A HIERARCHICAL GRAPHICS COMPOSITION MODEL

Chi-Leung (Andy) Lau
M.S., Montana State University, 1979
M.S., University of Oregon, 1976
B.S., Whitworth College, 1973

A thesis submitted to the faculty
of the Oregon Graduate Center
in partial fulfillment of the
requirements for the degree
Master of Science
in
Computer Science and Engineering

June, 1984

The thesis "A Hierarchical Graphics Composition Model" by Chi-Leung (Andy) Lau has been examined and approved by the following Examination Committee:

Richard B. Kieburtz
Professor and Chairman
Thesis Research Advisor

Robert G. Babb II
Assistant Professor

Steven R. Vegdahl
Research Computer Scientist
Tektronix, Inc.

DEDICATION

This thesis is dedicated to my wife Frances, who experiences deeper sorrow when I am sorrowful, and feels greater joy when I am joyful. Her continual care and encouragement are a big reason for where I am today, in all aspects of my life. Here and now, I am joyful!

ACKNOWLEDGEMENT

Special thanks are directed to Dr. Richard Kieburtz for his many valuable suggestions, and for his support and acceptance of new ideas throughout the preparation of this thesis.

Another person to whom I am deeply indebted is Linda Appel of Tektronix. She has shown the fine qualities of a professional librarian. Without her friendly and patient help, this thesis would still be lingering on.

TABLE OF CONTENTS

ABSTRACT

A Hierarchical Graphics Composition Model

Chi-Leung (Andy) Lau
Oregon Graduate Center, 1984

Thesis Research Advisor: Richard B. Kieburtz

1.0 Introduction:

The principal objective of this thesis is to demonstrate that the concept of hierarchical graphics composition is a workable model for creating, editing and managing computer-generated two-dimensional color pictures. As a result, a powerful tool has been developed which is easy to use and yet requires minimal hardware.

There are several things we want to accomplish in this paper: to introduce the concept of this hierarchial model and compare it with two other hierarchical models, to present the main algorithm of intersection of two convex figures whose edges can be circular arcs or straight line segments, to show an example using this graphics composition package, to describe the software and hardware environment for this package, and to include a user's guide for the package.

2.0 Concept:

A point is the fundamental unit in point set geometry, and it is denoted by an ordered pair (x,y) in the Cartesian coordinate system. Any collection (set) of points that can be described by certain mathematical rules is called a primitive set, or simply a primitive. We will introduce the primitives that we will be using shortly. For

convenience, we also consider text a primitive.

An object in our model is defined by the intersection of primitives. Objects form the foundation for our hierarchical model, which goes as follows:

$$\text{Picture} \subset \text{Groups} \subset \text{Members}.$$

In words, a picture is a collection of groups, and a group is a collection of members. Furthermore, each member is an object with assigned attributes, and each object (with the exception of text) is defined by the intersection of certain geometric primitives. Hence, an object can be referenced by different members (possibly with different attributes). There are four attributes to each member, they are pivot point, scale factor, rotation angle and color, thus allowing transformations on the objects.

The entire structure is depicted in Figure 1.

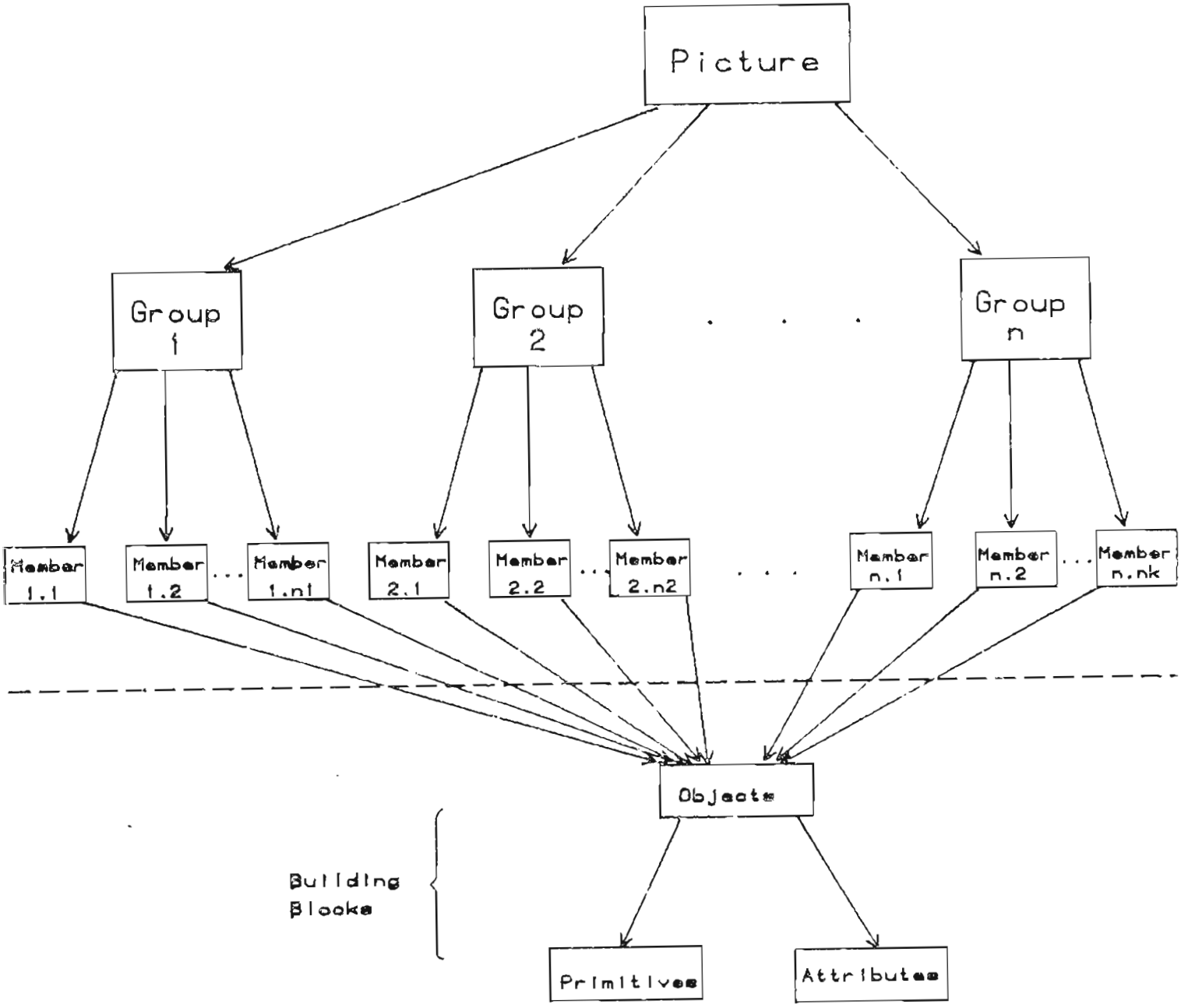


Figure 1.

