GPL/I—A PL/I extension for computer graphics

by DAVID N. SMITH
Boeing Computer Services Company Inc.
Wichita, Kansas

INTRODUCTION

The tools available to the professional graphics applica-
tions programmer are without a doubt in the model T
stage. They are often implemented as subroutine
packages that either provide limited power or place a
large burden on the user of the package. In addition, in
an industrial environment the cost of training an
experienced programmer to become proficient with the
package can be very high.

The users of computers have long since become
accustomed to dealing with normal input-output in a
logical rather than a physical manner, often not
knowing or caring what mechanism is employed to
read a sequential file or where, at compile time, the
file is stored. High level languages have provided the
ability to have logically identical files on such diverse
units as tapes, disks, card readers and data cell drives.

A graphics terminal is just another input-output
device. Granted, it is a relatively new and still glamor­
ous device, but this should not prevent creation of
software to allow a programmer to use the device
rather than to fight cumbersome packages and produce
unreadable source code. Several applications oriented,
specialized or device dependent languages have been
described in the literature, but none are general.

In an attempt to find a better solution to the problem,
a design study was initiated by the author in 1968. This
paper will present the status of this study as of Septem­
ber, 1970. Discussed first will be general considera-
tions in the design of a high level graphics language,
followed by a short description of the PL/I based language.
PL/I was chosen as the base language, rather than
ALGOL or FORTRAN because of its built-in interrupt
handling structure, its richness of data types and
structuring, its built-in input-output (as compared to
ALGOL) and its “extendability” (as compared to
FORTRAN). A basic knowledge of PL/I is assumed
throughout.

GENERAL CONCEPTS

Since almost any graphics application of any signifi-
cant size will require non-graphical computing support,
the language should at least allow access to a general
procedural language or be an extension of such a
language. Some of the advantages to the “base”
language approach are:

A. Provide a syntactic base to guide in the design of
the graphics statements.
B. Cut implementation time because less needs to be
implemented and because some or most of the
implementation can be done in the host language.
C. Take advantage of extensions to and improvements
of the base language.
D. Avoid “re-inventing the wheel”. A GO TO state­
ment is a GO TO statement whether in ALGOL,
FORTRAN, JOVIAL or “GRAPHICSTRAN.”
(It might differ in flavor and texture but not in
meaning.)
E. Cut training time when the base language is one
known to the users.

The language should provide general digital graphics
facilities that are not limited to any machine or device
type. The user should not need to know the specific
characteristics of the hardware to write a program. A
display should be able to be described in terms that are
general to most (if not all) devices, with user knowledge
of a specific device limited to knowing which language
features are supported. Defaults should be supplied
when possible. Thus, if an image in storage contains the
information that it is to be displayed as a red image,
the image should still be displayable on a device that
does not support red images by simply ignoring the
option.

The above brings up an important point: That images
(or pictures) should be defined in terms of concepts
rather than specific hardware commands. A statement like:

LOWER RED PEN;

could be defined for a plotter and a statement like:

MAKE BEAM RED;

could be defined for a CRT device. However, each of these statements is conceptually applying the attribute "color" to an image that is to be displayed. A more general statement might be:

ATTRIBUTE OF (image) IS RED;

where "image" is a picture name. This does not limit the application of the attribute "color" to existing or supported devices but has applicability even to such exotic output media as oil paintings and holograms.

It is a temptation to provide other than basic facilities in a language. This should be avoided unless some overall gain results. As an example, it is far better to provide the ability to create labeled axes in an applications subroutine library written in the language than to build it into the language.

Implied in much of the previous discussion is the idea that the language should support any and all devices that are considered to be graphics devices. This might well include, for example, line printers if print plots are desired as output.

Image data structuring should be built in the language in such a way that the details of the structure are hidden from the user. Many examples of powerful but complex image structuring schemes are described in the literature. They have a common purpose, to provide dynamic data structuring, but suffer from being so complex to use that the average user will never apply them. In addition, direct use of a specific data structure prevents easy structure conversion as a program or its application grows. Thus, image data structures should be included in the language in such a way that the details of the structure are hidden from the user and so that the structural mechanism may be readily changed. More than one structure should be available to allow the user to select which type best fits his needs at the moment, and a "non-structure," which eliminates the need for expensive structural manipulations, should be available.

The "non-structure" or sequential image provides the ability to store pictures as a sequential list or stream of graphics data. The characteristics of devices such as plotters require that the programmer have control over the order of the output.

Two approaches to causing the display of images can be conceived. One is the "describe-it-then-output-it" method and the other is the "it-gets-output-while-I-describe-it" method, or automatic method. The first requires the user to output any and all images explicitly after they are created. The second provides automatic output as the image structure is being manipulated. Both should be available with the automatic method being the default.

With present graphics programming techniques, it is necessary to have the graphics device allocated to the program for all debug runs from the very start of testing. This can place a hardship on the programmer who needs to gain access to an overcrowded graphics terminal, not to mention the increase in costs due to the use of the device and to the occupation of other computer facilities. In most cases, initial debugging of a program or of a change to a program can be planned ahead of time and the session at the terminal is necessary only because it is the only testing method available. An alternate solution is to provide a method of making debug runs without an interactive terminal. This would require a software simulated device with the following characteristics:

A. A trace of input-output operations including the names of all pictures displayed, whether or not lightpen detects are allowed, which functions keys are enabled, etc.

B. An input stream containing simulated user responses to displays and actions to be taken to allow further processing when errors occur.

C. The debug package should react to all operations in the same manner as the device it replaces.

D. No program changes are required. The use of the simulated device should be set by control cards.

E. Simulation of a high cost device might be provided on a low cost terminal.

LANGUAGE DESCRIPTION

Vectors

Vector data and vector operations have been added to GPL/I to allow for ease in geometric calculations and to provide a method of describing language elements that contain coordinate information. Points are described by the components of a vector with its tail at the origin. Scaling information is given by a vector which describes the "distortion" along each of the coordinate axes. Rotation information is given by a vector which describes the "torque" perpendicular to the plane in which rotation occurs.

