1990

Using conformal mapping to aid in computer animation

Highland Mary Agnello

Rochester Institute of Technology
Computer Science Department

Using Conformal Mapping to Aid in Computer Animation

by
Highland Mary Agnello

October 2, 1990

Approved by:

Prof. Peter Anderson

Prof. Chi Ming Tang

Prof. Guy Johnson

Table of Contents

Problem Statement

The purpose of this thesis is to explore applications of the mathematics of conformal mapping to interactive computer graphics design and animation. We have constructed a prototype graphics design tool to provide the user with a simple and expedient way of manipulating an object using conformal mapping, specifically polynomial interpolation. Existing computer animation tools give the artist a choice between the slower drawing board, the computer, or a combination of both. Scientists, engineers, and artists can benefit from computer aided animation when trying to simulate their ideas.

Using complex analytic functions, control points, and the complex plane, the user is able to manipulate a two dimensional object considered as a subset of the complex plane. Given the artistic perspective, the user may choose to produce an animated sequence or one final desired image. Engineers and scientists may also be able to employ this system in areas of stress analysis, fluid flow, and so on.

History of Animation/Computer Animation

Animation is the process of creating images that appear to move [FOX 87]. Persistence of vision plays a major role in this process. The brain retains an image longer than the retina. This persistence causes consecutive images to be blurred together if they are presented at a rate faster than 18-24 times a second. Thus, one needs 1,440 frames for a minute of animation, and 172,8000 frames for a two hour movie. From this perspective, it is clearly evident that the task of developing an animated film is labor intensive. Hence, development of animated films would profit from computer support.

Animation, from the beginning, has been aided by one type of machine or another [FOX 87]. In the mid 1820's, Dr. John Paris of England developed the Thaumatrope which consisted of a piece of paper with two pieces of string attached to opposite corners. When these strings were pulled, the paper would flip, revealing the two pictures on either side. Because of persistence of vision, the two pictures would become blurred, thus creating an animated effect. After this, many other contraptions set out to produce the same effect. The Zoetrope and the Zoopraxiscope are examples of early animation devices. (see figure 5a). The Zoopraxiscope transferred images to a

screen, but it was the Praxinoscope which opened the first movie theater in Paris in 1892. This device took images on strips of translucent paper and projected them on screen. In 1868, what is commonly known as the "flip book" was patented as the Kineograph. This device consisted of a stack of cards with drawings on them. These cards were held together at one end and flipped through to create animation through persistence of vision.

After the turn of the century, developments occurred to bring animation closer to today's technology. Phase animation, where drawings are superimposed on one another, saved repeated drawings of background scenes. Soon cel animation replaced phase animation by using transparent celluloid to draw foreground figures. This is still in common practice today. All these developments lead us to the Disney Classics as well as Saturday morning cartoons.

It was not until the 1960's that animation became an application for the computer. The first applications were far from the entertainment industry, but dealt with texture patterns modeled on screen. Zajac amd Knowlton pioneered these applications at Bell Labs [FOX 84]. These techniques and advancements paved the way for further technical uses for computer animation such as simulations of viscous fluids, vibrations through solids, and simulated landing of an aircraft. Since computer graphics are much more advanced today than during the 1960's, we see computer animation in the entertainment and

advertising industries. Since computer image resolution is comparable to film, it can be very effective to use the computer in these industries. Computers are also more time and cost efficient in restructuring parts of a film. In many movies with special effects, models are used (plastic, clay, or cardboard) to simulate an aircraft or spaceship, etc. If scenes must be shot several times it can become very expensive. Using a computer would initially be time consuming because the database for all the objects being shot would have to be constructed. However, after this initial investment, any scene involving the objects in the database may be repeatedly shot with minor changes to lighting, movement, etc. Star Wars, TRON, and Star Trek: The Wrath of Khan are prime examples of movies that have used computer animation. Lucusfilm Ltd. is one of the pioneers in the computerized special effects.

Just recently, we have seen computer animation in media communication with the space travels of Explorer. The views we saw of Neptune were largely due to the developments of James Blinn with Charles Kohlase. These views of the outer planets that we visualize are aided through the laws of physics. Some simulated views are from the perspective of riding on the spacecraft or riding behind it. Some other simulations were used for future exploration techniques. One computer simulation was the reentry of a space shuttle. In a situation like this, time and cost efficiency are not as important as reducing accidents in the future. With the computer, tests can be

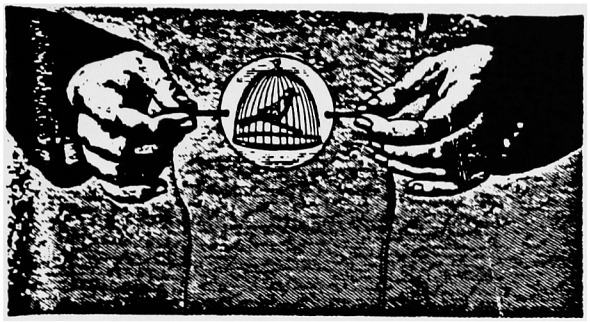


Figure 1.1: The first animation device — the Thaumatrope (circa 1826). *(Courtesy of Stanford University Museum of Art.)*

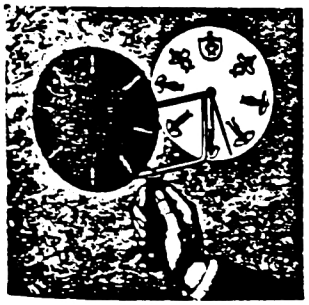


Figure 1.2: The first animated picture — the Phenakistoscope (circa 1832). *(Courtesy of Stanford University Museum of Art.)*

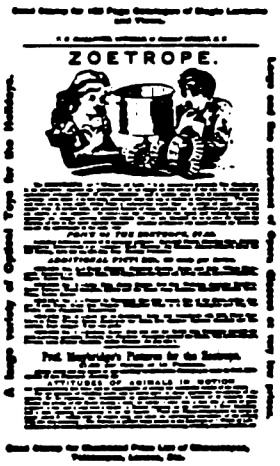


Figure 1.3: Zoetrope — the wheel of the devil. *(Courtesy of Stanford University Museum of Art.)*

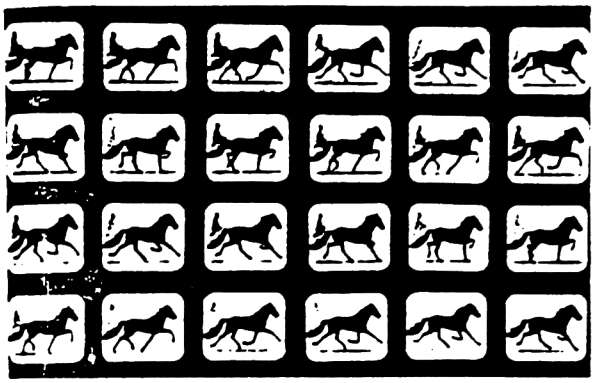


Figure 1.4: The horses of Eadweard Muybridge.

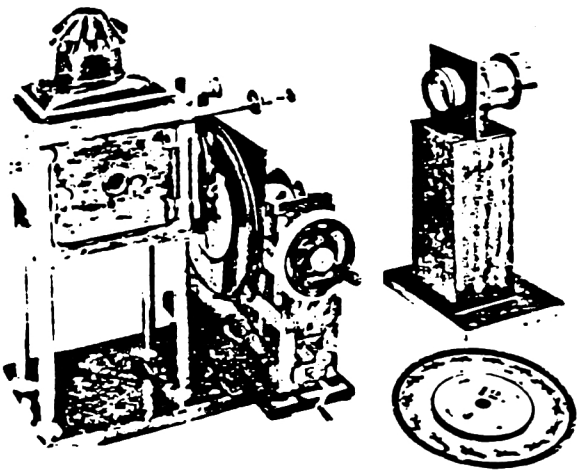


Figure 1.5: Muybridge's Zoopraxiscope. *(Courtesy of Kingston-upon-Thames [illegible] and Art Gallery, Stanford University of Art.)*

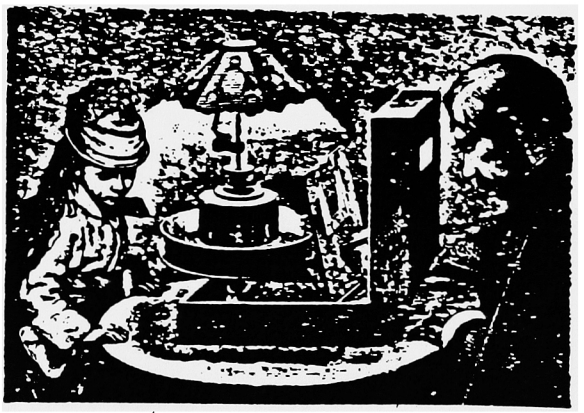


Figure 1.6: The Praxinoscope. *(Courtesy of Stanford University of Art. [Reproduced from Gaston Tissandier, Popular Scientific Recreations, N.Y., c 1880, n.d.].)*

simulated without risking life. There are many other applications in the engineering world besides space travel. These all add up to better decision making at a lower cost.

Of course, we have all seen arcade games and know how they attract kids to spend their parents' money. Now educators are trying to capture children's curiosity in computers to bring them to enjoy serious lessons. The "attract mode" could be put on when the computer was not being used. This mode is what we see today when a video game is not being used. Through computer animation, people are attracted to the computer screen to learn what the program does and hopefully to use the application. In education, it would be important to use the effects of computer animation to keep the student's interest throughout the lesson.

Computer Animation Today

Computer animation today can take many forms because the computer as a tool can accomplish many things for the animator [MAGN 85]. To finish the job of making an animated work, the computer takes on many different functions. Primary functions are to capture, create, and store drawings. The most important drawings the designer works with are called key frames, which represent the two extreme positions in an animated movement. They can be either digitized or produced through computer programs. Color schemes can be laid out interactively or images can be created through software. "Offline", computers can be used to control camera movement and aid in post production work. When all the work is complete, computers can aid in editing and synchronization of the finished product.

Computer applications in the animation environment are classified according to levels of computer systems [MAGN 85]. Level one consists of a basic graphics editor to help the artist put up a picture and store it for future use. At this low level, time is not the primary factor. At level two, systems take time into account, because movement of the image is aided. Here, systems can compute in-betweens and move an object. This could be used to aid or replace the in-betweener. Level three gives the user the ability to apply different transformations to the image, such as rotations, translations, pan, and zoom. Level four allows for the definition of actors.

An actor is an object in a picture which has its own animation, such as the spinning wheels of a car. Level five does not exist yet, but it will be a very powerful tool eventually. With the advent of artificial intelligence, this level will learn in its operating mode, thereby becoming more and more powerful. The work in the present thesis is a contribution to levels one through four.

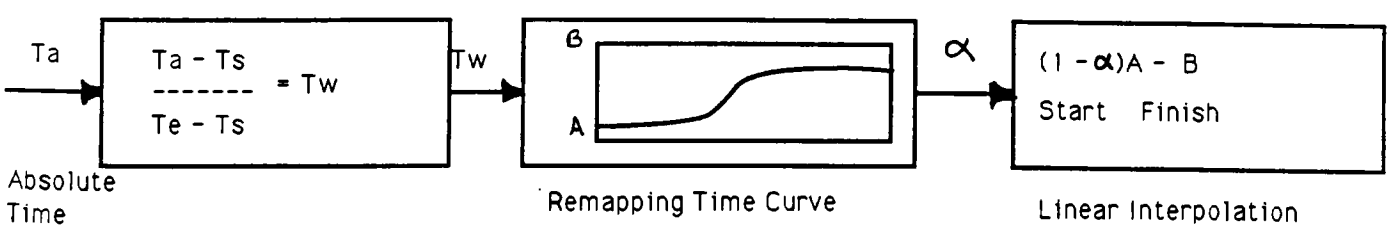
A simpler way of classifying computer animation systems is to divide them up into computer-assisted and modeled animations. Computer-assisted animation, namely the level two system, generally aides traditional key frame techniques. Modeled animation deals with the complex world of three-dimensional space. Calculations dealing with three dimensional space are very complex without the aid of a computer. To summarize, the difference between computer-assisted and modeled animation is that modeled animation uses programmed software to do the rendering, moving, and image creation while computer-assisted animation is just an extension of older techniques prior to computer intervention.

In our present application we are helping the animator to create a movement from one key frame to another. The computer is calculating the inbetween frames through software. With the level definitions above one can conclude that this is a Level 2 application. Although we are using the same software utilized in modeling systems, we are doing this to develop in-between frames between two key frames. Since the computer

is aiding the old techniques of key frame animation, this approach is classified under computer-assisted animation.