In his article "Functional Geometry," Henderson [5] describes another approach for picture description. However, his is different from ours in several aspects.

a. The very basic notion for Henderson's approach is that of a line segment, whereas our basic notion is that of an area. Pictures in

his model are described in terms of objects formed by line segments. We describe our pictures in terms of objects obtained by intersecting certain areas.

b. Henderson's manipulation of line segments is based on some mathematical transformations, whereas we use set algebra to manipulate areas.

c. Both approaches are hierarchical, but with different organizations. Henderson's is well suited for generating repetitive patterns. That is, a new object can be obtained from an old one by applying functions such as rotation or reflection. Ours is suited to general purpose 2-dimensional geometric modeling.

d. Henderson's model is monochromatic, while color plays a very significant part in ours.

e. Henderson's does not explicitly support text as ours does, although our text support is severely limited by hardware. Also, Henderson's is limited to straight edges while we have more freedom with circular arcs.

Caesar [8] is another approach to picture description. Although it is also hierarchical in nature, it differs significantly from our model in many regards:

a. Caesar is motivated by its intended use, layout design of VLSI (Very Large Scale Integrated) chips.

b. The only geometric object supported in Caesar is the rectangle.

c. Although the user can specify his own color map, the number of colors supported by Caesar is limited to 5.

d. Transformation of an object is limited to reflection and rotation through an angle that is a multiple of 90 degrees.

e. Text is for labeling purpose only, and its size cannot be changed by the user.

f. The basic hierarchical unit in Caesar is a cell, which is comprised of colored, labeled rectangles. Each cell can be stored away individually. To elevate to a higher level, a new picture (file) is created and new rectangles can be drawn, or previously designed cells can be brought in and become part of the drawing. The old cells have now become subcells, and a new (and bigger) cell is created. This process can be repeated as many times as the user wants.

g. The most important part of Caesar is that it provides a post processing function that will allow data to be converted into a textual format (CIF) that will be acceptable for simulation or manufacturing of the chip. The ability of being able to go from graphics to simulation or manufacturing is the very essence of CAD/CAM.

2.1 Primitives.

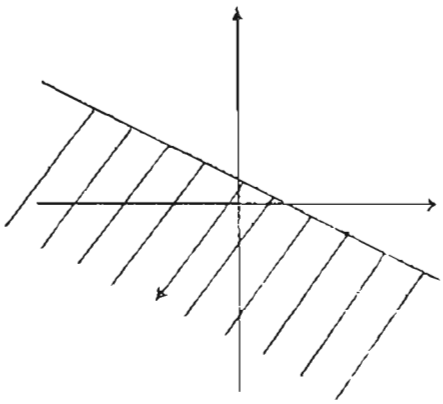
The primitives used in this geometric model are called HL, HG, VL, VG, CL, PL and TX, each of which is described below:

HL(m,b) = {(x,y) : y <= mx + b}, a half-plane with m being the slope and b the y-intercept of the line dividing the whole Cartesian plane into two halves. HL(m,b) refers to the lower half-plane.

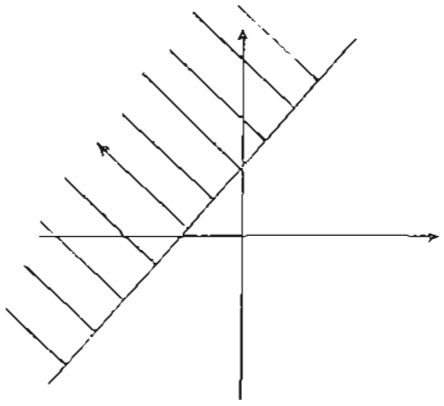
HG(m,b) = {(x,y) : y >= mx + b}, is the upper half-plane.

VL(c) = {(x,y) : x <= c}, is the half-plane to the left of the vertical line going through the point (0,c). Vertical lines must be treated as a special case because they do not have a well-defined slope that can be expressed as a real number.

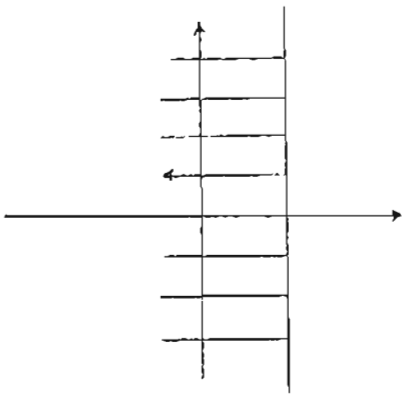
VG(c) = {(x,y) : x >= c}, is the right half-plane.



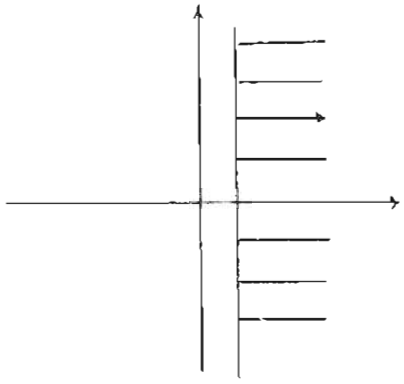
Lower half-plane, HL(-0.75,1)



Upper half-plane, HG(1,3)



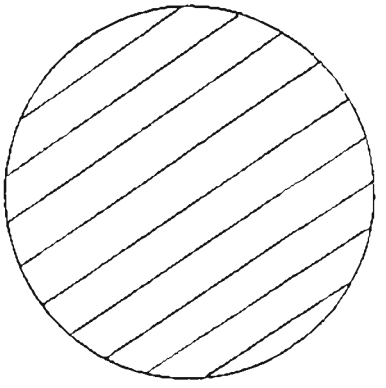
Left half-plane, VL(3.5)



Right half-plane, VG(1.5)

Figure 2.a

CL(h,k,r) = {(x,y) : (x-h)**2 + (y-k)**2 <= r**2}, is a disc with (h,k) being the center and r the radius of the circle enclosing the disc.