Vectors may be two-dimensional (VECTOR (2)), which is the default when the number of dimensions is omitted, or they may be three-dimensional (VECTOR (3)).
A vector is described in terms of its components along the coordinate axes. A component constant is a constant of any base, scale or precision followed by an X, Y or Z. All of the following are vector components:

\[ 14.7X \]
\[ 2Y \]
\[ 10010BX \]
\[ 0.214159E1Z \]

A vector may be formed by an expression of these components. Several examples are:

\[ \text{VEC1} = 43.5X + 2.2Z + 1.4X + 2.7Y; \]
\[ \text{VEC2} = 8X + 2Z; \]
\[ \text{VEC3} = 8 + 2Z; \]

The Y component of VEC2 is zero. VEC3 has the same value as VEC2 since a scalar constant is converted to an X component. Vectors may be defined by the VECTOR function and examined by the MAG, XMAG, YMAG and ZMAG functions.

Operators are provided for the vector and scalar products. The dot product of two vectors is:

\[ \text{VEC1} \cdot \text{VEC2} \]

and the cross product is:

\[ \text{VEC1} \times \times \text{VEC2} \]

Addition and subtraction are performed by the standard operators. In the hierarchy of operations, \( \times \times \) is highest and \( \cdot \cdot \) is next highest.

**Images and image expressions**

The “image” is the foundation of GPL/I. The language would not exist without it, and the bulk of the power of the language is dependent upon the manipulations that may be applied to it. Thus, a careful understanding of the concept and properties of the image is essential.

| POINTS | 7.2X + 3.3Y - 1.5Z |
| TEXT | "DEPRESS ANY FUNCTION KEY", 0.14 |
| FUNCTIONS | ARC(POINT1,POINT2,3.14159) |
| COPY(IN) | |
| INSTANCE(PIC) | |

<table>
<thead>
<tr>
<th>NAME</th>
<th>FORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inclusion</td>
<td>+&gt;</td>
</tr>
<tr>
<td>Connection</td>
<td>III</td>
</tr>
<tr>
<td>Positioning</td>
<td>ø</td>
</tr>
<tr>
<td>Scaling</td>
<td>&lt;&gt;</td>
</tr>
<tr>
<td>Rotation</td>
<td>++</td>
</tr>
</tbody>
</table>

Defining an image as “an identifier to which the IMAGE attribute has been applied”, is correct but provides little information. An image may be better defined as a variable which may take as values some combination of pictorial data, pictorial function values and other images. This is still incomplete but is a good starting definition. The best definition may be obtained by listing the properties of an image as described in this paper.

An image is a new data type which may have any storage class and may be structured. Its value is in two parts, pictorial values and attribute values. Both may be changed at execution. Dynamic attributes are discussed later. The storage for the value of an image is acquired dynamically and is structured as necessary to record relationships between images, their contents and attributes. Although the structuring mechanism may be very complex, it is totally hidden from the user.

Image data consists of points (represented as vectors), text, or values returned from image functions. See Table I for an illustration of each. Image data and other images, whether defined (given values) or not, may be combined to form a new image by the five image operators in a statement patterned after the arithmetic assignment statement. The operations are: inclusion, connection, positioning, scaling and rotation. Table II summarizes the operators.

The priority of graphics operators is lower than all other PL/I operators, as shown in Table III.

The inclusion operator causes the arguments to be included in a resultant image. Thus,

\[ A +> B \]

results in an image which contains A and B. The connection operator is similar to the inclusion operator except that a line is drawn from the last point of the first argument image to the first point of the second argument image. Assume that P4 and P5 are points,
TABLE III—Operator Priority

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>***</td>
<td>Highest</td>
</tr>
<tr>
<td>* .</td>
<td></td>
</tr>
<tr>
<td>** prefix + prefix -</td>
<td></td>
</tr>
<tr>
<td>* /</td>
<td></td>
</tr>
<tr>
<td>+ infix + infix =</td>
<td></td>
</tr>
<tr>
<td>&lt; &gt;</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>&lt;&gt; &lt;&gt; @</td>
<td></td>
</tr>
<tr>
<td>III ++</td>
<td>Lowest</td>
</tr>
</tbody>
</table>

then

P4 || P5

are the two points connected by a line.

The positioning operator specifies how an image is to be positioned. All images have a default coordinate system. The positioning operator specifies how the origin of this coordinate system is to be positioned in the coordinate system of the resultant image.

DEAD_MAN

= GUN @ P1 ++ SLUG @ P2 ++ MAN @ P3;

The origin of the image, GUN, is positioned at point P1 of a new image, SLUG is positioned and included and MAN is positioned and included and the result is assigned to the image DEAD_MAN.

The scaling operator specifies how the image is to be scaled. It is like the positioning operator in that the scaling applies to the resultant image.

LITTLE_BOX = BOX < > (1000.0X + 1000.0Y);

BIG_BOX = BOX < > (.01X + .01Y);

Scaling may be used to create concentric circles, assuming that CIRCLE contains a circle and that CONCIRC has no previous value.

DO I = 1 TO 5;

CONCIRC = (CONCIRC + CIRCLE)

< > (.9X + .9Y);

END;

The rotation operator is used like the scaling operator. It provides an amount in radians to rotate an image about the Z axis.*

DEL = DELTA * >> 3.14159Z;

Images may be defined in two or three dimensions.

Any legal non-complex arithmetic expression or vector expression may occur in an image expression in place of an image. The expression is evaluated and then converted to vector (if necessary). The vector is considered to be based at the origin. Its components then describe a point which is at the tip of the vector. The vector is thus converted to an image of this point.

Images, when used to define other images, are assigned by name rather than by value. This is one of the more powerful features of the language. Consider three images, PA, PB and PC, that contain the coordinates of the vertices of a triangle. The triangle is formed by the statement:

TRIANGLE = PA || PB || PC || PA;

The last side is drawn by connecting PC with PA. Since PA is an image and is assigned by NAME, this is exactly the same point as the first PA. Thus, the image TRIANGLE not only looks like a closed figure, it is a closed figure. Applications in several fields** either require or are simplified by an image which has the same structure as the thing it represents.