In-betweening (i.e., the interpolation or sequence of images between two still frames or key frames) has been around since animation began [MAGN 85] [ROSE 87]. In-betweening requires certain procedures be performed in order to achieve a realistic effect. Many times the first key frame will have a different number of key strokes than the second key frame. Strokes on a frame are defined as sequences of visible line segments. If the number of strokes on each key frame is the same, then only simple calculations are needed before the in-between frames to be drawn; otherwise additional calculations must be done. Suppose key frame 1 (KF1) has four strokes and KF2 has two. To balance the two frames, the lines of KF2 must be divided so that it has the same number of strokes as KF1.

In our application, the main driving force controlling the in-between generation is an interpolation algorithm and a history of control point motion. The advantage of this method is the number of frames between the key frames may be altered without affecting the algorithm. Through normalization of time and distance on the path of movement, such changes are transparent to the user. The General Parameter Interpolation is shown in figure below [ROSE 87]:



This parameter interpolation involves renormalizing time, curve shape, and interpolation of actual parameters. The curve is the rate at which the object accelerates and decelerates. The graph shows the relationship between time and distance. As the object moves, acceleration increases as the curve shows with the increased distance traveled over time. In most cases, movements begin slow, increase their speed in the middle of the movement, and finally slow to the final stop. This is known in animation as an "ease in ease out" procedure. If acceleration is an issue, one can use the Newtonian equation for acceleration [ROSE 87]; the position p, is described as a function of time:

$$p = p_0 + vt + .5at^2$$

Here t is the frame number, v is the velocity, a is the acceleration, and p_0 is the initial position. If we want the movement to use ease-in-ease-out we deviate from this formula as follows:

first half

$$p = A + (2(B - A)/F)*t^2, \quad 0 =< t =< t_{Mid}$$

second half

$$p = A + (2(B - A)/F)*t + (-2(B - A)/F)*t^2, \quad t_{Mid} <= t <= t_{End}$$

Here F is the frame number. This will give the highest acceleration at the midpoint of the movement. If one wants the height of acceleration to be somewhere other than at the midpoint one can use the sinusoidal formula:

$$p = p_0 + (B - A)\sin((t/F)\ /2),\ 0 =< t =< t_{End}$$

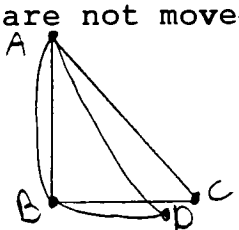
The present program does not deal with the issue of acceleration, but it could be an additional topic worthy of thesis evaluation.

Theoretical and Conceptual Development

This section describes the mathematical background for the generation and animation of an image. Here we will discuss the original polynomial interpolation approach and the complex plane control point manipulation approach.

Development

We interpret the use of complex analytic functions for animation as a polynomial interpolation problem. Our problem involves a given set of points, the "control points" in an image. When these points are moved, they define some function which will connect the set of new points. We begin with a table of x values, denoting complex numbers, i.e., points in a plane, defining our original object. One of these points, C, is moved to a new location D, defining a new object (the other two points, A and B, are not moved).



x	f(x)
x_0=A	A
x_1=B	B
x_2=C	D

We will find an interpolating polynomial, f, that fits the points in this table. Given n+1 points, the function f will be of degree n. In this example, we will have to determine a

quadratic function, since we have 3 given points. The function will be of the form:

$$f(x) = a_0 + (x - x_0)a_1 + (x - x_0)(x - x_1)a_2$$

And, in general, for any group of n+1 points, f(x) will be of the form:

$$(*) \quad f_n(x) = a_0 + (x - x_0)a_1 + (x - x_0)(x - x_1)a_2 \cdots + (x - x_0)(x - x_1)\cdots(x - x_{n-1})a_n$$

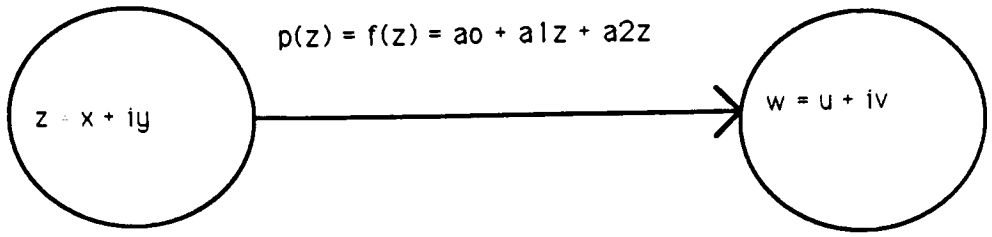
The interpolation method utilized here is Newton's Divided Difference Interpolation formula. Suppose that $f(x_k) = y_k$ for $k=0,\ldots,n$. Then we can compute the coefficients $a_0,\ldots,a_n$ for formula (*) using the following recursively defined values.

$f[x_s] = y_s$

if $k >= 1$,

$$f[x_s,x_{s+1},\ldots,x_{s+k}] = (f[x_{s+1},\ldots,x_{s+k}]-f[x_s,\ldots,x_{s+k-1}])/(x_{s+k}-x_s)$$

Then, the coefficient $a_k=f[x_0,x_1,\ldots,x_k]$, for $k=0,\ldots,n$ [BURD 81].



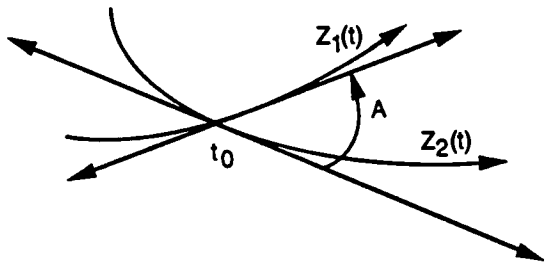
In the plane, we are accustomed to the real coordinate system dealing with x and y as separate real numbers [KREY 72]. In the complex plane, x and y are represented by the real and imaginary parts of a complex number. The complex number z = a + bi, corresponds to the point (a,b), where a and b are real numbers. Thus, the coordinates of a point in the plane come from a single complex number z.

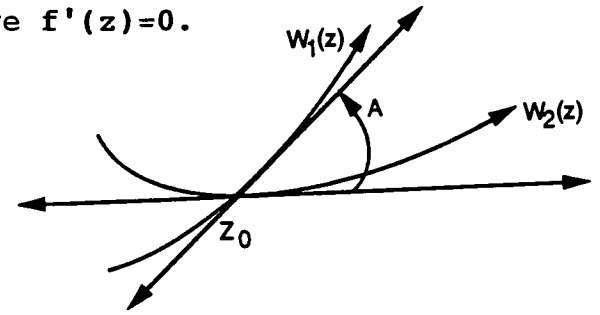
In our case, we have one set of complex numbers numbers in the z-plane (form z = a + bi) and another set of complex numbers in the w-plane (form w = u + vi). The mapping is accomplished through p(z) = f(z) = w.

The coordinates in the W plane may be thought of as two real functions of x and y, w = u(x,y) + iv(x,y). The complex polynomial f(z) is an analytic function defined on the entire complex plane. A function is said to be analytic in the domain D if it is defined and has a (complex) derivitive at every point in D. Analyticity is key to proving this mapping conformal. A conformal mapping is any transformation which preserves angles between the tangent lines of any two intersecting curves in both magnitude and direction. Since the function formed to represent our object manipulation is a polynomial and hence analytic, it will be conformal almost everywhere.

Theorem The mapping defined by an analytic function f(z) is conformal, except where the derivative f'(z)=0.

Proof:





Let there be two curves $z_1(t)$ and $z_2(t)$ which intersect at t_o. Both $z_1(t)$ and $z_2(t)$ have continuous, non-vanishing tangents, so we say they cross at angle A:

$$A = \arg z_1'(t_0) - \arg z_2'(t_0)$$

Let $w_i(z) = f(z_i(t))$, where:

$$w_1(z) = f(z_1(t))$$

$$w_2(z) = f(z_2(t))$$

and $w_1(z)$ and $w_2(z)$ intersect at z_0.

$f(z_0) = f(z_1(t_0)) = f(z_2(t_0))$ where $z_0 = z_1(t_0) = z_2(t_0)$

and $\arg f'(z_0) = \arg f'(z_1(t_0)) = \arg f'(z_2(t_0))$.

Assume $f'(z_1(t_0)) = 0$ and $f'(z_2(t_0)) = 0$.

By the chain rule,

$$w_1'(z_0) = f'(z_1(t_0))z_1'(t_0)$$

$$w_2'(z_0) = f'(z_2(t_0))z_2'(t_0)$$

Since arg(rs) = arg(r) + arg(s) when r and s are two non-zero complex numbers,

$$\arg w_1'(z_0) = \arg f'(z_1(t_0)) + \arg z_1'(t_0)$$

$$\arg w_2'(z_0) = \arg f'(z_2(t_0)) + \arg z_2'(t_0)$$

To get this new angle of intersection we subtract arg $w_2'(z_0)$ from arg $w_1'(z_0)$:

$$\arg w_1'(z_0) - \arg w_2'(z_0) = \arg f'(z_1(t_0)) - \arg f'(z_2(t_0)) + \arg z_1'(t_0) - \arg z_2'(t_0)$$

$$\arg w_1'(z_0) - \arg w_2'(z_0) = \arg z_1'(t_0) - \arg z_2'(t_0)$$

Consequently the angle of intersection of $w_1(z)$ and $w_2(z)$ equals angle A. This shows that angles are preserved when a mapping is defined by an analytic function.

To follow is a discussion on the present systems mathematical theory. Since the new approach also uses analytic functions, the previous proof shows that the new mapping is also conformal.

Further Development and Strategies

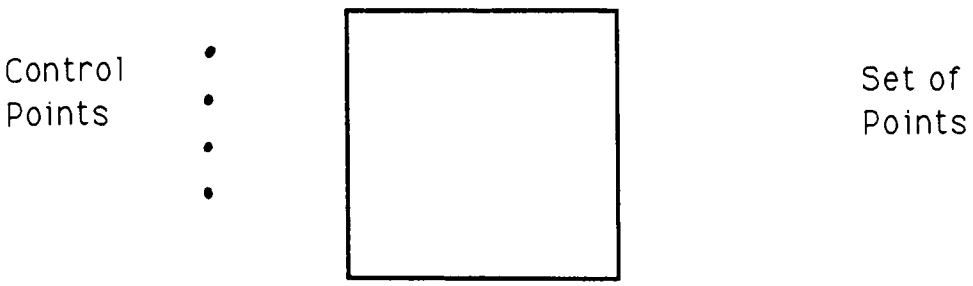
The present thesis project has evolved through several different interpretations of what the system should do. The most limiting parts of the first proposed system was that the outer object boundary must be a triangle. Furthermore, only one point of the outer object could be moved. The major changes made to this original system are as follows:

1) Allow any shape to be displayed and manipulated.

2) Allow the user to decide where to place the control points to manipulate the object (maximum of 4). The control points do not need to be on the object itself.

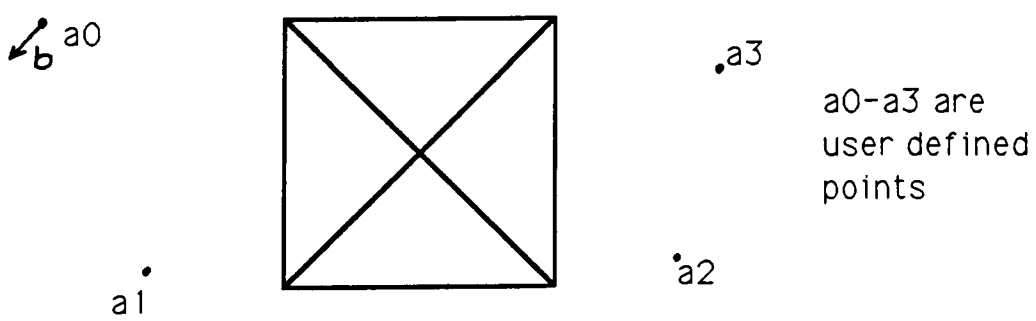
3) Define the object as a finite set of points rather than a set of line segments. It will then be this set of points that is mapped by the interpolated function. Now, all objects which can be represented by a set of points can be manipulated by the system. This will greatly expand the scope of the finished product. Since the interpolation of the coefficients and the control points no longer depend on the object in question, we can use one common method for determining

the mapping function.



4) The set, S, which represents the object may be as large as 2000 points. When the user manipulates one of the control points by dragging the mouse, a subset of S is correspondingly dragged by using the newly interpolated function. Having only a subset of the image mapped as the cursor is dragged will speed up processing and give the user a good idea of what the object will look like as it moves. When the mouse button is released the entire image (up to 2000 pixels) will be mapped according to the last placement of the control points. Then the internal representation of the object will be updated with the newly mapped image. Now, when another control point is moved this new image, not the original will be manipulated.