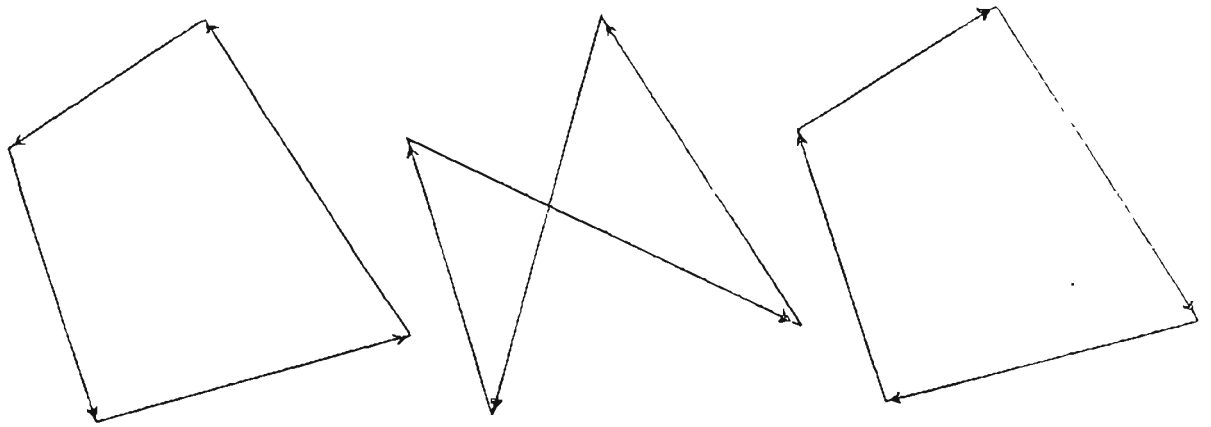
Disc, CL(0,0,5)

Figure 2.b

PL is a polygon, a closed area defined by a finite number of ordered vertices such that,

1. The initial vertex is also the final vertex.
2. A line segment connecting any two consecutive vertices will not intersect with any other such line segment.
3. The vertices are specified in such a way that when a line segment between one vertex and its succeeding vertex is traversed, the area will always be "to the left" of the segment.

For examples, let A = (5,0), B = (10,3), C = (8,10) and D = (0,8), then the set {A B C D} legitimately specifies a polygon, while {A C B D} and {A D C B} do not, because they violate conditions 2 and 3 respectively. (See figure below).



Valid polygon Invalid polygon Invalid polygon

Figure 2.c

TX = text, a string of characters.

In the above mnemonics, H stands for "Half-plane", G for "Greater than", L for "Less than", V for "Vertical", C for "Circle" and P for "Polygon". Other primitives, such as the remaining conic sections besides the circle, are not included in this model because of the time constraint.

2.2 Objects.

An object is an intersection of primitives, which incidentally, can be empty. Objects are not displayable; they are rather building blocks which define geometric shapes or characters to be written.

Since text is non-geometric, the meaning of intersecting text with another primitive must be defined. To avoid any confusion, a simple rule is provided here, that is,

Text $\cap$ Geometric Primitive = Text (commutative),

Text1 $\cap$ Text2 = Text2 (non-commutative).

Admittedly, the above rule does seem arbitrary. In fact, text is not central to this model, its inclusion is simply for completeness sake.

Here are some examples. An object of a diamond shape can be defined by the intersection of the four half planes, HL(-1,1), HG(1,-1), HG(-1,-1) and HL(1,1). This object may perhaps be named "Diamond" for later reference. An object of a wedge shape can be defined by the intersection of the disc CL(0,0,1), the half plane HG(1,0) and the half plane HG(-1,0). This object may be called, say, "Wedge".

2.3 Picture.

A picture is the overall composition of a drawing or the root of the hierarchy being discussed. It is identified by a name, and has the responsibility of defining the window of viewing. A window, which is constrained to be rectangular in shape, is specified by supplying its lower left corner and its upper right corner. The larger the window specified, the smaller the picture will appear on the screen.

2.4 Groups.

A group, referenced by its name, is just below picture in the hierarchy. Only groups are displayable. If there are five groups in the picture, for example, then any combination of these five groups can be displayed. The order of display appearance of groups follows the order in which group names are entered in the display command. Should overlaps occur, newly painted figures will overwrite those already drawn

on the screen.

2.5 Members.

At the bottom of the hierarchy is the member. Each member has its own name. To construct a member, first take an object (by name) and then assign attribute values to this object. One or more members can reference the same object, they are differentiated by the attributes they assign to this object. The attributes are pivot point, scaling factor, rotation angle and color.

For example, we can take the object "Diamond" as defined before and assign to it the following attributes:

pivot point: (10,10)

scale factor: 2

rotation angle: 45

color: 5

We then get a square (due to rotation of 45 degrees), with each side being 2*sqrt(2) in length (due to scaling factor), and having center at (10,10). The color will be color 5 of the coloring scheme. Incidentally, the color number starts from 0 to more than 15, (see B.2).

Due to the hierarchical nature of the model, access to any group or member is very fast and, changing any part of the picture is quite easy.

3.0 Operations:

There are essentially two types of operations. One type is concerned with objects, the other with the picture. We outline the key ideas here. For further detail, refer to Appendix B.

3.1 Object operations.

Object operations include creating an object, editing an object, deleting an object and listing names of the objects.

3.2 Picture operations.

a. At the picture level, one can list names of the groups, adjust the window of viewing, create and delete a group.

b. At the group level, one can define a member of the group, delete a member from the group, or list names of the members of the group.

c. At the member level, one can modify any of the four attributes. Changing the pivot point will cause a relocation of the member, changing the scale will make the member bigger or smaller, changing the rotation angle will make the member tilt according to the degree entered, and changing the color will let the member be painted with a different color.

4.0 Implementation:

We describe the implementation in detail. Some important algorithms along with the data structure are explained here.

4.1 Hardware.

Development of this project was done on a VAX11/780 running UNIX. Besides a terminal, a Metheus Omega 400 Display Controller and a Tektronix 690SR Color Monitor were used.

4.2 Software.

The main program, awlge (humorously stands for "Andy's Wonderful Little Graphics Editor"), was written in PASCAL, but the driver for the display controller was contained in the software package AXIA, written in FORTRAN by the Metheus Corporation. In order to make use of the subroutines in AXIA, it was necessary to write in C an interface so that awlge could call a subroutine in the interface, which in turn called the corresponding subroutine in AXIA. It was done this way because Berkeley Pascal cannot call Fortran subroutines.