Since images are assigned by name, any change in the value of an image causes a change in any image which

TABLE IV—Examples of Image Assignment Statements

| 1) | A = B + C + D; |
| 2) | IX = (BB ++ CD) @ POS(J) ++ LNM @ XYZ; |
| 3) | EZ = (U @ X + C @ Y) << SC ++ ANG; |
| 4) | XX = A III B III 2.91 + 3.9Y III (0 + 0Y); |
| 5) | A = B III C III D; |
| 6) | DCL MSG CHAR(13), ERMSG IMAGE, ERPOS VECTOR; |
|     | MSG = "ERROR IN DATA" |
|     | ERMSG = "MSG" @ ERPOS; |
|     | /* IS EQUIVALENT TO */ |
|     | A = B III C III D III E; |
|     | DCL MSG CHAR(13), ERMSG IMAGE, ERPOS VECTOR; |
|     | MSG = "ERROR IN DATA" |
|     | ERMSG = "MSG" @ ERPOS; |
|     | /* IS EQUIVALENT TO */ |

* Three dimensional images may be rotated about any or all axes.
** Such as electronic circuit design.
contains it. An image may be defined in terms of other images which have no value at the time of assignment, thus allowing definition to be deferred. As an example, consider the concentric circle example given earlier. The nest of circles becomes a nest of squares by:

\[
\text{CIRCLE} = (OX + OY) \text{III} \{(IX + OY) \text{III} \{(IX + OY) \text{III} \{(OX + OY)}\}
\]

The image, CONCIRC, would now be changed as would a display of CONCIRC* on a terminal.

The COPY function may be used to assign images by

\[
A = XX; \\
B = \text{COPY} (A + R); \\
A = YY;
\]

The value of B is not altered by the redefinition of A in the third statement.

The effect of the scale, rotation and position operators as well as most image attributes** applies to all images in their scope. For example,

\[
\text{DCL A IMAGE REGION}(OX + OY, IX + IY), \\
B \text{ IMAGE REGION}(-IX -IY, OX + OY);
\]

The regions of A and B do not overlap. Thus,

\[
B = A;
\]

would cause B to be empty. (The reason is that everything outside the region of B is scissored or deleted and all of the region of A is outside the region of B.) Consider also:

\[
\text{DCL (AA, B, C) IMAGE SCALE} (2X + 2Y); \\
\text{AA} = (OX + OY) \text{III} \{(IX + OY); \\
B = AA; \\
C = B;
\]

AA is a one-inch line scaled by 2X + 2Y (which would be a \(\frac{1}{2}\) inch line when displayed). B is AA scaled by 2X + 2Y (which would be a \(\frac{1}{4}\) inch line when displayed). C is B scaled by 2X + 2Y which is AA scaled by 2X + 2Y (which would be a \(\frac{1}{6}\) inch line when displayed).

Image grouping is obtained by the INSTANCE function. Consider an image named IMX which is made up of a number of other images:

\[
\text{IMX} = A + >B + >C + >D;
\]

A lightpen detect on IMX will be correlated with the most basic image. Thus a detect on the part of IMX corresponding to A would cause the detect to be correlated with A (or a component of A) rather than IMX. However, if IMY is defined as:

\[
\text{IMY} = \text{INSTANCE (IMX)};
\]

then a lightpen detect on any part of IMY causes IMY to be correlated.

A number of useful image attributes are available. They are described in detail in an appendix but they include:

BLANKED  INTENSITY  THICKNESS
DASHED  PERSPECTIVE  VIEWPLANE
FLICKER  STYLE

The images described up to this point have been structured images. A second type of image, the stream image, may be operated upon like a structured image. It is always assigned by value (and thus must be defined before assignment) and lacks the structure which is so useful in lightpen correlations, graphics design, etc. A stream image does have several advantages; among them are savings in storage, savings in time to manipulate images and a direct correlation with sequential devices (plotters, for example). Unless noted otherwise, all images described are structured.

** INTERRUPT management **

Interrupts are manipulated by an extension of the standard PL/I interrupt management statements. An interrupt queue is provided for interrupts that are a result of a user's action at a terminal. Such interrupts raise the FUNCTIONKEY, KEYBOARD, LIGHTPEN, and DESIGN conditions. Interrupts that raise the IMAGEINTERRUPT, CHARACTERDISPLAY, and REGIONBOUNDARY conditions do so when the interrupt occurs.

Interrupts are queued as they occur in the queue of the task owning the file on which the interrupt occurs. However, if the condition is disabled, it is ignored, and if enabled for standard system action, the action is performed immediately. The interrupt queue is examined for processing when any of the following occur:

1. When the TAKE or WAIT statements are executed.
2. When the file is closed.
3. When a task is terminated.

The TAKE statement allows interrupts to be processed without entering a wait state. Interrupt conditions may be selected by name or by file, or all may be taken. The

* See the I/O section and the ACTIVE attribute.
** COLOR is a partial exception, for example.
TABLE V—Interrupt Processing Statement Examples

| 1) | TAKE FILE(X) LIGHTPEN FUNCTIONKEY; |
| 2) | TAKE LIGHTPEN; /* ALL FILES */ |
| 3) | TAKE FILE(X) NONE; /* CLEAR QUEUE FOR FILE X */ |
| 4) | TAKE FILE(X) ALL, FILE(X2) DESIGN; |
| 5) | ON LIGHTPEN(ADAGE) GO TO LPEN; |
| 6) | ON ENDKEY(TUBE) BEGIN; END; |
| 7) | ON LIGHTPEN(X) IDENT(A,B,C) |
|   | BEGIN; |
|   | GO TO LAB(IDENT); |
|   | LAB(1): |
|   | --- |
|   | LAB(2): |
|   | --- |
|   | LAB(3): |
|   | --- |
|   | END; |

Queue may be selectively or completely cleared. The statement:

```
TAKE FILE(TUBE) LIGHTPEN;
```

causes all LIGHTPEN interrupts on file TUBE to be processed. Additional examples are shown in Table V.

Eight graphics conditions are defined. The DESIGN condition is raised when a graphics device signals that a change to the displayed image is complete. The FUNCTIONKEY condition is raised in response to a function or selector key being depressed while the KEYBOARD condition is raised in response to the end of manual data entry. REGIONBOUNDARY is raised upon an attempt to output an image that is bigger than the area (REGION) in which it is contained. IMAGEINTERRUPT is raised by the display of an image created by the built-in function, IMINTR. The LIGHTPEN condition is raised as the result of the detect of an enabled image by a lightpen, and the CHARACTERDISPLAY and IMAGEDISPLAY conditions are raised in response to an attempt to display images on a file that does not support the given attributes.