5) Since the control points have been changed to be user defined and limited to a small number, we can more easily generalize the mapping to the whole complex plane.



Mathematical Theory

We introduce an auxiliary polynomial, g(z)=f(z)-z, in order to simplify some calculations. In the following table, $a_0,\ldots,a_n$, are the control points; a_0 is the chosen point moved to b.

z	$f(z)$	$g(z)=f(z)-z$
a_0	b	b
a_1	a_1	0
a_2	a_2	0
a_3	a_3	0
.	.	.
.	.	.
.	.	.
a_n	a_n	0

The preponderance of zeros in this table makes the interpolation algebra for g(z) simpler.

$$g(z)=\frac{(b-a_0)\prod_{k=1}^{n}(z-a_k)}{\prod_{k=1}^{n}(a_0-a_k)};\ \text{consequently,}\ f(z)=z+\frac{(b-a_0)\prod_{k=1}^{n}(z-a_k)}{\prod_{k=1}^{n}(a_0-a_k)}$$

Where z is any point in the given object set S (an element in our coordinate array). Notice that on the control points we have:

$$f(a_0) = a_0 + \frac{(a_0 - a_1)\ldots(a_0 - a_n)}{(a_0 - a_1)\ldots(a_0 - a_n)} * (b - a_0)$$

$$f(a_1) = a_1 + \frac{(a_1 - a_1)\ldots(a_1 - a_n)}{(a_1 - a_1)\ldots(a_1 - a_n)} * (a_1 - a_0) = a_1$$

etc.

One thing to keep in mind is that one can get a close match to the original object if the control points are replaced to their original positions in the opposite order they were moved. For example, suppose there are three control points, A, B and C. If B was moved, then C, then A the user must put A back to its original place then C then B to get a close match to the original object scene. Since the control points are moved by just selecting a pixel within a five pixel radius, there is a cumulative chance of error each time a control point is selected. This results in a slight distortion of the original object. This was necessary to remove a frustration from the user interface. A single pixel is very difficult to select with the mouse.

Example Screens

This design tool uses a technique which manipulates a scene with a continuous deformation. Smooth curves remain smooth while sharp edges remain sharp. We have previously shown how this technique is a conformal mapping.

After the user defines an object or scene and chooses the control points, it is ready to be manipulated. The user may define up to four control points. Control points are shown as a five pixel "star" configuration. The user changes the object by "grabbing" a control point with the mouse and dragging it. To make the interface easier to work with the system accepts any mouse click within a five pixel radius of the control point.

As shown in the first group of screens (appendix A.1 to A.3) we can manipulate a pennant object by moving the control point at the far right end. We move the end upward and see the angles in the grid design remain the same at the intersection points. In the next set of screen prints (appendix A.4 to A.10) we pull the end across the pennant itself. To preserve the angles of intersection the pennant appears to twist as it is folded over itself. If one brought the control point back to its original position, the flag would go back to a form close to the original screen, since the control points are chosen within a five pixel radius of the actual point.

In a different figure, we can see how the object is restored

if we return all the control points back to its original position. In appendix A.11 to A.14 we moved control point 3 away from its original position (A.11) and then moved it back again. One can see from the final screen (A.14) the object is slightly distorted. We saw this as being irrelevant because of the improvement in the user interface.

To show how this tool can be used for animation our last set of screens (appendix A.15 to A.18) shows a typical cartoon face being manipulated in our system. Some of the control points are not on the object. This shows us how it is the complex plane being acted upon and not the figure. At first we only moved the bottom control point, then we moved the two higher ones. With this tool we not only changed the shape of the original face but the direction of vision.

These are the only simple examples of how a figure can be manipulated. The possibilities are limited only to the user's imagination.

Conformal Mapping Animation Program

Overview

The Conformal Mapping Animation Program, or CMAP, is user interactive and mouse/menu driven. It is written in the C language and utilizes the X windows, chosen for portability and familiarity. The system allows the user to manipulate an object by representing that object on the complex plane. The user defines the object as a set of objects and up to four control points. These control points may be anywhere on the complex plane, but generally will be a part of the main object (for instance, at the principal vertices). The user will move the control points using the mouse. Through a technique similar to rubberbanding, the object (or a fraction of its points) will "follow" the cursor as it is manipulated by the changing control point. Each control point can be moved, but only one at a time. A more detailed explanation of the user interface will follow. Initially, the discussion will concentrate on the hardware/software platform, with an emphasis on the X Window System.

The X Window System

The X Windows system played an important role in the development of this application. It has always been stressed that X was a truly machine independent graphics environment, and this held true in the multi-platform environment at RIT. Throughout the course of development, the hardware platform changed three times. Each transition was virtually effortless. X Windows' fundamental strategy is that the base window system is its foundation [JONE 88]. The base window system is to deal with the mechanism of drawing output to the screen. The policy of user interaction is left to the X application program. The handshaking between the display hardware and the application is done by the X network protocol. It is this network protocol that gives X the device/vendor independence. Another strength of X is that it provides the mechanism for a variety of human interface strategies while leaving the implementation of those policies to the application. The C subroutine library, XLib, is one of the programmer's interfaces to the network protocol. Other interfaces to the protocol are high-level X toolkits. This application dealt only with XLib which, once interfaced with protocol, proceeded to interface with the base hardware.

The XLib library gave the application access to many human

interface fundamentals, but was not so structured as to inhibit designers' creativity. The first thing an applications programmer must do is create a connection between the application and the workstation. This is called a display connection which is a logical network connection between the application and the workstation. Using the environment variable DISPLAY, the application now has the ability to run on a CPU different from the one containing the graphics hardware. It is this that allows CMAP to run on a VAX/Ultrix machine and display on a Digital workstation running VMS. A further discussion of this configuration will follow. When one works on an X workstation, the hardware is running an X network server program. It is this program that handles requests from application programs. These requests include opening a display connection and creating windows.

The fundamental resources X provides for human interface design are: windows, graphic context, fonts, color maps, pixmaps and cursors. The resource that gives flexibility to a designer is graphic context. It is here that one can specify how graphical output is displayed. Properties which may be altered are background and foreground color, line width and style, and patterns for the background. XLib provides functions to alter these properties. An exciting extension to CMAP would be to utilize the colormap or color look-up table provided in X.

Events in X are instances where the user interacts with the computer using the mouse or the keyboard. Events come back

to the application through an event queue. These events are time stamped. Although X Windows' event handling mechanism is one of the more powerful features, it is also the most difficult for a novice to master. To use the event queue most efficiently, the applications programmer must explicitly plan for and state the exact events the system will utilize. The events used in CMAP are: Mouse Button 2 Press, Mouse Button 2 Release, Key Press, Mouse Button 2 Motion, and Window Expose. The user interface of CMAP will be described further in the following section, and the use of these events will become apparent. The most complicated event is the Button 2 Motion event, since its use has to do with timing. When the rubberbanding technique was first incorporated into the system, there was a serious problem with the processing time and the final output. The system should provide the illusion that the new image of the object can keep up with the motion of the cursor. Unfortunately, the event queue would get packed with Motion Notify events while the processing was still operating. This problem was alleviated by the use of the time stamp on each motion event; then the system could take an event every 200 milliseconds. This improved the illusion of rubberbanding. However, due to the limitations of processing time, equipment, and overall machine usage, this illusion could not be complete for all cursor speeds, but the time stamp on events helped it come close. Event handling in X Windows is very interesting and powerful.

As mentioned above, CMAP finally ran with the best performance on a Digital X Windows workstation with processing on an VAX running Ultrix. The reason behind this configuration stems from X Windows' development of LAN compatibility. X wanted the capability to run applications stand-alone as well as a mainframe with a variety of workstations linked in like terminals. CMAP was run in both configurations. The transparancy of changing platforms for the application programmer is ideal for a multi-platform environment. There is no need to modify a program for changes in hardware configuration. The workstation runs the X Window server and window manager while remotely logged into two other processing units (see figure 28a).

While processing may be on other CPU's, the display on the workstation and all locally registered events appear as though the program was running locally, due to the transparency of the X network protocol and the -DISPLAY command line argument or DISPLAY environment variable. The display connection is the first thing that should be established in an application. Since this setting is external to the program, there is no need within the program to tailor the code to the configuration. Once the display connection is set, the application knows where to send graphical output and where to get user interactive events. The remote machine just needs to do the processing and send/receive from the workstation accordingly.

The best way to see just how powerful X is on a network

is to show all the possible cases it can handle. On the next page you will see various ways an X application can run [JONE 88]. The rectangular boxes represent CPU's and the machines connected to these boxes represent workstations.

The CPU's will be referenced as one, two, three, and four. For instance, designate one as an Ultrix VAX, two as a Digital workstation, three as an AT&T 386 with two display terminals, and four as an AT&T 386 with three monitors. The first scenario CMAP used was two. Since the system ran on the local CPU and the DISPLAY variable was set to the local screen, everything was straight forward. CPU two also shows how a Digital workstations must be set up to run applications remotely. Once remotely logged in to the Ultrix VAX (CPU one), the environment variable DISPLAY had to be set to "doc::0.0" (doc was the name of the workstation). The two colons tell the system that the network establishing the remote connection is a DECnet. Single colon would disignate a TCP/IP connection. The first number after the colon represents the workstation number attached to the local CPU. The next number, after the period, gives the monitor screen number. One can carry the statement format down to CPU's three and four.

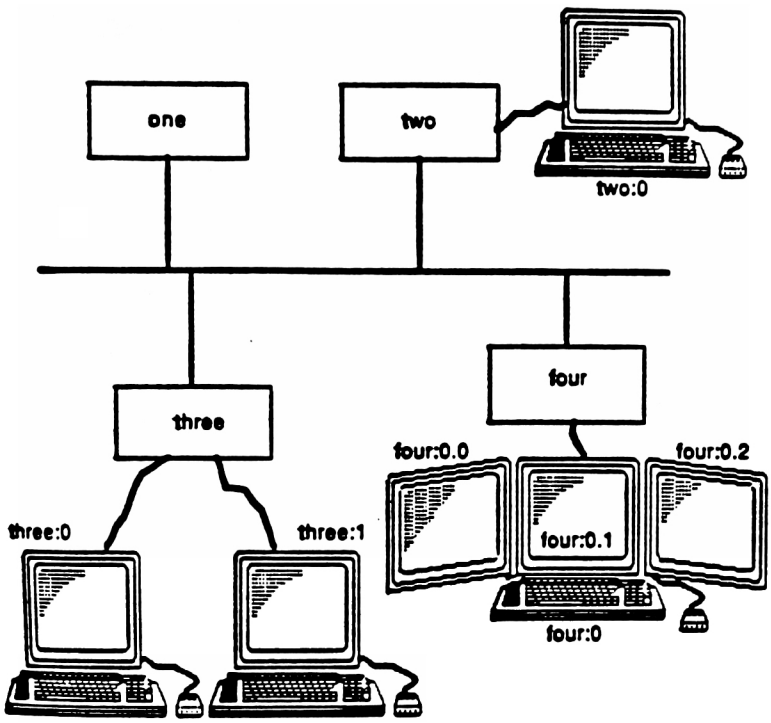


Figure 3-1. Local-area-network configuration.

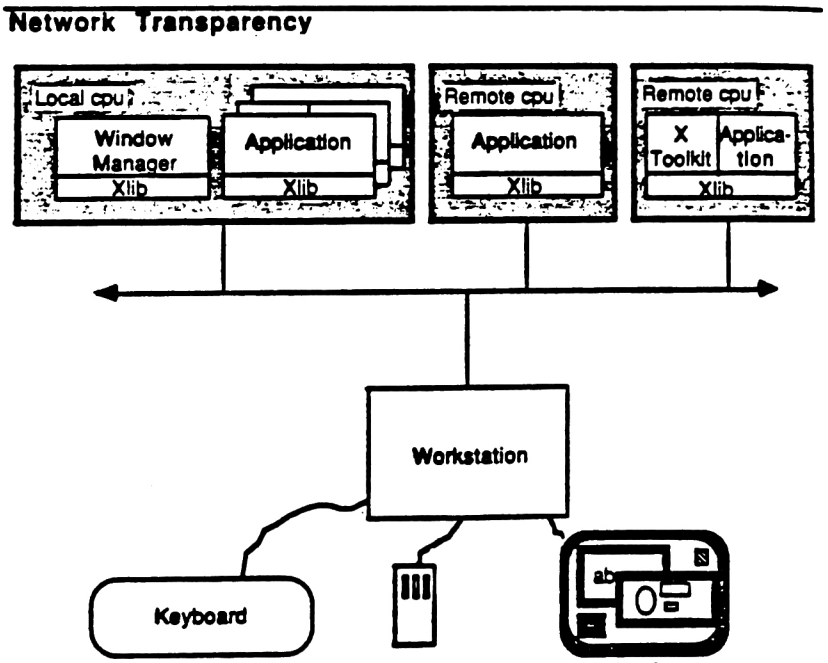


Figure 1-2. Workstation running local and remote applications.

