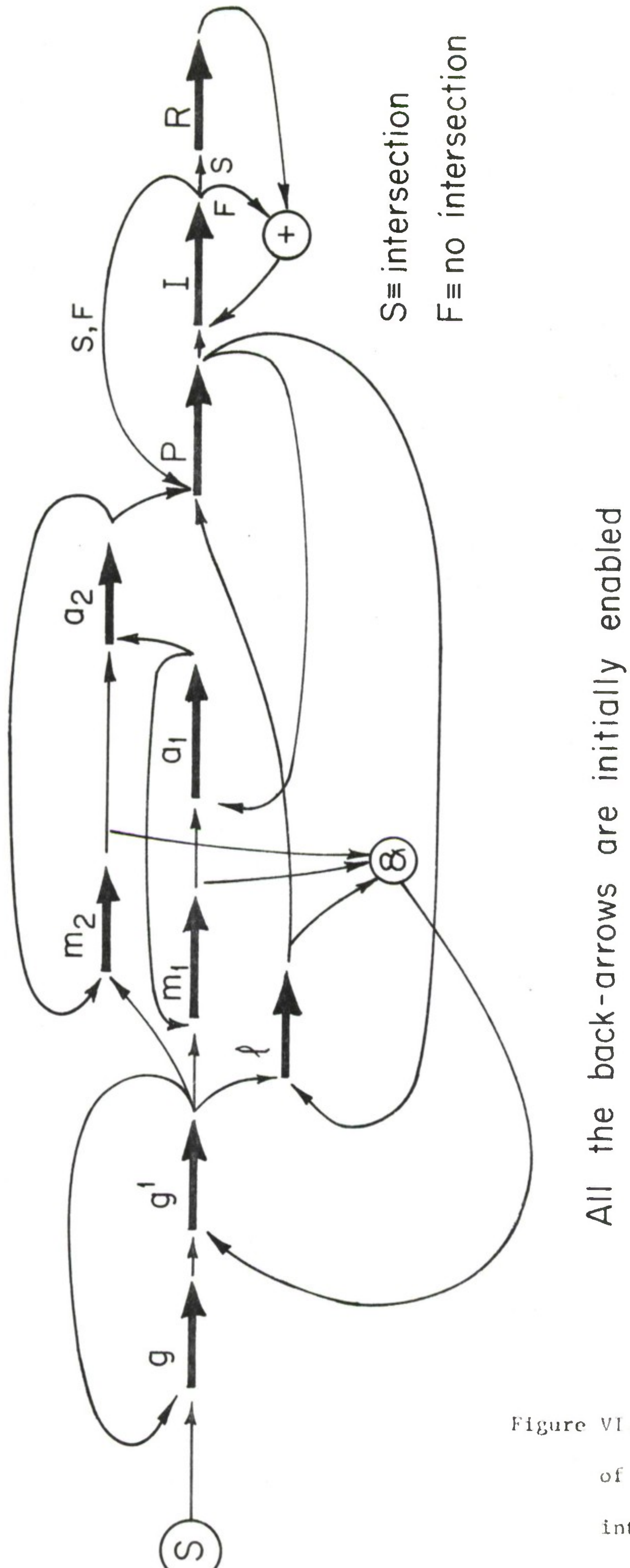


Figure VI.4.3 : The operatio

of the line/segments

intersector.

REFERENCES

[1] Sutherland, Ivan E., "A Head-Mounted Three-Dimensional Display," Proceedings of the Fall Joint Computer Conference, 1968, p. 764.

[2] Baker, H. F., Principles of Geometry, 6 vol., Cambridge University Press, Cambridge, 1922.

[3] Winger, R. M., An Introduction to Projective Geometry, Heath and Co., Boston, 1923.

[4] Cohen, D. and Lee, T. M. P., "Fast Drawing of Curves for Computer Display," Proceedings of the SJCC, 1969.

[5] Hildebrand, F. B., "Introduction to Numerical Analysis," McGraw-Hill, 1956, chapter 4.

[6] Roberts, L. G., "Homogeneous Matrix Representation and Manipulation of n-Dimensional Constructs," The Computer Display Review, Adams Associates, 1965.

[7] Roberts, L. G., "Machine Perception of Three-Dimensional Bodies," MIT Lincoln Laboratory TR-315, May 1963.

[8] Cohen, D., "A Program for Drawing Bodies with the Hidden Lines Removed," a term-project for course 6.539 MIT, Fall 1965.

[9] Haynes, H. T., "A Computer Method for Perspective Drawing," Master's Thesis, Texas A and M University, August 1965.

[10] Loutrel, P., "A Solution to the Hidden-Line Problem for Computer-Drawn Polyhedra," Ph.D. Thesis, NYU, September 1967.

[11] Appel, A., "The Notion of Quantitative Invisibility and the Machine Rendering of Solids," Proceedings of the 22nd National Conference ACM.

[12] Wylie, Romney, Evans and Erdahl, "Half-Tone Perspective Drawing by Computer," Technical Report 4-2, Computer Science, University of Utah, February 1968.

[13] Mathematical Application Group, Inc. (MAGI), "3-D Simulated Graphics," Datamation, February 1968.

[14] Galimberti, R. and Montanari, U., "An Algorithm for Hidden-Line Elimination," Communication of the ACM, April 1969.

[15] NASA Contract NAS9-3916 with the General Electric Company Electronic Laboratory.

[16] Warnock, J., "A Hidden-Line Algorithm for Halftone Picture Representation," University of Utah Technical Report 4-5, May 1968.

[17] Cheatham, T. C., "The Theory and Construction of Compilers," Computer Associates, 1967.

[18] Holt, A. and Shapiro, R., "Final Report for the Information System Theory Project," Contract AF30(602)-4211, Applied Data Research, February 1968.

[19] Sproull, R. and Sutherland, I., "A Clipping Divider," Proceedings of the Fall Joint Computer Conference, 1968.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Harvard University Cambridge, Massachusetts	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
	2b. GROUP N/A

3. REPORT TITLE

INCREMENTAL METHODS FOR COMPUTER GRAPHICS

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
None

5. AUTHOR(S) (First name, middle initial, last name)

D. Cohen

6. REPORT DATE April 1969	7a. TOTAL NO. OF PAGES 115	7b. NO. OF REFS 19

8a. CONTRACT OR GRANT NO. F19628-68-C-0379	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-69-193
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Directorate of Planning and Technology, Electronic Systems Division, AFSC, USAF, L G Hanscom Field, Bedford, Mass. 01730

13. ABSTRACT

This report is concerned with incremental methods for computer graphics. The application of the incremental approach to some advanced problems in computer graphics is discussed and demonstrated. In Section II, the problem of fast curve generation and display is discussed. This section uses the mathematical approach to curves as a perspective projection of polynomial-curves in higher-dimensional spaces. Section III, also discusses a fast generation of curves, but using another mathematical approach, the linear-differences method, only two-dimensional curves are discussed. Section IV, shows how to make the Warnock algorithm, for hidden lines elimination, incremental. Section V, discusses the generation of half-tone images, in real-time. The technique suggested there requires a special-purpose hardware to be built. Section VI, discusses an incremental method for finding the intersection of given line-segments. The technique suggested there also requires a special hardware for implementation.

DD FORM 1 NOV 65 1473

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer Graphics Curves Linear-Differences-Curves Half-Tone Images Hidden Lines Elimination Lines Intersecting Conics Shading						