4.3 Data Structure.

There are four major files in awlge, the Object Name File (Onf), the Object Description File (Odf), the Group Name File (Gnf) and the Group Display File (Gdf). In essence, each entry of Onf contains an object name and a pointer into Odf. The entry in Odf that is being pointed to by this pointer is the first primitive of the object. Definition of the object ends with a delimiter after the intersecting primitives. Similarly, each entry of Gnf contains a group name and a pointer into Gdf. The corresponding entry in Gdf is the first member in this group. Again, the last member of the group is followed by a delimiter. The exact coding of these files in the program is given in Figure 3 below:

```
const
    Length = 25;
    Onflen = 50;
    Odflen = 200;
    Gnflen = 20;
    Gdflen = 100;
type
    String = packed array [1..Length] of char;
    Pair =
        record
            A, B: real
        end;
    Objrec =
        record
            Nameobj: String;
            Size: Pair;
            Odfptr: integer
        end;
    Primitive = (LN, RE, RG, RL, HE, HG, HL,
        VE, VG, VL, CE, CG, CL, PN,
        PE, PG, PL, PT, NM, TX, EO,
        OHEAD);
    Objdesc =
        record
            Objcode: Primitive;
            case Primitive of
                OHEAD: (
                    Nameobj: String;
                    Size: Pair
                );
                LN, RE, RG, RL: (
                    Point1, Point2: Pair
                );
                HE, HG, HL: (
                    Plane: Pair
                );
                VE, VG, VL: (
                    Vert: real
                );
                CE, CG, CL: (
                    Hx, Ky, Radius: real
                );
                PN, PE, PG, PL: (
                    Npts: integer
                );
                PT: (
                    Point: Pair
                );
                NM: (
                    Objnm: String
                );
                TX: (
```

```
                    Textt: String
                );
                EO: (
                    Cont: integer
                )
        end;
    Grpname =
        record
            Namegrp: String;
            Gdfptr: integer
        end;
    Display = (OB, NX, GHEAD, WINDOW);
    Grpdisp =
        record
            Grptag: Display;
            case Display of
                WINDOW: (
                    Lower, Upper: Pair
                );
                GHEAD: (
                    Namegrp: String
                );
                OB: (
                    Member, Objname: String;
                    Pivot: Pair;
                    Scale: real;
                    Angle: real;
                    Color: integer
                );
                NX: (
                    Nextptr: integer
                )
        end;
var
    Onf: {Object Name File} array [1..Onflen] of Objrec;
    Odf: {Object Description File} array [1..Odflen] of Objdesc;
    Gnf: {Group Name File} array [1..Gnflen] of Grpname;
    Gdf: {Group Display File} array [1..Gdflen] of Grpdisp;
```

Figure 3.

4.4 Algorithms.

The concept of convexity of geometric figures plays a very important role in this model. Intuitively, a figure is convex if whenever we pick two points inside or on the boundary of this figure and

draw a line between them, no part of this straight line will be outside the figure. Primitives HL, HG, VL, VG and CL are all convex (PL will be handled separately in 4.7). Note that the intersection of two convex figures is also convex. This useful fact keeps other algorithms quite simple. As an example, the object "Wedge" that we defined before is convex, because it is the intersection of three convex figures, two half-planes and a disc.

Convexity is a relatively simple idea. However, to describe a convex figure whose boundary consists of more than just straight edges, or to manage intersections of such convex figures with other convex figures, it requires a detailed algorithm, which we wish to present here. For instance, how do we resolve the intersection of the object "Wedge" with another disc?

4.5 The vertex form.

We have defined the object "Wedge" to be the intersection of three primitives. However, such definition is not suitable for the purpose of displaying it, since most, if not all, display controllers need to know the "coordinates" of Wedge. Instead, letting p = square root of 1/2, we can say, "The boundary of Wedge is formed by drawing a straight line from (0,0) to (p,p), drawing an arc of unit radius with center at (0,0) from (p,p) to (-p,p), and drawing another straight line from (-p,p) to (0,0)." In this way, we have "spelled" out the "vertices" and the "shapes" of Wedge.

So, in awlge, a convex figure is represented by an array, Vxf (Vertex file), of records whose record type is given below as,

```
type Path= record
           Vertex: Pair;
           Shape: Objdesc;
           Status: integer;
           end;
```

where Pair and Objdesc were defined in 4.3, while the field Status will be explained later in 4.6. The only subtle thing to be pointed out is that this array is circular, that is the initial vertex is also the final vertex of the figure. To reflect this fact, the array starts with subscript 0 and ends at n-1 if there are n distinct vertices. We then call Vxf the vertex form of the figure and call the representation in Odf the intersection form. Coming back to Wedge, its Vxf becomes (ignoring Status for the time being),

```
Vxf[0] := (0,0) | HG(-1,0)
Vxf[1] := (p,p) | HG(1,0)
Vxf[2] := (-p,p) | CL(0,0,1)
```

The interpretation of Vxf has already been given at the beginning of of this section. This is just a symbolic way of rephrasing it.

4.6 Main algorithm.

The main algorithm is to convert an object from the intersection form to the vertex form, so that the object will become display-ready for those controllers that are capable of reading coordinates.

Giraud [4] introduces the php (Presque half-plane) representation of an object, which is quite similar our Intersection Form, but he offers no algorithm of conversion to the Vertex Form.

In fact, most algorithms differ from ours because they tend to deal with problems that are more or less different in nature from ours.

Bentley and Ottmann [2] address the problem of detecting whether any two objects in a planar set intersect. They offer algorithms that count the number of such intersections and algorithms that report all such intersections. However, the objects that they deal with have to satisfy three rather severe conditions: (1) Any vertical line through the object intersects the object exactly once. (2) For any pair of objects intersecting the same vertical line it is possible to determine algorithmically (at constant cost) which is above the other at that line. (3) Given two objects it is possible to determine algorithmically if they intersect, and if so to compute their leftmost intersection point after some fixed vertical line.

There are a few algorithms on intersecting convex polygons. Shamos [10] offers an algorithm that divides one polygon into angular sectors, and locates each vertex of the other polygon either inside or outside of these sectors. Hoey [11] designed a somewhat simpler algorithm. The plane is partitioned into parallel "slabs," with every polygon vertex lying on some slab edge. Within the slabs, the problem reduces to the intersection of two trapezoids. The resulting regions within the slabs can then be merged together in a single pass over all slabs to form the output polygon.

Both Shamos' and Hoey's are fundamentally different from ours. However, O'Rourke and others [7] present another algorithm of intersecting convex polygons that is similar to ours in spirit. The algorithm they propose maintains two special pointers, distinguishing one edge on each polygon. These pointers are advanced around the polygons such that their edges "chase" one another, searching for the

intersection points. All the intersection points can be found within two cycles around the polygons.