Users Manual/Example Session

Getting Started

To begin CMAP, make certain you are on a workstation running the X Windows server. Generally the xinit command is all that is needed to initiate an "xterm window". Once an xterm window is up, type the name of the application, in this case "cmap". An exception to this procedure applies if a remote CPU is used for the processing. In this case the environment variable DISPLAY must be set to tell the application what display is being used. Once the application has been started it will prompt the user for a data file name. The data file describes the number of objects, types of objects, and certain properties given to these objects. At the end of a file the number of the control points (up to 4) and the coordinates of those control points are listed. Here is an example:

File Content	Commentary
2	Number of objects
1	Type of object (1 for line)
140.0 140.0 280.0 280.0	End points of line
2	Type of object (2 for circle)
140.0 140.0 50.0	Center coordinates and radius

3	Number of control points
140.0 140.0	Coordinates of control pt 1
280.0 280.0	Coordinates of control pt 2
90.0 140.0	Coordinates of control pt 3

The name of this file should end with the .dat extension.

Once this file is read into the application, the data structure which holds the original object is loaded. This structure is an array of structures called objects. The object structure is a union of the possible data structures representing object types (lines, circles, etc.). As each object is read in, the points included in its image are read into an array of complex coordinates. It is this array which is manipulated throughout processing. The control points are stored in two arrays of complex coordinates, control and cntldraw. Control is used to hold the original control points and cntldraw stores the control points as they are moved. Once the original object is drawn, the system waits for user interaction. The user can either choose to move one of the control points using the mouse, bring up a menu using the "m" key, or quit the session using the "q" key.

The user moves one of the control points by first positioning the cursor within a 5-pixel radius of one of the control points. Keeping the second button of the mouse held down, the user can continue to move ("drag") that control point. The cursor will

then move the selected control point with the object rubberbanding to keep up. As long as the second mouse button is held down, the system will only draw a select number of points in the image array. This could be all or a user-selected fraction of the points. The user can change this factor by using the "m" key to bring up a menu. This menu allows the user to change the number of points being processed from every 10, 20, etc. Once the pop up menu comes up, highlight the desired choice and let go of the first mouse button. As objects get complex, processing time between control point moves may be overwhelming and degrade the rubberbanding performance. We have discovered by experience that allowing all the points to be drawn while a control point changes is generaly worth slowing the process down.

When the user does not wish to continue to move the control point, the second mouse button is then released. This sends a Button Release event to the application. The application then knows that it is to draw the complete manipulated object at this final placement of the selected control point. Now all the points of the scene will be processed and output to the screen. The system will again wait for user interaction. Any of the control points may be chosen; one of them is in its new location.

CMAP as an Animation Tool

Currently, CMAP has exists as a prototype. It is not ready to be used as a full fledged animation tool. Rather, it is a shell around a theoretical concept. In this section we will discuss the enhancements needed to convert the present application upgrade from prototype to tool. These enhancements are largely those involving the user interface.

Scene Construction

The user must now design an original object by plotting its points and encoding them in a structured data file: this is tedious and frustrating. A complete system needs a graphics editor to allow the user to design objects with the mouse and keyboard.

The graphics editor may be like MacPaint or MacDraw. This user friendly design would not only allow different object types to be easily drawn, but also include such features as rotation around a given point, reflection over a given line, filling of closed objects, and color. Our graphic editor, will also allow the user to decide where the control points would be placed. Once this editor is used, the data on the screen can be named and stored in a bitmap file for future reference.

A library of images could also be used to develop a scene. This will give the user an unlimited supply of scenes to use in the system.

Animation

To supply an animated sequence where inbetween frames are important, the system must save the history of control point positions for each frame. It must also provide a means to do away with frames not wanted. This would be done by keeping a record of control point positions throughout the course of the session in an output buffer. Through an animator's pop up menu, the user can take away control point positions, thus take away inbetween frames from the output buffer. Again, with the use of the animator's popup menu, the user should be able to name and store this movement sequence associated with the scene file. Given the sequence file, the same control points are understood to be defined. At a later time, the user can then call up the scene file with the related movement file and display the movement sequence automatically.

Scene Design

If the user is interested in only producing one final picture, then inbetween processing speeds may be greatly reduced. In this case there would be no need to store the control point positions as the object is acted upon or keep track of the recording buffer size. The Mode pop up menu would allow

selection of animation or still mode. At the end of the still mode, the user may wish to store the final drawing in a named scene file. This stationary scene may then be used by the graphics editor or the CMAP tool.

To summarize, the three major enhancements to CMAP would be a graphics editor, a Mode menu, and an Animators' menu. The best approach to the menus would be a menu bar at the top of the screen, which would be available during the course of a session. Non-applicable menus would be inactivated, and dimmed. An example of this would be the Animator's menu while the session was in still mode. Other general menus could be added. The points per move menu could be carried over from the original CMAP prototype. A Background menu to enhance the background pattern or color would allow changes behind the scene without affecting the manipulation of the complex plane in the forground. The rubberbanding illusion can be tailored to the speed of the computer or to the preference of the user by a Speed menu. This menu would let the user decide the frequency of Button 2 Motion event accepted on the event queue. Default settings would be set in a user preference file during the first run of the application. There would also be a utility menu to allow changes to these default settings.

The still mode or graphic editor previously mentioned can be used in another software application. The final bitmap can be used as input for another application's display, perhaps when designing menus. This would allow application developers

to use this tool to develop systems in X.

Other Applications

Although CMAP was developed primarily for animation applications, conformal mapping can also be utilized in many engineering applications. Theories dealing with fluid flow, electrostatics, heat flow and stress dynamics have fundamental concepts based on conformal mapping. The following discussion will concentrate on these topics using conformal mapping and extending the application CMAP.

In the case of fluid flow, the assumption that the flow is in two dimensional space is taken to allow a mapping to CMAPs complex plane. Other necessary assuptions are: a steady flow, constant density of the fluid, and no internal pressure in the fluid itself. With this in mind, simulation with regard to special flows may be done. The system can accept the user's selection of sources and sinks locations. Then, through a data file, one may describe the flow itself in terms of velocity and direction. The user can then introduce obstacles. The system would use these obstacles much like the control points in the prototype. If the obstacle were moved, then the flow would change accordingly. For aesthetics the system would query the user for distance between the lines representing the fluid flow.

In electrostatics, the flow being simulated would not be molecules of a fluid but electrons. The system can have an electromagnetic flow which would be affected by certain points with different magnetic charges and intensities. The simulation would show how these charges determined the flow and direction of the electrons. Different weighting strategies could be put on the magnetic points as well as their charge.

CMAP may not be well suited to address the complexity of heat flow properties. Simulating the the varying temperature distributions in solid objects, heat flux, and thermal conductivity which is dependent upon the material conducting the heat transfer. Manipulating control points in conformal mapping would probably not be mathematically useful in this engineering context.

However useful these engineering theories may be, the focus of this thesis was computer graphics and its relevance to animation. Therefore, let these topics stand as suggestions to expand this thesis.

Bibliography

BURD 81 — Burden, Richard L.; Numerical Analysis 2nd Ed., Prindle, Weber and Schmidt, Boston 1981.

BRUC 75 — Bruch, John C.; "The Use of Interactive Computer Graphics in the Conformal Mapping Area", Computers and Graphics, Vol. 1 pp. 361-374.

CHUR 60 — Churchill, Ruel V.; Complex Variables and Applications, McGraw-Hill Book Co., NY, 1960.

FOX 84 — Fox, David; Waite, Mitchell; Computer Animation Primer, McGraw-Hill Book Co., NY, 1984.

HAYW 84 — Hayward, Stan; Computers for Animation, Focal Press, England, 1984.

JONE 88 — Jones, Oliver; Introduction to the X Window System Prentice Hall Inc., Englewood Cliffs, NJ, 1988.

KERN 78 — Kernighan, Brian W. and Ritchie, Dennis M.; The C Programming Language, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1978.

KREY 72 — Kreyszig, Erwin; Advanced Engineering Mathematics Third Edition, John Wiley and Sons, Inc. NY, 1972.

MAGN 85 — Magnenat-Thalmann Nadia; Computer Animation Theory and Practice, Springer-Verlag, NY, 1985.

MAGN 88 — Magnenat-Thalmann Nadia; Thalmann, Daniel Ed.; New Trends in Computer Graphics, Proceedings of CG International 88, Springer-Verlag, NY, 1988, pp. 54-63, pp. 621-630.

REEV 81 — Reeves, William T.; "Inbetweening for Computer Animation Utilizing Moving Point Constraints", ACM Computer Graphics, Vol. 15, Num. 3, Aug. 81.

ROSE 87 — Rosbush, Judson; ACM Siggraph 87 Course Notes
Vol 7 "Advanced Computer Animation"
Vol 10 "Computer Animation 3D Motion Specification and Control"
Vol 28 "Computer Graphics and Animation in the Physical Sciences"

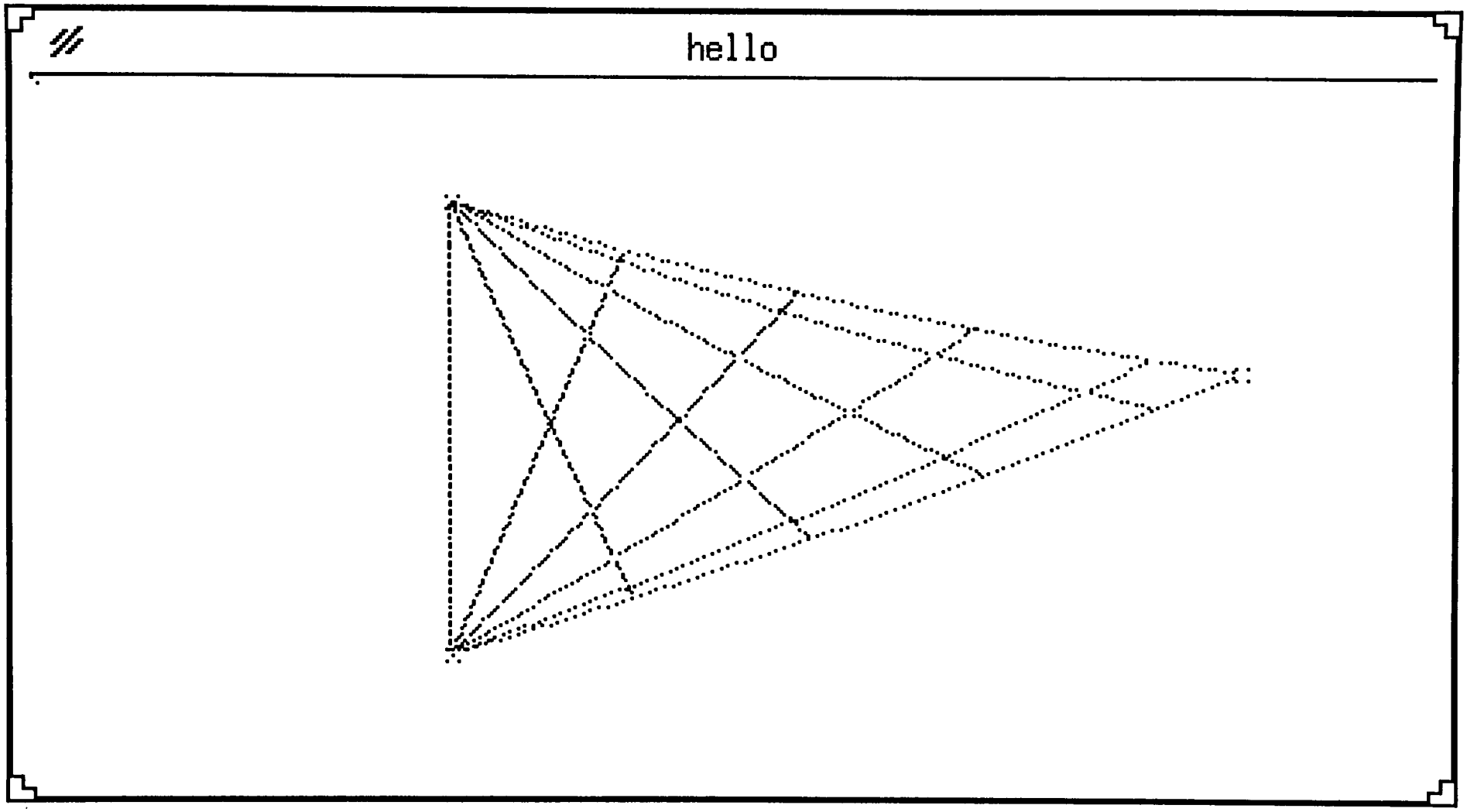
PENN 70 Pennington, Ralph H.; Introductory Computer Methods and Numerical Analysis, Second Edition, MacMillian Co., 1970.