Most graphics conditions allow the use of the IDENT option* to assist in the correlation of interrupts and the item causing the interrupt. Imagine a device (HAL 3361, for example) with four function selector buttons. The FUNCTIONKEY on statement may be:

```
ON FUNCTIONKEY(HAL3361) IDENT('1101'B) GO TO LAB(IDENT);
```

The bit string bits correspond to buttons 1, 2, 3 and 4 and specify that buttons 1, 2 and 4 are to be enabled for interrupts. (The bit string may be a bit string variable and will be referenced by name.) When an interrupt occurs and the on-unit is invoked, the function IDENT returns the button number. The IDENT option may be used to correlate lightpen detects:

```
ON LIGHTPEN(HAL3361) IDENT (A,B,C,D,E) ---;
```

The IDENT list contains image names that are to be enabled for interrupts. The IDENT function returns the sequence number of the image in the list.

File management and input/output

GPL/I defines a fourth file type, GRAPHIC,* and a set of attributes that describe the characteristics of the file. GRAPHIC files may be of three types: DESIGN, DISPLAY, and STORAGE. DESIGN files are defined as files on which an image is displayed, and from which it is possible to change some or all images. (Note that the images need not be changed but that it must be possible to do so.) Most C.R.T. devices fall in this category.

DISPLAY files are defined as files on which an image is displayed but on which no action at the device may directly cause a change in the image. Most plotters fall in this category. A device that can be associated with a DESIGN file may also be associated with a DISPLAY file to eliminate any possible overhead associated with DESIGN files.

STORAGE files are defined as files on which no image is displayed but on which images may be stored for later retrieval.

DISPLAY files may be declared as being HARDCOPY or SOFTCOPY. HARDCOPY files are files on which the output image is permanent; that is, the image cannot be modified once output. Images on SOFTCOPY files may be modified.

Files are opened and closed as in PL/I. Implicit opening is supported. Additional open options are also supported:

```
STORAGE, DISPLAY, DESIGN
ASSOCIATE (image-list)
PAGESIZE (vector-expression)
HARDCOPY, SOFTCOPY
UPDATE (option)
```

PAGESIZE specifies the size of the area (or volume) on

* KEYBOARD is the exception.

* STREAM, RECORD and TRANSIENT are the other three.
the device on which images are to be displayed. UPDATE specifies that the file may accept input/output from ACTIVE images. The other options are described elsewhere.

Output may occur either explicitly by execution of a PUT or ANIMATE statement, or implicitly by a change in an image that has the ACTIVE attribute. The ACTIVE attribute may be applied directly:

```
DECLARE BRACKET ACTIVE(X) IMAGE;
(BRACKET is active on file X), indirectly by the ASSOCIATE file option, and may be changed dynamically.
```

Any change made to an active image is output and any change made at a terminal is input with no further action on the part of the program being necessary.

Explicit output is controlled by the PUT statement:

```
PUT FILE(ABC) LIST(CAR, HOUSE,
(CLOUDS(I) DO 1=1 TO N));
```

EDIT directed output allows use of a format which specifies overriding attributes.

```
PUT FILE(XXX) EDIT (IM1,IM2)
(2 G(SCALE(V)));
```

The G (graphics) format item specifies the scale for the two images. Options are available to sound alarms at the terminal, lock and unlock keyboards, change the page on hardcopy files or clear softcopy files, and position the new page on hardcopy files.

The GET statement is used to retrieve images from storage and design files.

```
GET FILE(ABC) LIST(CAR, HOUSE);
```

The ERASE statement can cause selective or total erasure of the display on a softcopy file.

```
ERASE FILE (XYZ) LIST (CAR, CLOUDS(4));
```

**Example of image construction**

The example below creates an image of part of an electronic circuit. The components are generated by image functions. The function that generates capacitors is given as an illustration. The generated circuit is given in Figure 1. Each component is assumed to be one inch long. The X and Y axes are given for reference only.

```
DECLARE (NODE1,NODE2,NODE3) VECTOR,
CIRCUIT IMAGE;

DECLARE
(RESISTOR,CAPACITOR,VARINDUCT,
DIODE) RETURNS (IMAGE);

NODE1 = 2X + 2Y;
NODE2 = 3X + 2Y;
NODE3 = 4X + 2Y;

CIRCUIT = (RESISTOR => 180Z) @ NODE1 +
         (RESISTOR => 90Z) @ NODE1 +
         CAPACITOR @ NODE1 +
         (RESISTOR => 90Z) @ NODE2 +
         VARINDUCT @ NODE2 +
         DIODE @ NODE3 +
         NODE3 || || (4X + 0Y);

CAPACITOR: PROCEDURE RETURNS (IMAGE);

```

```
END CAPACITOR;
```

**Animation**

Images may be animated in two basic ways. The first is accomplished by outputting an image that
contains a reference to the IMINTR function. IMINTR
returns an image that causes the IMAGEINTERRUPT
condition to be raised when it is displayed. The on-unit
can then change the image and re-display it. The
amount of control available using IMINTR is greater
than with the second method but can require extensive
use of the C.P.U.

The second type of animation is controlled by the
ANIMATE statement which may be implemented
in part or all by hardware features or software in a
peripheral unit or display controller. As an example,
consider the following:

ANIMATE FILE (X) LIST (CAR)
(OFFSET (.2X+1.7Y)) EVENT (EV);

The image CAR (on file X) is to be moved a distance
of .2X+1.7Y inches every second. The process continues
until the event variable, EV, is set to completion, the
file is closed or the image is erased.
The list of animation options may contain any of the
following:

 ROTATE
 OFFSET
 SCALE

The argument gives the change per second. It is evalu­
ated only when the animate statement is executed.
However, the dynamic option allows the value of the
argument to be evaluated each tenth of a second.

ANIMATE LIST(A,B,C) (ROTATE(ANG))
FILE (X) DYNAMIC;

Dynamic attributes

Certain image attributes may be changed at execution
time. A dynamic change in an attribute is properly a
dynamic change in the current VALUE of the attribute.
An image always has some color; for example, the
current value of the color attribute may be RED at
certain times and BLUE at others.
Attributes are applied to images in one of two ways.
Functions may be used which return an image which is
the argument image with the attribute applied. Thus,

COLOR(IMAGEX,'YELLOW')
returns an image which is identical to "IMAGEX" but
with the color attribute applied with a value of
'YELLOW'.