The major similarity is that both this algorithm and ours use some kind of "advance rule". The major difference is, of course, that we are addressing the problem of intersecting a primitive with a convex object, whose boundary is not limited to straight edges.

For our algorithm, suppose the window of viewing for the picture has lower left corner at (W1,W2) and upper right corner at (W3,W4), and also suppose an object, B, is the intersection of primitives P(i), where i = 1, ..., n, and P(i) is one of {HG, HL, VG, VL, CL}, must now construct Vxf for B.

1. Initialize Vxf to be the following, with each record having 1 for the Status value (Status value of 0 means vertex is to be discarded, 1 means to keep, and 2 means status unknown),

```
Vxf[0] := (W1,W2) | VG(W1)   | 1
Vxf[1] := (W3,W2) | HG(0,W2) | 1
Vxf[2] := (W3,W4) | VL(W3)   | 1
Vxf[3] := (W1,W4) | HL(0,W4) | 1
```

and set Numvertex, the number of vertices of Vxf, to be 4. This step ensures that every object is bounded by the window.

2. Decompose B and recursively intersect Vxf with each primitive of B, resulting in the vertex form of B.

Before we go deeper into the algorithm, it may be helpful if we first illustrate the algorithm with an example. Suppose existing is a convex figure of four straight sides with vertices at P, Q, R and S respectively, and we want to intersect it with a circle (see Figure 4).

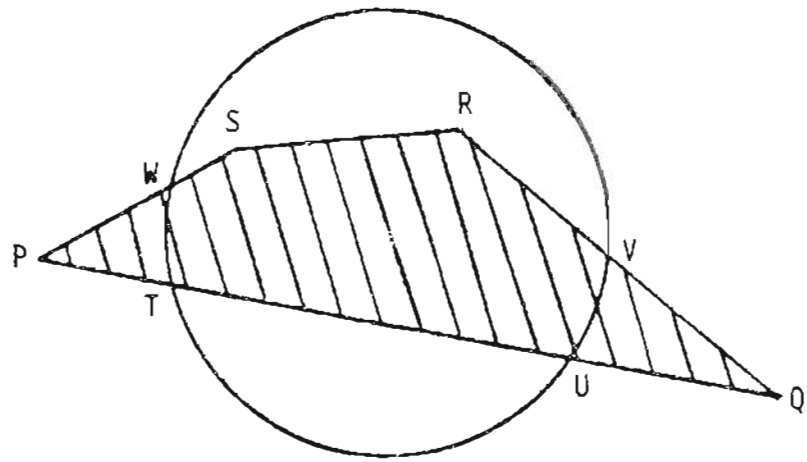


Figure 4.

The steps are as follows:

a. Assign P, Q, R and S the status value of 1.

b. Since P and Q are outside the circle, re-assign their status value to be 0.

c. Get new points T, U, V and W. Assign the status value of 2 to them. There are two things to note,

   1. Make sure the new points are on (and not outside) the edges of the existing figure.

   2. Since the vertices are ordered, we have to keep the new points in order as well. In our case, we have T before U, but not the other way around.

d. We have to decide on the paths between vertices. For instance, there are two paths from T to U, one is an arc and the other is a straight line. Which one to choose?

Consider the vertex form (Vxf) at this stage now. We have,

```
Vxf[0] := P | HL(m1,b1) | 0
Vxf[1] := T |     ??    | 2
Vxf[2] := U |     ??    | 2
Vxf[3] := Q | HG(m2,b2) | 0
Vxf[4] := V |     ??    | 2
Vxf[5] := R | HL(m3,b3) | 1
Vxf[6] := S | HL(m3,b4) | 1
Vxf[7] := W |     ??    | 2
```

Our task is to solve for the ??'s. Recall that Vxf is a circular representation, that is, each vertex has a predecessor as well as a successor. Now we go through each vertex of Vxf from Vxf[0] to Vxf[7]. If a vertex does not have status value 0, then move on to the next one. Otherwise, if its predecessor has status value 2, then the predecessor gets the Objdesc value of this vertex and we change the status value of the predecessor to 1. At the same time, examine the status value of the successor. If it is 2, then the successor will get the new intersecting object for its Objdesc value. In this case, it is the circle. Again, change the status value of the successor to 1. Move on to the next vertex.

Now, remove from Vxf all the vertices that have status value 0. Remove also every vertex whose coordinates are the same as those of its predecessor's.

Back to the example, going through the above steps, Vxf is then updated to be,

```
Vxf[0] := T |   circle   | 1
Vxf[1] := U | HG(m2,b2) | 1
Vxf[2] := V |   circle   | 1
Vxf[3] := R | HL(m3,b3) | 1
Vxf[4] := S | HL(m3,b4) | 1
Vxf[5] := W | HL(m1,b1) | 1
```

The result is shown below in Figure 5.

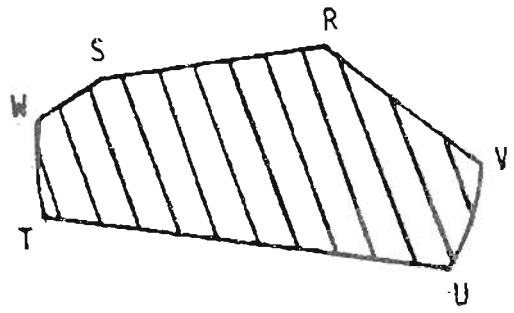


Figure 5.

The details of the algorithm are included in Appendix A.

4.7 Special algorithm.

The main algorithm is concerned with intersection of a convex object with a primitive which is either a half-plane or a disc. However, we still have to handle the case of intersecting a convex object with the primitive PL, polygon, and PL is not even convex. However, a polygon can be decomposed as a union of convex objects. Once this is done, then we can apply the main algorithm separately to each smaller convex object by virtue of the distributive law of intersection

over union in set theory. So, what we present here is then an algorithm for decomposing a polygon (satisfying the conditions as listed in 2.1) into smaller triangles. Hence, a convex object intersecting with a polygon will be turned into intersecting with a union of triangles.

The algorithm of triangulating a simple polygon as provided by Garey and others [3] is an often-cited one. It involves three rather complex steps--regularization, decomposition into monotone polygons and triangularization. The algorithm we want to present here, on the other hand, is a direct, simple to understand and easy to perform algorithm.