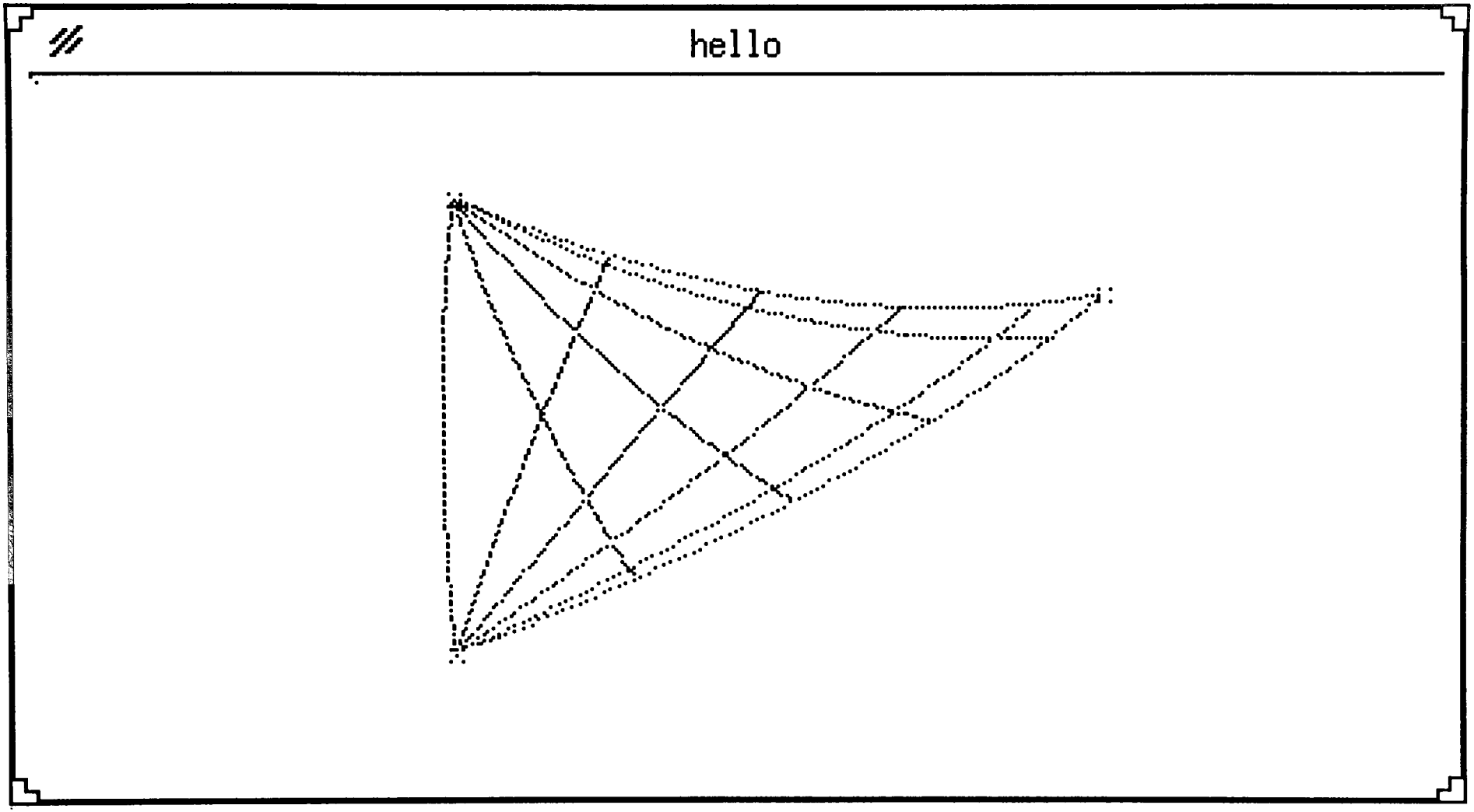
SCHE 68 Scheid, Francis; Theory and Problems of Numerical Analysis, McGraw Hill Book Co., 1968.

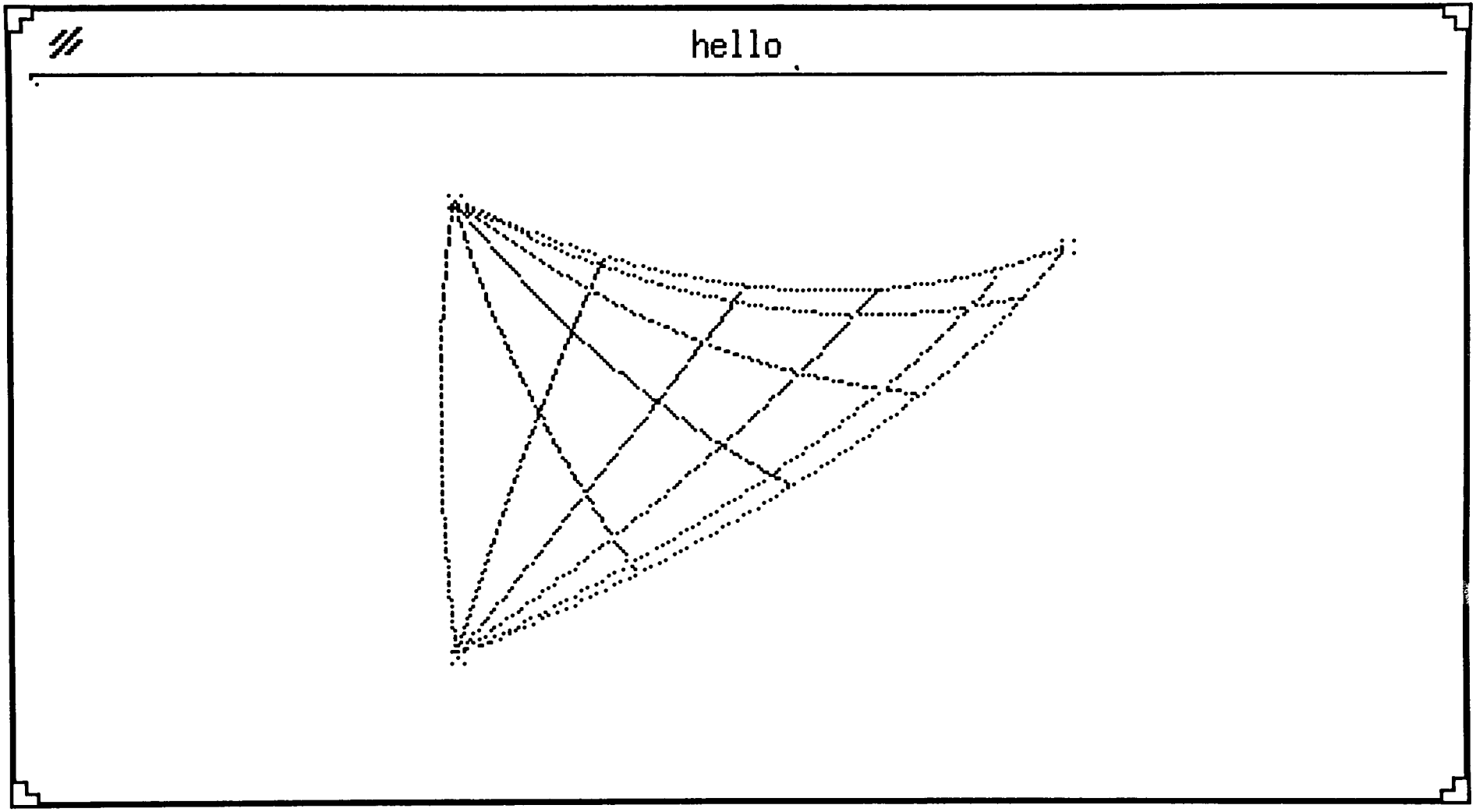
SPIE 64 Spiegel, Murray R.; Complex Variables, McGraw Hill Book Co., 1964.

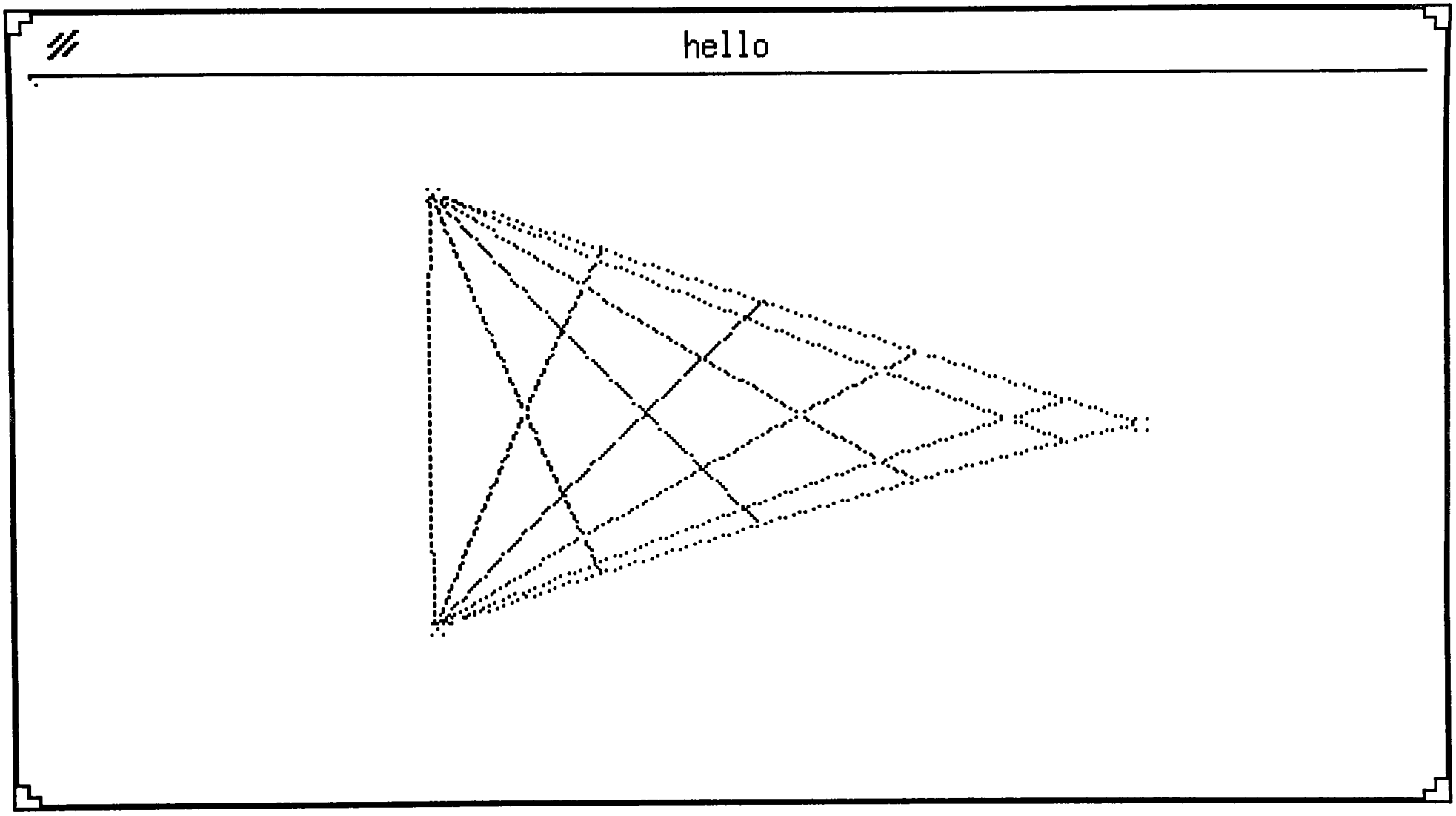
WHIT 81 Whitaker, Halas; Timing for Animation, Focal Press, London, England, 1981.