The second method of altering attribute values is with
the attribute assignment statement. Attributes may be
assigned to variables declared with the ATTRIBUTE
attribute or to the ATTR pseudo variable. For example,

ATR = COLOR('BLUE') + FLICKER(X);

causes the color and flicker attributes to be assigned
to the attribute variable ATR. No other attribute
values are set.

ATTR(IM) = THICKNESS(I)
+ REGION(X1,X2);

The thickness and region attributes of the image IM
are changed by the above example.

Other attributes that can be applied to an image in a
manner like the above include:

SHIFT
ROTATE
INTENSITY
DASHED
BLANK
ACTIVE
PERSPECTIVE
STYLE

Functions

A full set of graphic, vector and attribute functions
are defined in GPL/I and are described in an appendix.
Several are of sufficient power to warrant further
mention.

The CANONICAL attribute may be applied to any
ENTRY identifier with the RETURNS(IMAGE)
attribute. It specifies that the function is NOT to be
invoked when encountered but that the entry name
and parameters are to be retained. The function is
invoked when the image is displayed. This prevents
functions which return large amounts of data from
wasting storage unnecessarily.

The ATTACHER function returns a special image
to which lines may be later connected with the CON­
CAT function. It is useful in network or circuit design.

REFERENCES

1 Graphic Subroutine Package (GSP)
IBM Form C27-6932
2 GPAK—Version II
IBM Program 360D-34.005 September 1966
3 G BRACCHI M SOMALVICO
An interactive software system for computer-aided design—An
application to circuit project
CACM September 1970 pp 537-545
APPENDIX A—FUTURE LANGUAGE ADDITIONS

Major Additions

1. New coordinate systems—polar, spherical, cylindrical
2. User defined REGION boundaries
3. Debug feature
4. Define graphics design in more detail
5. HALFTONE, SOLID and LIGHTSOURCE attributes for solid and graytone images.

Research Items

1. Constraints
2. Structure analysis

APPENDIX B—CHARACTER SET

The GPL/I character set is identical to that of PL/I except for the changes listed below.

1. The commercial ‘AT’ sign, @, is an operator rather than an alphabetic extender character.
2. The following operators are added to the language.

<table>
<thead>
<tr>
<th>Char 60</th>
<th>Char 48</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ &gt;</td>
<td>INC</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>@</td>
<td>AT</td>
</tr>
<tr>
<td>&lt; &gt;</td>
<td>SCL</td>
</tr>
<tr>
<td>* &gt;</td>
<td>ROT</td>
</tr>
</tbody>
</table>

3. The double quote (") character is added to the character set.

APPENDIX C—GRAPHIC OPERATORS

Operator | Description
---------|-------------------
Inclusion | The first and second operands are included in a new image.
+ >       | A + > B
Connection | Include the first and second operands in a new image and connect the last point of the first image with the first point of the second.
|||       | A ||| B
Positioning | Position the image (first operand) at the point given by the second operand.
@         | A @ (2.7X + 3.3Y)
Scaling   | Scale the image (first operand) by the factor given by the second operand.
< >       | A < > (4.0X + 8.0Y)
Rotation  | Rotate the image (first operand) by the amount given by the second operand. Rotation is counterclockwise.
* >       | A * > (.172Z)
### APPENDIX D—ATTRIBUTES

The attributes that may be altered by functions at execution are flagged with an asterisk.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Applies To</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*ACTIVE(---)</td>
<td>Image</td>
<td>ACTIVE specifies that the image is to be automatically output when changed and that the display is to be input when a change is made to it. A list of files on which the image is active may be appended. QUIESCENT is the opposite and is the default. I/O must be explicit.</td>
</tr>
<tr>
<td>QUIESCENT(---)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASSOCIATE(---)</td>
<td>File</td>
<td>ASSOCIATE specifies images that are to be active on a file. A parenthesized list of images is appended to ASSOCIATE.</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>Data</td>
<td>ATTRIBUTE specifies that the variable is to have attributes as its value.</td>
</tr>
<tr>
<td>*BLANKED</td>
<td>Image</td>
<td>BLANKED specifies that the image is invisible. UNBLANKED is the default.</td>
</tr>
<tr>
<td>UNBLANKED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CANONICAL</td>
<td>Entry</td>
<td>CANONICAL specifies that the entry is not to be invoked when encountered but that the entry and its parameters are to be retained as an intermediate result. The entry is invoked when the image containing it is to be displayed. CANONICAL implies REDUCIBLE and RETURNS(IMAGE).</td>
</tr>
<tr>
<td>*COLOR(---)</td>
<td>Entry</td>
<td>Color specifies the color of the image. The parenthesized item is an implementation defined color name in quotes. The default is COLOR(&quot;&quot;). If a device does not support the given color, the default is used.</td>
</tr>
<tr>
<td>DASHED(---)</td>
<td>Image</td>
<td>DASHED specifies that the image is to be displayed as a dashed line or curve. The parenthesized list is of the form: (up-length, ...), (down-length, ...) The curve is drawn one “up-length”. Then one “down-length”. The next item in each list is selected and the sequence is repeated. A list is restarted if exhausted.</td>
</tr>
<tr>
<td>DATAFORM(---)</td>
<td>Data</td>
<td>DATAFORM specifies that the data is to have the same internal form as a reference to the CANONICAL entry name in the parentheses. Assignment may be made to a DATAFORM variable or it may be referenced in an image assignment statement.</td>
</tr>
<tr>
<td>DEFINED(---)</td>
<td>Image</td>
<td>DEFINED specifies that the image is to be defined on the value of the base image. Its value is that part (or all) of the value of the base image that is contained in the region of the defined image.</td>
</tr>
<tr>
<td>DETECTABLE</td>
<td>Image</td>
<td>DETECTABLE specifies that the image can be detected upon by a lightpen. It is default for all images.</td>
</tr>
<tr>
<td>DESIGN</td>
<td>File</td>
<td>DESIGN specifies that the file is to be associated with a device that can display and store images in such a way that they can be both accessed by the program and changed by the user at the terminal. DISPLAY specifies that the device can display images but cannot store them and that the image cannot be changed by the user. STORAGE specifies that the device may store images but cannot display them.</td>
</tr>
<tr>
<td>DISPLAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*FLICKER(---)</td>
<td>Image</td>
<td>FLICKER specifies that the image is to be made alternately...</td>
</tr>
</tbody>
</table>
### Attribute