We now present our version of the algorithm of triangularization. Intuitively, we examine the first three vertices of the polygon and see whether the triangle that they form lies outside the polygon or contains any of the remaining vertices. If so, rotate the vertices so that the first one becomes last, and the second one becomes first; otherwise, save away this triangle and repeat the process after the second vertex has been removed from the polygon.

Formally, let V = {v(1), v(2), ..., v(n)} be a set of "legitimate" vertices of a polygon. Let V' be obtained by removing v(2) from V and renumbering the elements so that V' = {v(1), v(2), ..., v(n-1)}. Also let V" be formed by a circular rotation of V by one element so that V" = {v(2), v(3), ..., v(n), v(1)}. For convenience, denote v(1), v(2) and v(3) by A, B, C respectively. Then we have,

```
while n > 3 do
  if ∠ABC > pi then V := V"
  else if ∠ABC = pi then
```

```
        begin
        V := V'
        n := n-1
        end
      else
        begin
        if any v(i), i = 4 to n, is inside ABC then V := V"
        else
          begin
          store triangle ABC
          V := V'
          n := n-1
          end
        end
    store last triangle ABC.
```

5.0 Example:

The following picture, "King", was taken as a 35mm slide and reproduced later as a print. The left iris was created by intersecting two verticle half-planes with a circle, and then rotated slightly. The right iris is just a duplicate of it. The nose is again the intersection of two half-planes and a circle. The mouth is the intersection of three half-planes. However, the crown is the intersection of a non-convex polygon with a circle. That is why the two outer peaks are not exactly pointed like spires.

Figure 6.

6.0 Manual:

For convenience of reference, we include the manual separately in Appendix B.

7.0 Summary:

As stated before, this thesis was set out to demonstrate that the concept of a hierarchical graphics composition model is a workable model for creating, editing and managing two-dimensional color pictures, and in this regard, the mission was accomplished. It has been shown that this is quite a powerful model which is also very flexible. The concept itself is not difficult to grasp, and its implementation is quite

friendly and easy to use. However, this has yet to be confirmed by other users. Although quite limitedly, we have seen that text is also used as part of a picture. Still, the most significant contribution of this thesis is the algorithm of converting from the Intersection Form to the Vertex Form, which facilitates the implementation quite substantially.

Of course, there is ground for improvement over what we have done so far. One of the improvements can be to use more set algebra in defining and manipulating the different entities. For instance, an object can be defined as a union of intersections of primitives instead of just an intersection of primitives. For another example, the package will gain more power if more operations are allowed at the group level, such as taking the union of previously defined groups, relocating or duplicating a group.

Finally, in reviewing this thesis, Dr. Vegdahl pointed out that the four primitives HG, HL, VG and VL can be represented in one unified manner, that is, the defining straight line can be specified by two points A and B, then the half-plane is obtained by using an infinite radius centering at A and passing through B to do a 180-degree counter-clockwise sweep. This representation does seem more elegant because it frees one from having to deal with slopes. A possible disadvantage is that it may be a little confusing for the novice since entering B first and then A will result in a different half-plane.

8.0 References:

[1] AXIA Graphics Software Package User's Manual, Metheus Corp., Hillsboro, Oregon, 1982.

[2] Bentley, J. L. & T. A. Ottman, "Algorithms for Reporting and Counting Geometric Intersections," IEEE Trans. on Computing, vol. C-28, no. 9, September 1979, pp. 643 - 647.

[3] Garey, M. R., et al., "Triangulation of a Simple Polygon," Information Processing Letters, vol. 7, no. 4, 1978, pp. 175 - 179.

[4] Giraud C., "Presque half-planes: towards a general representation scheme," Computer-Aided Design, vol. 16, no. 1, January 1984, pp. 17 - 24.

[5] Henderson, P., "Functional Geometry," Conference Record of the 1982 ACM Symposium on LISP and Functional Programming, ACM, New York, 1982, pp. 179 - 187.

[6] Kernighan, B. W. & D. M. Ritchie, The C Programming Language, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[7] O'Rourke, J., et al., "A New Linear Algorithm for Intersecting Convex Polygons," Computer Graphics and Image Processing, vol. 19, no. 4, August 1982, pp. 384-391.

[8] Ousterhout J., "Editing VLSI Circuits with Caesar," Computer Science Division, Electrical Engineering and Computer Sciences, UC Berkeley, Calif., 1983.

[9] Preparata, F. P. & K. J. Supowit, "Testing a Simple Polygon for Monotonicity," Inform. Process. Lett., vol. 12, no. 4, 1981, pp. 161 - 164.

[10] Shamos, M. I., Computational Geometry, Ph. D. Dissertation, Yale University, 1978.

[11] Shamos, M. I. & D. Hoey, "Geometric Intersection Problems," Proc. of 7th Annual Symp. on Foundations of Computer Science, Houston, Texas, 1976, pp. 208 - 215.

[12] UNIX Programmer's Manual, 7th ed., Bell Telephone Laboratories, Inc., New Jersey, 1979.

Appendix A

A.0 Purpose:

The algorithm of converting Intersection Form into Vertex Form is detailed here. It is presented in the style of PASCAL to make the reading of the code a little easier if one desires to do so.

A.1 The algorithm:

```
procedure PEELOBJ;
    {This procedure translates object from the intersection form (as in
Odf) to the vertex form (Vxf) by decomposing the object.}
  procedure CUTALOT(Tempobj: Objdesc);
      {This procedure prepares to take the intersection of a primitive
  P(i) and Vxf.}
    function MARKVERTEX: integer;
        {This function is to check each vertex of Vxf against P(i).
      If the coordinates of a vertex is not inside P(i), then this
      vertex will be marked with Status value of 0. MARKVERTEX returns
      the total number of marked vertices.}
    function INSIDESHAPE(X,Y: real): boolean;
        {This function is to determine if a point (X,Y) is inside the
      convex figure represented by Vxf or not. (X,Y) is inside the
      figure if it satisfies all the inequalities in Vxf.}
    function NEWSHAPE(Code: Primitive; R1,R2,R3: real): integer;
        {This function performs the intersection of P(i) with Vxf.
      When finished, NEWSHAPE will return the number of vertices of the
```

new Vxf.}

procedure ADDVERTEX;

{This procedure calculates all the intersection points of P(i) with all the edges of Vxf, and then decides which of them are valid points, i.e., points that are on the edge of Vxf.}

Algorithm:

Let us establish some notation and agreement before we proceed further. If Vxf[j-1] and Vxf[j] are two consecutive vertices, then we define Edge(j) to be the part of the boundary that goes from Vxf[j-1] to Vxf[j]. In this case, it is either an arc or a segment. Recall that vertices are ordered. Also, we will not distinguish circles from discs nor straight line from half-planes in this discussion. It should be clear from the context which ones we are referring to.