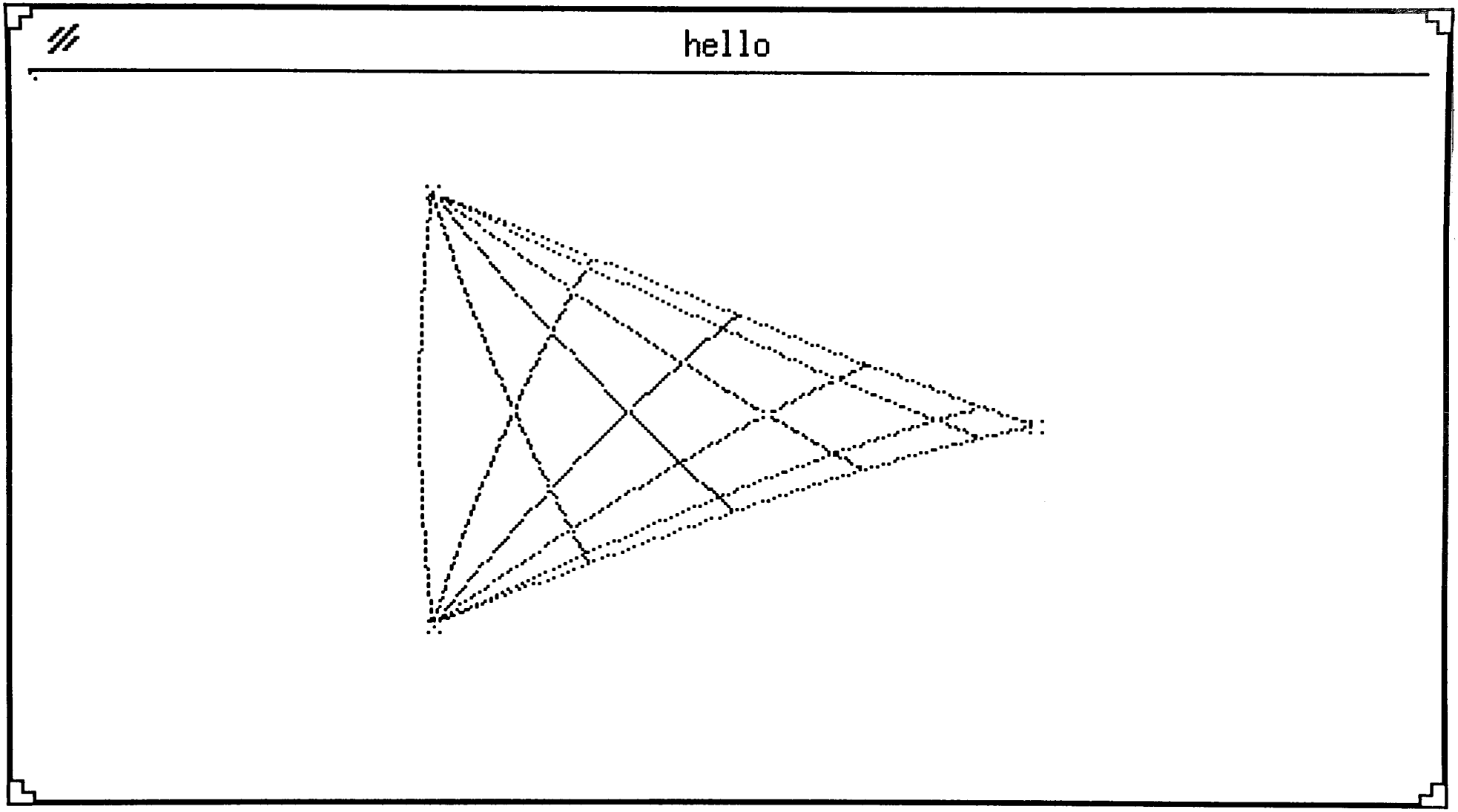
WILH 87 Wilhelms, Jane; "Torward Automatic Motion Control" IEEE Computer Graphics and Animation, April 1987.