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Applies To</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPHIC</td>
<td>File</td>
<td>visible and invisible. On and off times are specified in seconds in a list similar to that provided with DASHED. GRAPHIC specifies that the file supports graphic input and/or output.</td>
</tr>
<tr>
<td>HARDCOPY</td>
<td>File</td>
<td>A HARDCOPY graphic file produces permanent copies and thus cannot be changed by program action once output. SOFTCOPY files may be changed by program action. SOFTCOPY is default.</td>
</tr>
<tr>
<td>SOFTCOPY</td>
<td></td>
<td>IMAGE specifies that the data is to have a picture (or pictures) as its value. The parenthesized list may be (2) for two dimensional images or (3) for three dimensional images. See text of paper.</td>
</tr>
<tr>
<td>IMAGE(-)</td>
<td>Data</td>
<td>*INTENSITY(n) specifies the relative intensity of the image. The option “n” is a number between between 0 and 9. LOCATION specifies the position of the image in a containing image or display.</td>
</tr>
<tr>
<td>*INTENSITY(n)</td>
<td>Image</td>
<td>INTENSITY specifies the relative intensity of the image. The option “n” is a number between between 0 and 9. LOCATION specifies the position of the image in a containing image or display.</td>
</tr>
<tr>
<td>*LOCATION(-)</td>
<td>Image</td>
<td>PAGESIZE specifies the size of the page on a graphic file. The vector “V” describes the page coordinates. On a CRT, the page is the area of the screen that will contain all displays, on a plotter the area to be plotted in until the page is next changed and as needed on other devices to describe the working area. PERSPECTIVE specifies that the 3-D image is to be displayed with a perspective view.</td>
</tr>
<tr>
<td>PAGESIZE(V)</td>
<td>File</td>
<td>PROTECTED specifies that the image will be protected from change by a user at a terminal when it is displayed. It is the default. UNPROTECTED specifies that the image may be changed by the user at a terminal. REGION specifies the area or volume in which the image is to have a displayable value. Any parts of the image outside the region boundary are scissored when the image is displayed. The form is: REGION(vector, vector)</td>
</tr>
<tr>
<td>*PERSPECTIVE(---)</td>
<td>Image</td>
<td>STREAM specifies that the image is to be stored as sequential data items. Stream images are always PROTECTED and UNDETECTABLE. See text of paper.</td>
</tr>
<tr>
<td>PROTECTED</td>
<td>Image</td>
<td>STYLE specifies the form of display of character data in images. THICKNESS specifies the relative thickness of the display of lines in the image. The argument “n” ranges from 0 to 9.</td>
</tr>
<tr>
<td>UNPROTECTED</td>
<td></td>
<td>UPDATE specifies the status of automatic I/O on a GRAPHIC DESIGN file. The options are (the default) IMPLICIT which allows automatic I/O and EXPLICIT which does not.</td>
</tr>
<tr>
<td>*REGION(---)</td>
<td>Image</td>
<td>VECTOR specifies that the data is to be a 2-D (n=3) or 3-D (n=3) vector. VIEWPLANE specifies the size and position of a plane from which views of a 3-D image are to be taken for 2-D display.</td>
</tr>
<tr>
<td>*ROTATE(---)</td>
<td>Image</td>
<td>STREAM specifies that the image is to be stored as sequential data items. Stream images are always PROTECTED and UNDETECTABLE. See text of paper.</td>
</tr>
<tr>
<td>*SCALE(---)</td>
<td>Image</td>
<td>STRUCTURE specifies that the image is to be structured. See text of paper.</td>
</tr>
<tr>
<td>*STYLE(---)</td>
<td>Image</td>
<td>STYLE specifies the form of display of character data in images. THICKNESS specifies the relative thickness of the display of lines in the image. The argument “n” ranges from 0 to 9.</td>
</tr>
<tr>
<td>*THICKNESS(n)</td>
<td>Image</td>
<td>UPDATE specifies the status of automatic I/O on a GRAPHIC DESIGN file. The options are (the default) IMPLICIT which allows automatic I/O and EXPLICIT which does not.</td>
</tr>
<tr>
<td>*UPDATE(</td>
<td>File</td>
<td>VECTOR specifies that the data is to be a 2-D (n=3) or 3-D (n=3) vector. VIEWPLANE specifies the size and position of a plane from which views of a 3-D image are to be taken for 2-D display.</td>
</tr>
<tr>
<td>VECTOR(n)</td>
<td>Data</td>
<td>STREAM specifies that the image is to be stored as sequential data items. Stream images are always PROTECTED and UNDETECTABLE. See text of paper.</td>
</tr>
<tr>
<td>*VIEWPLANE(---)</td>
<td>Image</td>
<td>STYLE specifies the form of display of character data in images. THICKNESS specifies the relative thickness of the display of lines in the image. The argument “n” ranges from 0 to 9.</td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
APPENDIX E—DESIGN

Design

The DESIGN condition is raised when the device signals that a change in the displayed image (except for keyboard entry) is complete. The FILE and IDENT options may be used. Standard system action is to update the structure in core if the image is active or else to ignore the condition. DESIGN is enabled by default and may be disabled by the NODESIGN label prefix.

Functionkey

The FUNCTIONKEY condition is raised by depressing a function key or selector button. The IDENT option gives the name of a bit string with a length equal to the number of function keys. A one bit indicates that the on-unit is to be executed for that key. If omitted, a bit string of all one bits is assumed. The file option is also allowed. FUNCTIONKEY is enabled by default and may be disabled by the NOFUNCTIONKEY label prefix. Standard system action is to ignore the condition. The bit string is referenced by name.

Example:

```plaintext
DECLARE ENABLED-KEYS BIT (32)
INITIAL ('11111111111111111111111111'B);
ON FUNCTIONKEY (BW2250) IDENT (ENABLED__KEYS)
BEGIN; --- END;
```

Keyboard

The KEYBOARD condition is raised whenever the end of data entry is signaled. The IDENT option may not be used. Standard system action is to update the image if the image is active or else to ignore the condition. KEYBOARD is enabled by default and may be disabled by the NOKEYBOARD label prefix.