For each j, j = 1, ..., Numvertex, P(i) intersects with Edge(j) at at most 2 points if either P(i) or Edge(j) is circular, and at most 1 point if both are straight. If there is no intersection points, go on with Edge(j+1). Otherwise, for each intersection point check if it is between the end points of Edge(j). If not, discard this intersection point. If all intersection points are discarded, go on with Edge(j+1). If two intersection points still remain, sort them by comparing angles, or distances, depending on whether Edge(j) is circular or straight, between the intersection points and the first end point of Edge(j). Insert these intersection points into Vxf and assign each the Status value of 2. These new

points will become vertices after they receive their respective Objdesc assignments, which can be determined only when all edges have been processed.

function UPDATEVXF: integer;

{This function makes the Objdesc assignments to points having Status value of 2, and deletes all points of Status value 0.}

Algorithm:

If k is an integer and k is divided by Numvertex, denote its remainder by k', i.e. k' = k mod Numvertex. Consider Vxf(j'), 1 <= j <= Numvertex. If the Status value is not 0, then go on to Vxf((j+1)'). Otherwise, if the Status value of Vxf(j-1) is 2, then assign the Objdesc value of Vxf(j') to Vxf(j-1), and set the Status value of Vxf(j-1) to 1. Also, if the Status value of Vxf((j+1)') is 2, then assign the Objdesc value of P(i) to Vxf((j+1)') and set its Status value to 1.

The next step is to remove all the points in Vxf whose Status values are 0. After that, for j = 1 to Numvertex - 1, remove the j-th point from Vxf if its coordinates equal those of the (j-1)-th point. Deduct 1 from Numvertex for each point removed.

```
  begin (*NEWSHAPE*)
  ADDVERTEX;
  NEWSHAPE:=UPDATEVXF;
  end;
begin (*CUTALOT*)
```

First call MARKVERTEX to determine the number of vertices of Vxf that lie outside P(i). If all vertices lie inside P(i), then Vxf is a subset of P(i). The result of intersecting them is just Vxf itself, and hence we go on with P(i+1). If all vertices lie outside P(i), then consider two cases:

1. P(i) is straight, then it is an empty intersection.

2. P(i) is circular. Let P(i) = CL(h,k,r). If (h,k) is outside Vxf, then it is an empty intersection. Otherwise, P(i) is a subset of Vxf, and we can redefine Vxf to be

```
Vxf[0] := (h-r/2,k) | CL(h,k,r) | 1
Vxf[1] := (h+r/2,k) | CL(h,k,r) | 1
Numvertex := 2
```

and go on with P(i+1).

If, however, MARKVERTEX shows that some but not all vertices lie outside P(i), then call NEWSHAPE to get the new Vxf. If NEWSHAPE does not return an empty Vxf, go on with P(i+1).

```
begin (*PEELOBJ*)
```

Recall that object B has primitives P(1), ..., P(n). For i = 1 to n, call subroutine CUTALOT with P(i) being an input.

APPENDIX B

B.0 Purpose:

The main purpose of this appendix is to provide the necessary information so that after reading it, one will be able to run the program himself.

B.1 Operations:

The hardware requirement is a keyboard terminal, a Metheus Omega 400 display controller and a Tektronix 690SR color monitor. The host is a VAX 11/780 running UNIX. The ready-to-run program is called "awlge", and it is stored in directory ~andyl/graphics together with the other auxilliary files, which are:

```
awlge.p
awlge.c
awlge.h
oldaxia
I77
F77.
```

However, due to some technical difficulty, only the controller with device address 4 can be used. If one wants to use a different device address, he will have to change the line "setloc(4)" in procedure PICTUREINIT of the source file awlge.p to "setloc(n)", where n is the new device address, and it has to be a constant. The following is a shell script for the necessary work to compile, link and run.

```
#! /bin/csh -f
pc -w -c awlge.p
pc -o awlge awlge.o awlgec.o -loldaxia -lI77 -lF77
awlge
```

We will explain how the program works by providing a recorded session of running the program. Note that all commands can be abbre-

viated to be as little as the first two letters, although they are all spelled out here. For instance, the command list can be abbreviated by li. In the following dialogue, underscored letters are supplied by the user, and everything between $ and the end of a line is regarded as a comment for explanation purpose.