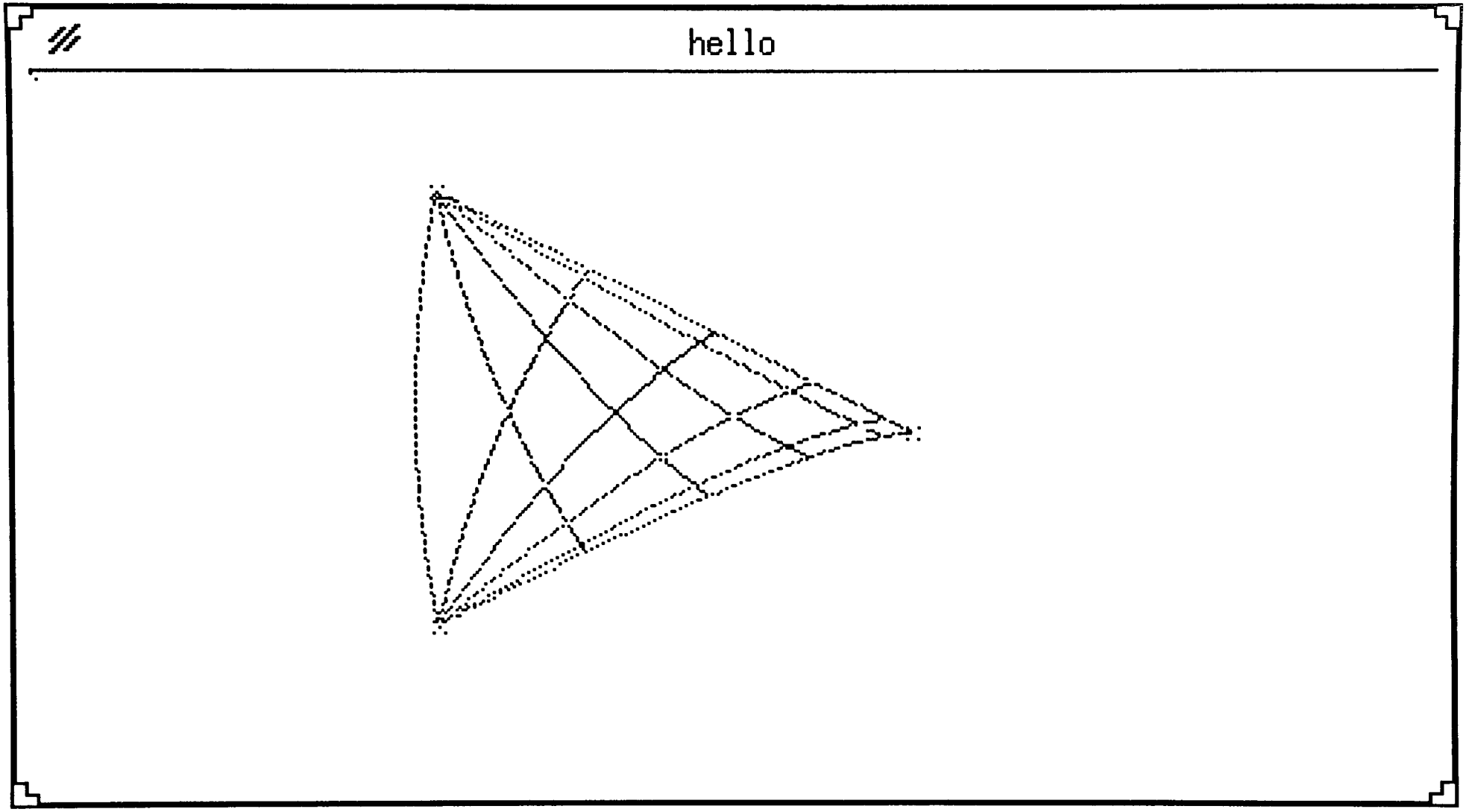

hello


hello


hello

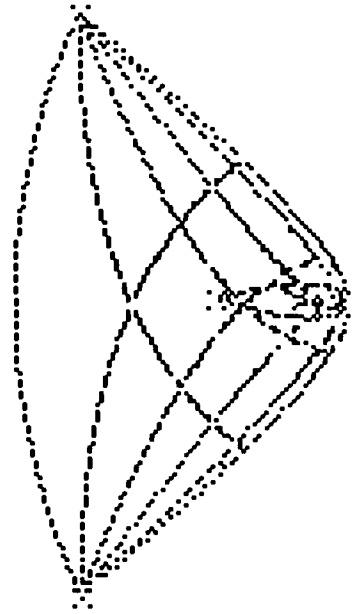

hello

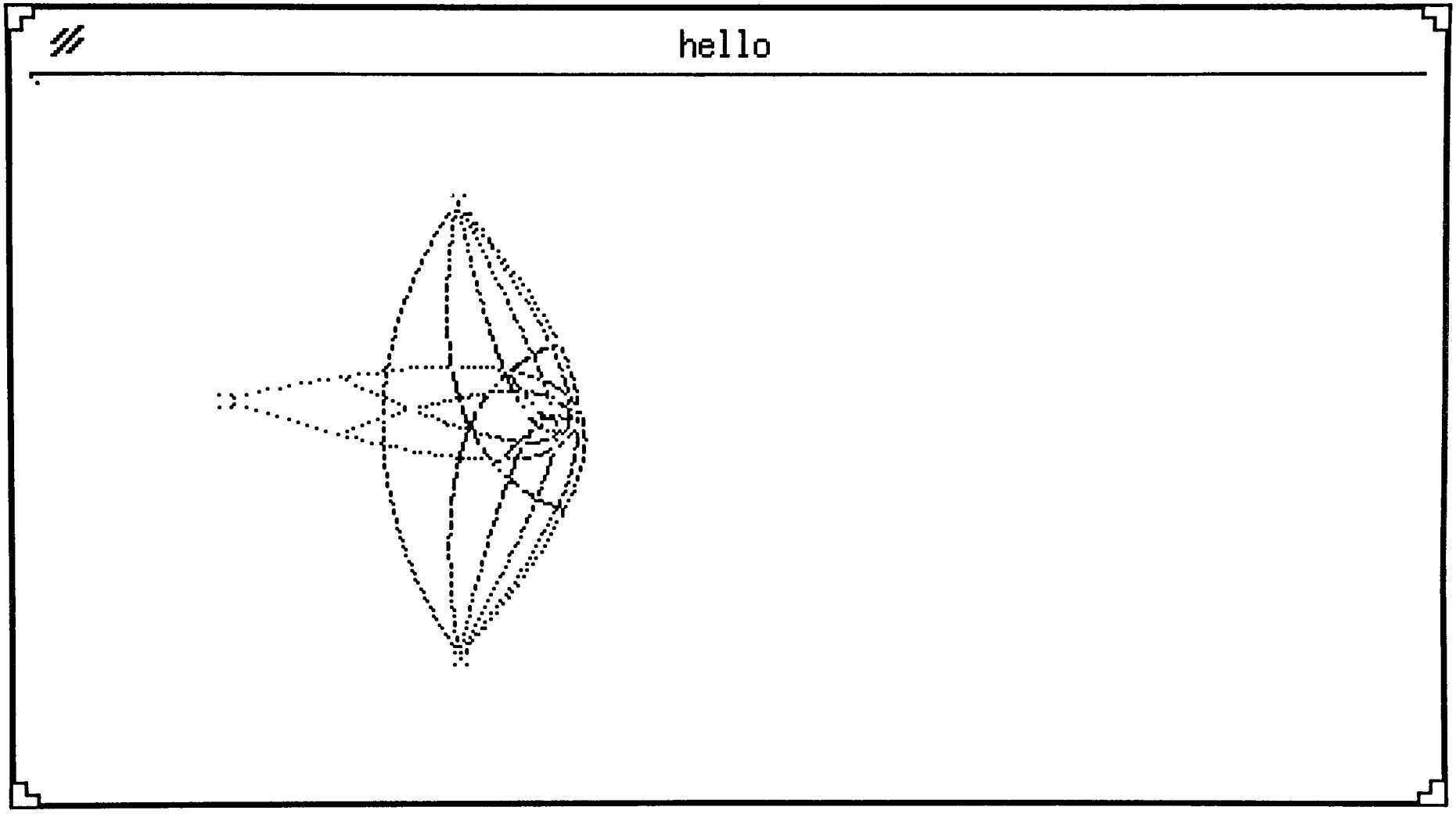

hello

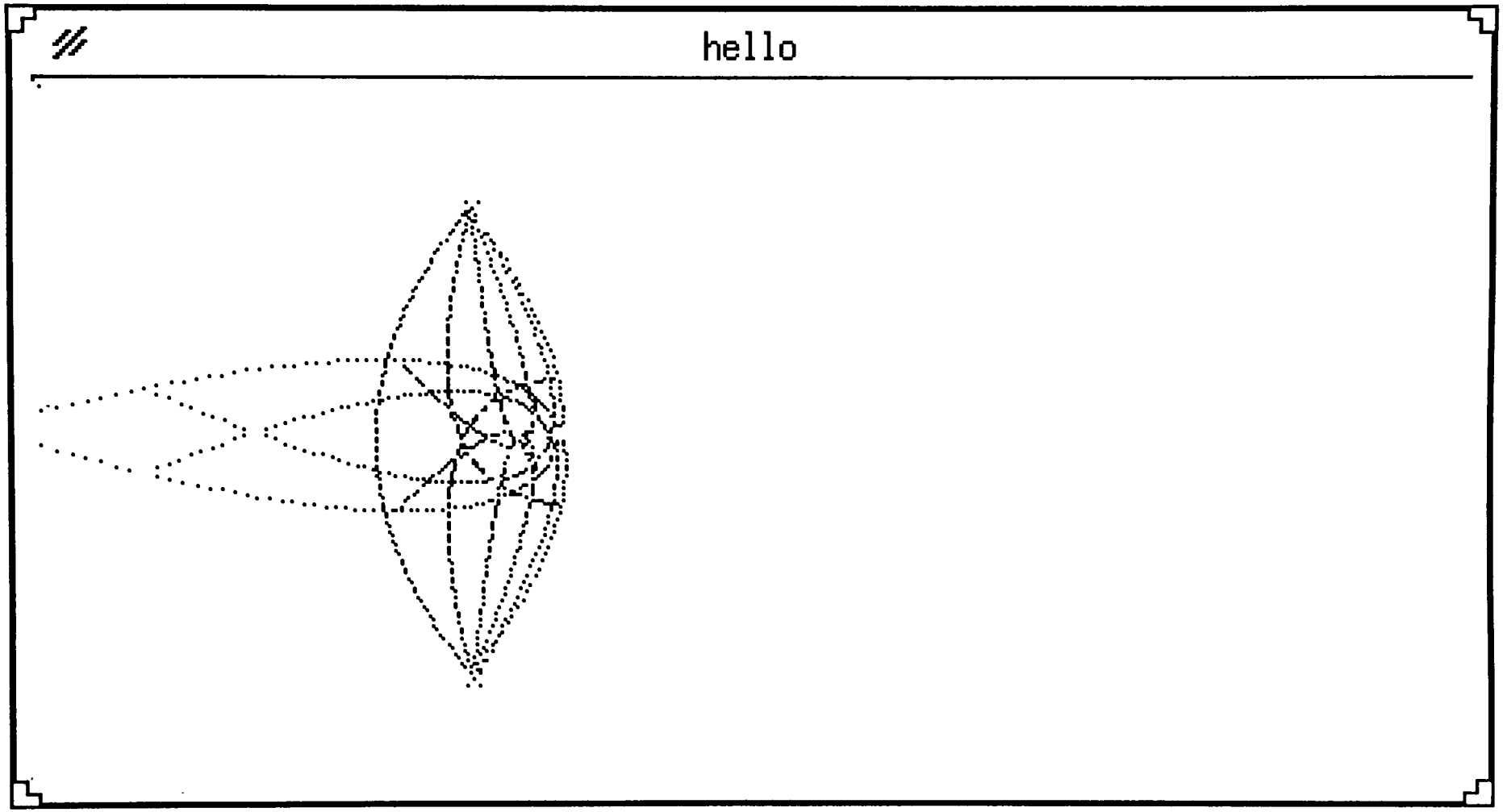

hello

hello

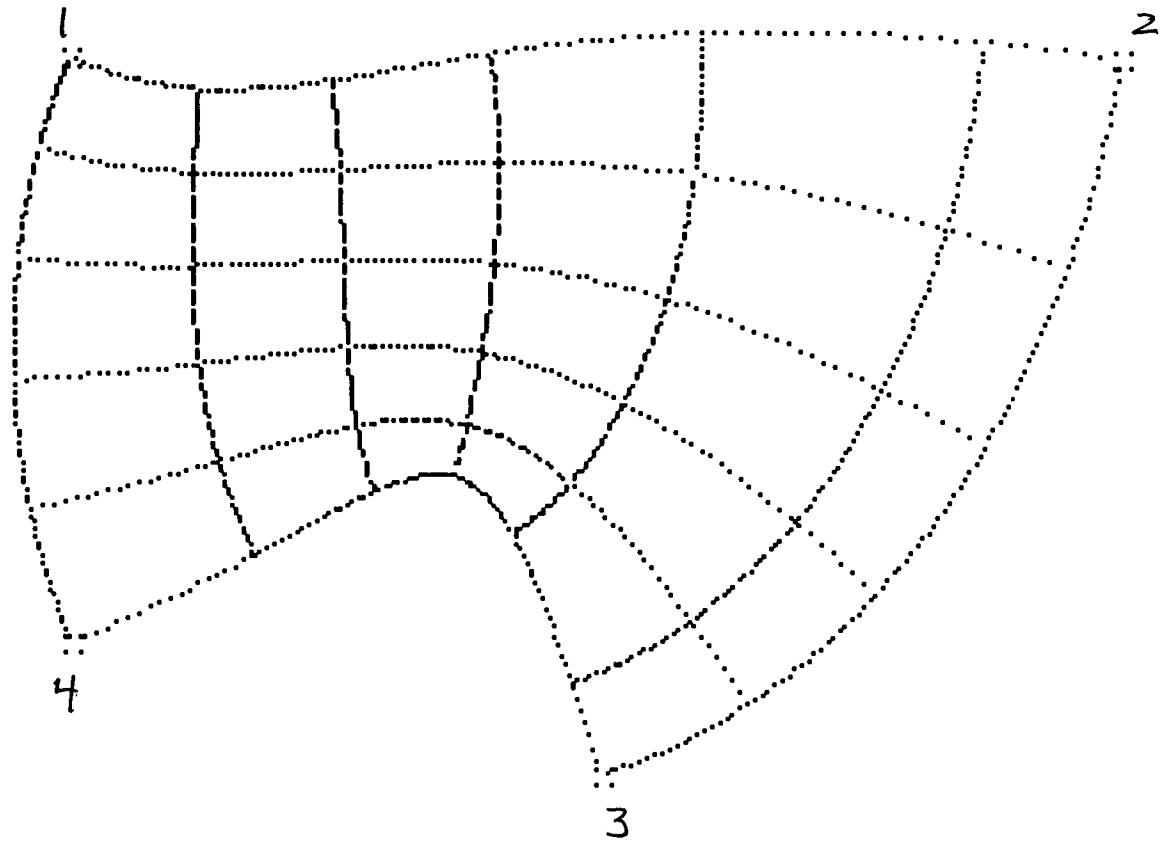
hello



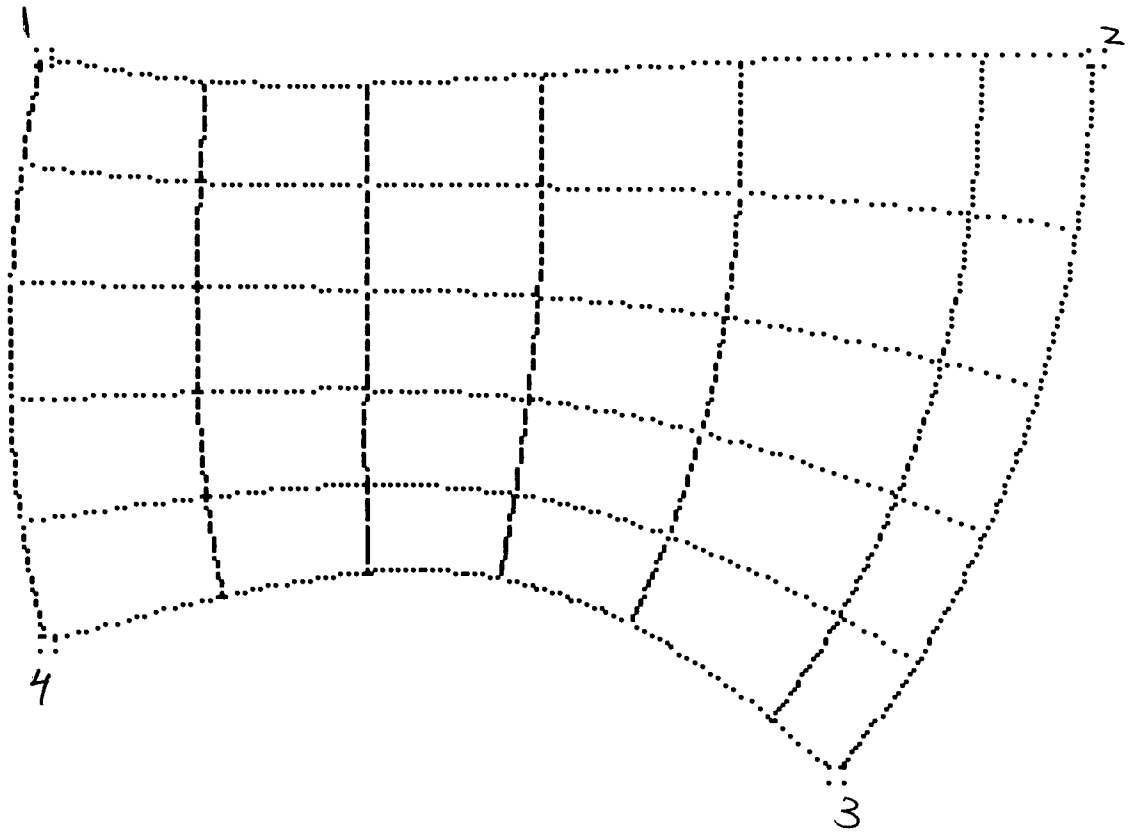