Regionboundary

The REGIONBOUNDARY condition is raised when an image is being output to a file and a portion of the image exceeds the region boundaries. System action is to delete all parts of the image that exceed the region. The IDENT option is allowed. REGIONBOUNDARY is enabled by default and may not be disabled.

Imageinterrupt

The IMAGEINTERRUPT condition is raised by the display of an image that was created by the IMINTR function. The standard system action is to ignore the condition. The IDENT option may be applied. IMAGEINTERRUPT is enabled by default and may not be disabled.

Lightpen

The LIGHTPEN condition is raised by the detection of a DETECTABLE image by a lightpen. The IDENT option may be applied. LIGHTPEN is enabled by default and may be disabled by the NOLIGHTPEN label prefix. The standard system action is to ignore the condition.
**Characterdisplay**

The CHARACTERDISPLAY condition is raised when an unprotected image containing text data is to be output to a file that cannot support unprotected text data with the given display characteristics. A standard return from the on-unit causes defaults to be applied in an attempt to display the data. If no on-unit is active, the ERROR condition is raised. The IDENT option may be applied. CHARACTERDISPLAY is enabled by default and may not be disabled.

**Imagedisplay**

The IMAGEDISPLAY condition is raised when an image is to be output to a device which cannot support certain attributes of the image. A standard return causes an attempt to display the image by changing certain attributes. If this fails, the image is ignored. Standard system action is to raise the error condition. The CHARACTERDISPLAY condition is raised for text items. The IDENT option may be applied. IMAGEDISPLAY is enabled by default and may not be disabled.

**APPENDIX F—FUNCTIONS**

**Vector manipulation built-in functions**

- `XPROD(V1, V2)` Vector cross product
- `DPROD(V1, V2)` Vector dot product
- `ANGLE(V1, V2)` Angle between vectors
- `MAG(V)` Vector magnitude
- `VECTOR(S1, S2[, S3])` Convert scalars to vector
- `XMAG(V)` Component magnitudes

**Attribute built-in functions (See attributes in Appendix D)**

- `SCALE` PERSPECTIVE
- `REGION` ACTIVE
- `COLOR` FLICKER
- `BLANKED` INTENSITY
- `STYLE` DASHED
- `SHIFT` THICKNESS
- `ROTATE` PAGESIZE
- `VIEWPLANE` UPDATE

**Condition built-in functions**

- `IDENT` Return number of item in on-unit IDENT option responsible for interrupt.
- `LPDETECT` Returns the image on which the lightpen detect occurred.

**Image built-in functions**

- `INTERSECTION` Returns image that is the intersection of the two argument (3-D) images.
- `VIEW` Return image that is the view of the argument (3-D) image from its viewplane.
- `TEXT` Build character images.
- `ARC` Build arcs and circles.

From the collection of the Computer History Museum (www.computerhistory.org)
ATTACHER
NULLI
CONCAT
SWEEP
IMINTR
INSTANCE
LPTRACK

Returns special image. See text.
Returns a "NULL" image.
Connects images at attachers.
Returns an image (3-D) that is the result of sweeping another image (2-D) along a given path.
Returns special image which causes an interrupt when it is displayed.
Grouping function. See text.
Returns the tracking symbol for graphics design.

Special graphic built-in functions
INTERSECT
CHARLEN

Returns '1' if the two argument images intersect.
Returns the total displayable length of a character string given its style and height.

Image structure built-in functions
COPY (IM)
IMAGEN0(A,B)
INIMAGE(IM,N)
SUBELEM(IM,N)
EXPAND(IM)

Copies the structure of IM (remove references by name).
Returns the sequence number of image B in image A.
Returns the Nth image that contains the image IM.
Returns the Nth image that is contained in the image IM.
Expands all canonical function references in IM.

APPENDIX G—STATEMENTS

Attribute assignment statement
Syntax:

\{attribute-data-element\} [ .. ] = attribute-expression;
\{ ATTR(image) \}

Where "attribute-expression" is:

\{ attribute-data-element\} \[ \{ attribute-expression\} \[ (+)attribute-expression \]
\( attribute-expression \) \[ \{ attribute-expression\} \[ (-)attribute-expression \]

The attribute expression is evaluated left to right. For addition (+ operator), attributes are collected to form a complete set. As additional attributes of the same type are encountered, the new attribute value replaces the existing value. For subtraction, attributes are removed from the set. On assignment to an attribute-data-element all unspecified attributes remain unspecified. On assignment to an image by the ATTR pseudo variable, all specified attributes replace those of the image and the others are unchanged.

Example:

DCL (ALPHA,BETA) IMAGE;

\ldots

ATTR(ALPHA) = ATTR(BETA) - REGION(BETA) + COLOR('RED');

Logical expressions

Images and attributes may be compared to images and attributes, respectively, with the = and ≠ operators. The value and type of attributes is compared but image comparison is of identity only.
Image assignment statement

Syntax:
image [, . . . ] = image-expression;

where "image-expression" is:
Vector-Expression
| Image |
| image-expression | + > | image-expression |
| image-expression | | | | |
| image-expression | * > | vector-expression |
| (image-expression) | < > |

"vector-expression" is any legal vector or scalar arithmetic term or expression.

"text-item" is {"character-string",size"
| "character-string-identifier",size"

See text of paper for details of use.

On statement

Only the modifications to the ON statement are specified in detail:
Syntax:
ON graphics-on-condition(filename) [IDENT(option)] on-unit;

where:
"option" is
| bit-string |
| image-name [,image-name, . . . ] |

The use of the IDENT option is described in the text of the paper and with the on-condition descriptions.

Take statement

Syntax:
TAKE [FILE (filename [, . . . ])] {ALL
| NONE |
| on-condition-name[, . . . ] [ . . . ];

Interrupt(s) in the queue are processed when the TAKE statement is executed.
ALL Process all interrupts.
NONE Clear the queue.
"on-condition-name" Process interrupts for the given condition.
The FILE option limits the selection to interrupts from the given file.

Erase statement

Syntax:
ERASE FILE (filename [, . . . ]) [(list)];
where

‘list’ is an image list that has the same syntax as a list in a PUT LIST statement.

ERASE will selectively clear images from a display. If the list is missing, the entire file is erased.