In 2.2 we mentioned the objects "Diamond" and "Wedge", and in 5.3 we showed the picture "King". This is how they are done:

```
% awlge                  $start the program on UNIX
* AWLGE *                $program header
>object Diamond          $define object called "Diamond"
 o:hl -1 1               $
 o:hg 1 -1                                  } intersection of
 o:hg -1 -1                                 } 4 half-planes
 o:hl 1 1                $
 o:eo                    $end "Diamond" definition
>object Wedge            $define another object called "Wedge"
 o:cl 0 0 1              $a disc with center at (0,0) and radius 1
 o:hg 1 0
 o:hg -1 0
 o:eo                    $end "Wedge" definition
>object many             $define "many"
 o:pl 5                  $polygon with 5 ordered vertices
  :0 0                   $
  :1 0                                      }
  :2 1                                      } 5 vertical points
```

```
  :1 3
  :0 1.5                    $
 o:cl 0 0 2.5
 o:eo                       $end "many" definition
>object text                $define "text"
 o:tx "Picture Shapes"      $text string
 o:eo
>list                       $list the object names
 Objects:-
 Diamond
 Wedge
 many
 text
>object many                $recall "many" for edit
        10 pl          5
        11 pt  0.00  0.00
        12 pt  1.00  0.00
        13 pt  2.00  1.00
        14 pt  1.00  3.00
        15 pt  0.00  1.50
        16 cl  0.00  0.00  2.50
 delete: 16                 $delete item #16 (a disc) from "many"
 delete: 0                  $end delete
 o:cl 0 0 1.5               $a new disc
 o:eo
>delete many                $delete "many" from object list
```

```
>list
 Objects:-
 Diamond
 Wedge
>new shapes                  $create new picture called "shapes"
 p:window 0 0 100 100        $window coordinates (lower-left,upper-right)
 p:group different           $a group of "shapes", called "different"
   g:member square           $a member of "different", called "square"
     m:object Diamond        $"square" has "Diamond" as shape
     m:pivot 50 50           $offset to (50,50)
     m:scale 2.5             $scaled 2.5 times
     m:angle 45              $rotated by 45 degrees   applied to "Diamond"
     m:color 5               $assign color 5
     m:em                    $end "square" definition
   g:member header           $2nd member of "different", called "header"
     m:object text
     m:pivot 20 60
     m:scale 1.75
     m:color 4
     m:em                    $end "header" definition
   g:member pie              $3rd member of "different", called "pie"
     m:object Wedge
     m:pivot 50 20
     m:scale 3
     m:color 12
     m:em
```

```
  g:eg                    $end "different" definition
 p:group same             $another group of "shapes", called "same"
  g:member bigdiamond
   m:object Diamond
   m:piv 10 10
   m:scale 4
   m:color 6
   m:em
  g:member small
   m:object Diamond
   m:pivot 10 10
   m:scale 2
   m:color 9
   m:em
  g:list                  $list member names of "same"
   Members:-
   bigdiamond
   small
  g:eg
p:list                    $list group names of "shapes"
 Groups:-
 different
 same
 p:plot same different;   $display two groups, ";" is crucial
p:ep                      $end "shapes" definition (saved to files)
```

```
$ Next we show how picture "King" was created, and how it can be edited.
>old King                    $recall old picture "King" for edit
  Window coordinates: -25.00 -25.00 25.00 25.00
 p:list
   Groups:-
   happy
 p:ep
>list
 Objects:-
 isoc2
 retcircle
 circle
 isoc1
 poly
 text
>object isoc2
         1 hl  0.00  0.00
         2 hg -0.50 -1.00
         3 hg  0.50 -1.00
 delete: 0
 o:eo
>object retcircle
         5 cl  0.00  0.00  1.00
         6 vg -0.50
         7 vl  0.50
 delete: 0
```

```
 o:eo
>object poly
        15 pl          7
        16 pt  1.25  0.00
        17 pt  8.75  0.00
        18 pt 10.00 10.00
        19 pt  7.50  3.33
        20 pt  5.00 10.00
        21 pt  2.50  3.33
        22 pt  0.00 10.00
        23 cl  5.00  5.00  6.00
 delete: 0
 o:eo
>object text
        25 tx KING
        26 vl  0.00
        27 vg  1.00
 delete: 26
 delete: 27
 delete: 0
 o:eo
>save                        $save edited picture (very important)
>old King
  Window coordinates: -25.00 -25.00 25.00 25.00
 p:plot happy;
 p:ep
```

>exit $exit from the program, back to UNIX monitor

B.2 Summaries:

We now provide a summary of commands and a summary of color references. Each command can be identified by its first two letters.

1. Commands:

a. At the ">" level,

object object-name

:define an object with name object-name, which is kept in the object list.

list

:list all names in the object list.

delete object-name

:delete object-name from the object list.

new picture-name

:specify a new picture with picture-name to be created.

old picture-name

:retrieve an existing picture with picture-name for edit or display.

save

:save picture into disc files under filenames picture-name.obj, picture-name.grp.

exit

:get out of awlge and get back to the UNIX monitor.

b. At the "p:" level,

window X1 Y1 X2 Y2

:specify the viewing window for the picture. (X1,Y1) and (X2,Y2) are the coordinates for the lower-left and upper-right corners of the window respectively.

group group-name

:define a group with group-name, which will be kept in the group list.

list

:list all names in the group list.

delete group-name

:delete group-name from the group list.

plot group-name1 group-name2 ... group-nameN;

:display these groups on the display monitor. Don't forget the semicolon.

ep

:exit the "p:" level to return to the ">" level. Picture will automatically be saved.

c. At the "g:" level,

member member-name

:define a member with member-name.

list

:list all members on the member list for a particular group.

delete member-name

:delete member-name from the member list of a particular group.

eg

:exit the "g:" level to return to the "p:" level.

d. At the "m:" level,

object object-name

:use object-name from the object list for the shape of the member. Same object-name can be used by any member of any group.

pivot X Y

:X and Y are the offsets by which the object, called upon by the member, will be translated.

scale factor

:enlarge (or shrink) the object by this factor.

angle degree

:rotate the object in the counterclockwise direction through this angle.

color number

:assign to the object a color corresponding to the color number.

em

:exit the "m:" level to return to the "g:" level.

2. Colors:

With the system default color map, the correspondence between the color numbers and the colors themselves are as follows:

color number	color
0, 1	black (background)
2, 3	red
4, 5	green
6, 7	yellow
8, 9	blue
10, 11	pink
12, 13	light blue
14, 15	white

and more.

B.4 Cautions:

This project involves many different components, including three programming languages and several pieces of hardware. As a result, some peculiarities do exist; here are the known ones:

1. At the "p:" level, when one issues the ep command, the picture will be saved automatically. However, if this is done too many times without going back to the UNIX monitor, a UNIX system error will occur, due to too many files open. So, it is a good idea to save the picture each time when any object or the picture is changed.
2. At the "p:" level, sometimes there will be no response to the plot command. When that happens, try going into the "g:" level and come back out. If that does not help, reset the Metheus controller.
3. If someone exits Caesar abnormally, it will leave the hardware setting in a state that is not proper for our program. To correct it, use the following commands:

```
stty >/dev/ttyhe 9600 even odd -raw -nl -echo -lcase tabs -tandem cbreak
stty >/dev/ttyhe erase undef kill undef intr undef
```

BIOGRAPHICAL NOTE

The author was born in Hong Kong on February 15, 1950. He came to the United States in 1969, and is now a permanent resident.

He received his Bachelor of Science degree with cum laude in Mathematics from Whitworth College, Spokane, Washington in 1973. In 1974, he received his Master of Science degree in Mathematics from University of Oregon, where he continued his post-master's study in mathematics until 1976. He then attended Montana State University and received his Master of Science degree in Industrial and Management Engineering with a minor in Statistics.

Upon graduation from Montana State University, the author joined Tektronix, Inc. in July 1979 as a software engineer to design and implement some statistical and other software packages.

In June 1984, while still being employed by Tektronix, the author completed the requirements for the degree Master of Science at the Oregon Graduate Center.

The author has published the following articles:

"A Simple Series for the Incomplete Gamma Integral," Applied Statistics, JRSS, Series C, vol. 29, no. 1, 1980.

This article will be republished in a book of collected algorithms, edited by I. D. Hill et al.

"The Periodic Generating Sequence," The Fibonacci Quarterly, vol. 15, no. 2, April 1977.

Since 1973, the author has been married to the former Frances Wong, who is a professional librarian by training and is working for Blackwell North America as a sales manager.