hello


hello

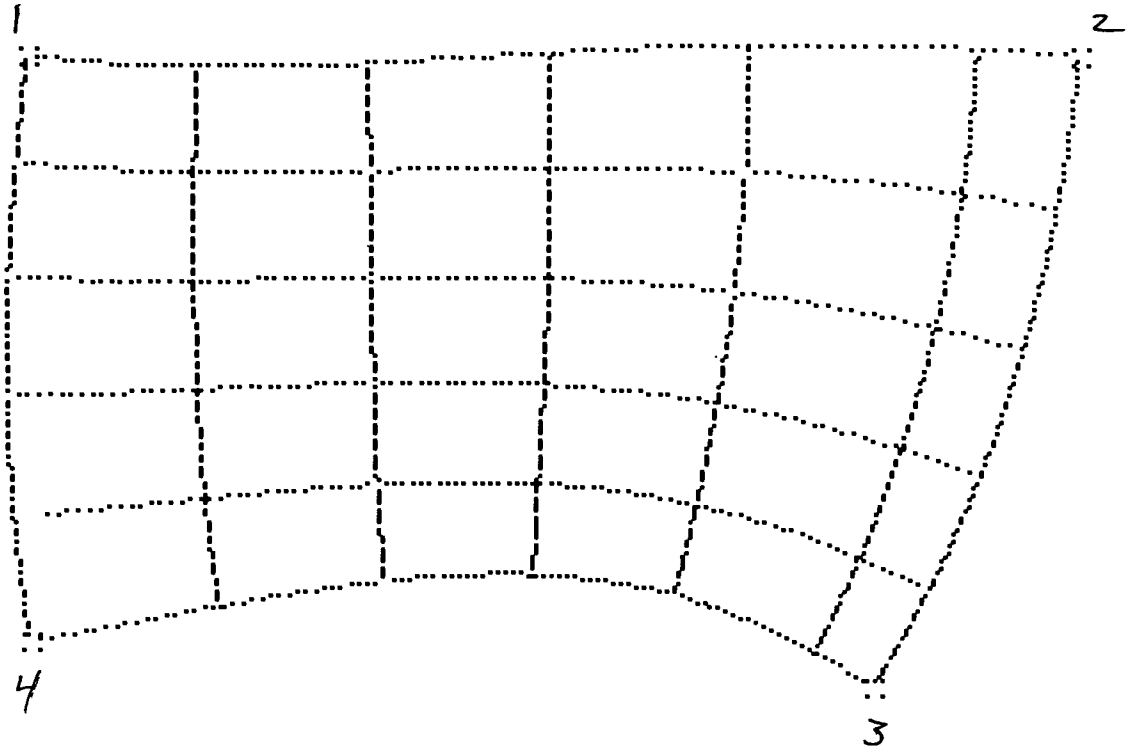
hello



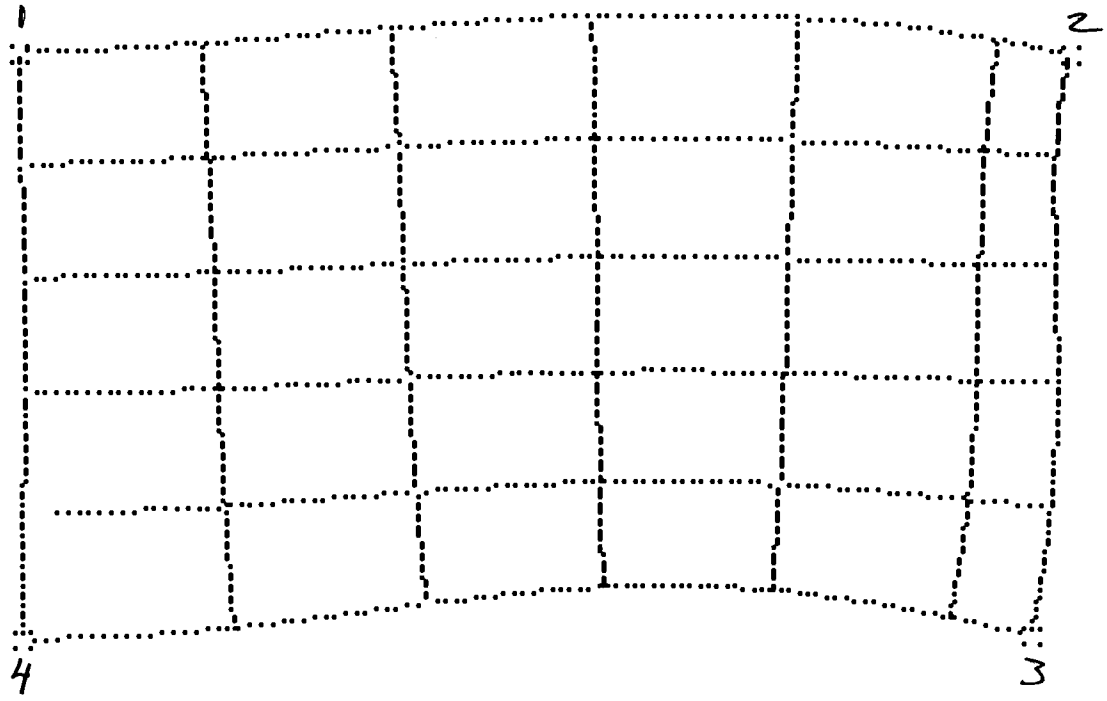
hello



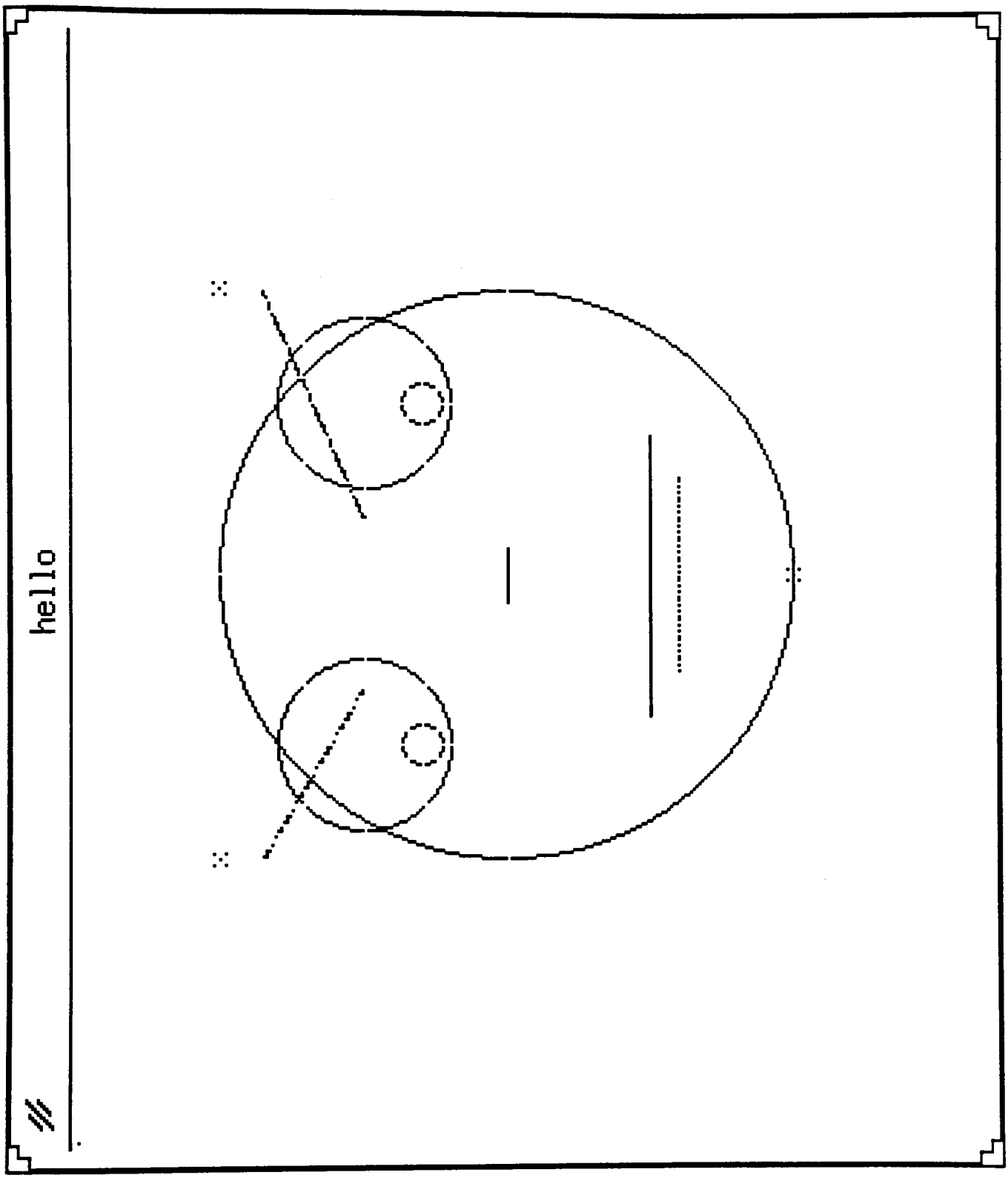
hello



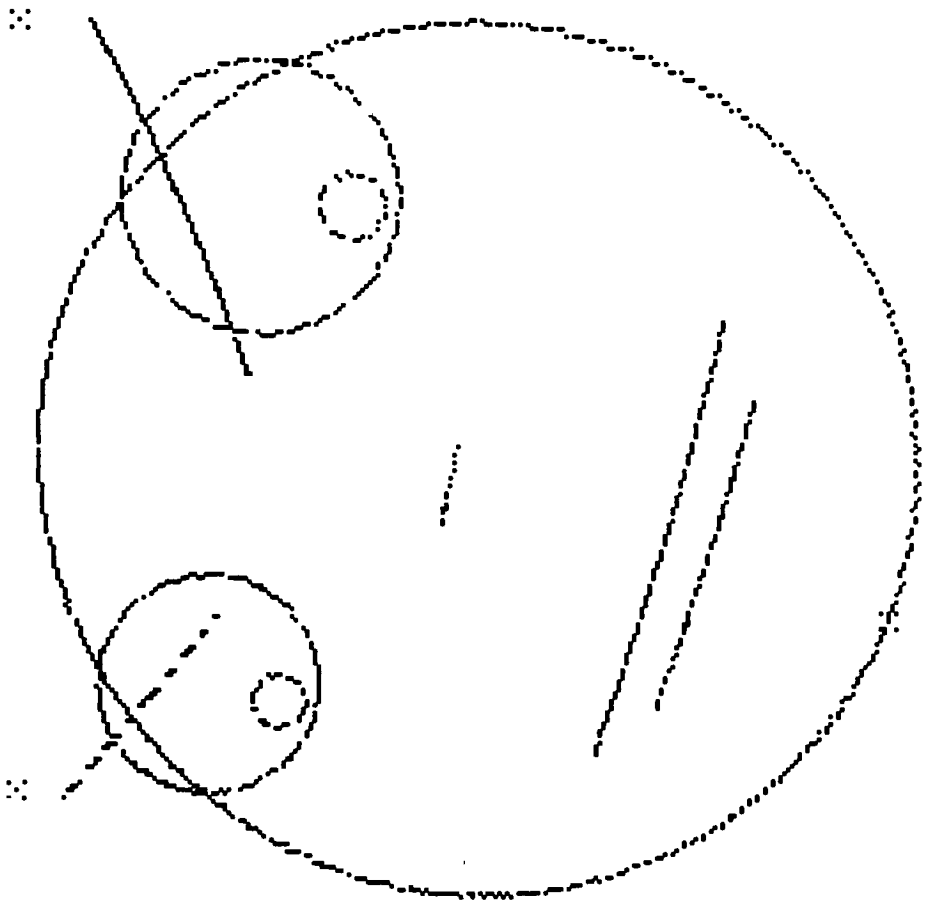
hello

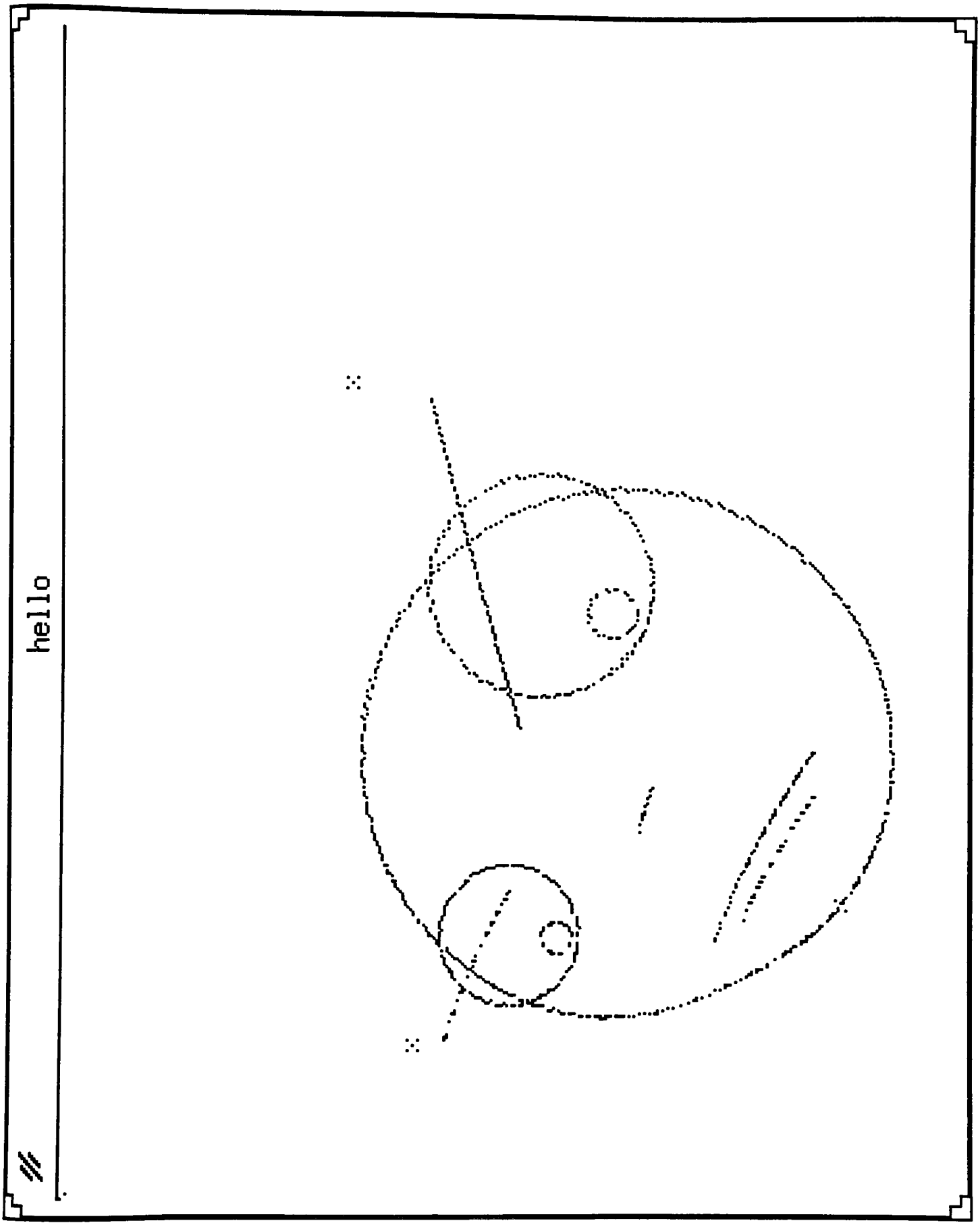


A4


hello

hello




hello

hello