**Animate statement**

Syntax:

```
ANIMATE FILE (filename) LIST (image-list) (parameters)
    EVENT (event-variable) DYNAMIC;
```

The images in “image-list” are displayed on FILE (filename) and animated as specified by the “parameters”. The animation of an image continues until the image is nullified, the image display is erased, the “event-variable” is set to completion or the file is closed. The allowable “parameters” are:

- ROTATE (change-in-angle)
- POSITION (change-in-position)
- SCALE (change-in-scale)
- THICKNESS (change-in-thickness)
- PERSPECTIVE (change-in-perspective)
- VIEWPLANE (change-in-viewplane)

The arguments to the parameters may be variables or expressions whose value changes each \( \frac{1}{15} \) of a second when the DYNAMIC option is specified and the implementation allows dynamic animation parameters for that device. Otherwise the initial value is always used.

ANIMATE may be used on GRAPHIC DESIGN files only.

**Signal statement**

The SIGNAL statement is extended for GPL/I. Only the extension is described in detail.

Syntax:

```
SIGNAL on-condition [SET (set-values)];
```

where:

“set-values” are references to built-in condition functions.

The SET option defines values to be returned by condition functions that are invoked to determine the reason that the condition was raised. The “set-values” are function names with the values that they are to return in parentheses.

Example:

```
SIGNAL LIGHTPEN(FILEX) SET (IDENT(1));
```

**Open statement**

Additional options are allowed for the open statement:

- DESIGN
- ASSOCIATE
- HARDCOPY
- STORAGE
- PAGESIZE
- SOFTCOPY
- DISPLAY
**Put statement**

Only the extensions for graphic files are shown.

Syntax:

```
PUT FILE (filename) [data] [options];
```

where:

```
{LIST (image-list)  
EDIT (image-list) (edit-format) }
```

'`data'` is

```
{EDIT (image-list) (edit-format) }
```

“options” are

```
SIGNAL sound an alarm at the terminal at the start of output.
{LOCK } lock or unlock the terminal keyboard before output starts.
{UNLOCK }
```

```
PAGE change the page on the device. This clears the CRT screen prior to output or
changes the physical page on HARDCOPY devices.
```

```
POSITION(--) Position the pen, scribe, etc., to the given point before changing the page. POSI­
TION applies to HARDCOPY files only and may be used only with the PAGE
option. It overrides automatic page repositioning implied by the files PAGE-
SIZE attribute.
```

“edit-format” is a format made up of the existing R format item and/or a G format item with the form: “n G
(attributes)”.

The attributes override those of the output image.

**APPENDIX H—EXAMPLE PROGRAM**

The program displays a circle and its center point. When the circle is detected upon it gets larger while detection
upon the center point causes the circle to shrink. The character string, ‘END-PROGRAM’ is displayed and initiates
program termination when detected upon.

```
/* SAMPLE GRAPHICS LANGUAGE PROGRAM */
SAMPLE: PROCEDURE OPTIONS(MAIN);
```

```
S1: DECLARE
IBM2250 GRAPHIC FILE,
(CENTER,CIRCLE,MENU) ACTIVE(IBM2250) IMAGE,
RADIUS FLOAT BIN(21) INIT(2.0),
LP(3) LABEL;
S2: CENTER = (5.0X+5.0Y);
S3: CIRCLE = ARC((5.0X+5.0Y), VECTOR(RADIUS+5.0, 5.0),360.0);
S4: MENU = 'END PROGRAM',.2" @ (.1X+.1Y);
S5: PUT FILE(IBM2250) SIGNAL;
S6: ON LIGHTPEN IDENT(CIRCLE,CENTER,MENU) BEGIN;
S7: GO TO LP(IDENT);
LP(1): /* DETECT ON CIRCLE */
RADIUS = MAX(RADIUS+0.25, 6.0);
CIRCLE = ARC((5X+5Y), VECTOR(RADIUS+5.0, 5.0), 360Z);
GO TO END;
```

From the collection of the Computer History Museum (www.computerhistory.org)
LP(2): /* DETECT ON CENTER */
RADIUS = MIN(RADIUS-0.25, 0.5);
CIRCLE = ARC((5.0X, 5.0Y), VECTOR(RADIUS+5.0, 5.0), 360Z);
GO TO END;

LP(3): /* END PROGRAM */
ERASE FILE(BW2250);
COMPLETION(ENDPROGRAM) = '1'B;
END: END;

S8: WAIT(ENDPROGRAM);
END SAMPLE;

Description of example program

<table>
<thead>
<tr>
<th>Label</th>
<th>Discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>IBM2250 is declared as a file. The device is an IBM 2250 or its functional equivalent. The images: CENTER, CIRCLE and MENU are declared as ACTIVE on the file IBM2250.</td>
</tr>
<tr>
<td>S2</td>
<td>The center of the circle is created as a point at 5.0X and 5.0Y and it is output.</td>
</tr>
<tr>
<td>S3</td>
<td>The circle is defined as a 360 degree arc with a 2 inch radius and it is output to the file IBM2250.</td>
</tr>
<tr>
<td>S4</td>
<td>The image MENU is given the value of the character string 'END PROGRAM'. It is .2&quot; high and its lower left corner is at .1X+.1Y.</td>
</tr>
<tr>
<td>S5</td>
<td>The console alarm is sounded to indicate the start of the program to the user.</td>
</tr>
<tr>
<td>S6</td>
<td>The actions that are to be taken when a lightpen detect occurs are defined. The IDENT option limits the invocation of the ON unit to the three images named.</td>
</tr>
<tr>
<td>S7</td>
<td>The IDENT function returns a 1, 2 or 3 corresponding to the image in the IDENT option (S6 above) that was detected upon. A branch is than made to one of the labels LP(1), LP(2) or LP(3).</td>
</tr>
<tr>
<td>LP(1)</td>
<td>The radius of the circle is increased by 0.25 inches. Then the image, CIRCLE, is redefined and output to replace the previous displayed circle.</td>
</tr>
<tr>
<td>LP(2)</td>
<td>The same action as above is taken except that the radius is decreased by 0.25.</td>
</tr>
<tr>
<td>LP(3)</td>
<td>The display is cleared and the event variable, ENDPROGRAM, is set to completion to end the program.</td>
</tr>
<tr>
<td>S8</td>
<td>Wait on the end of the program.</td>
</tr>
</tbody>
</table>