

The GSYM System  
An Interactive System For  
Displaying Linear Graphs

by

Robert Brien Maguire

Technical Report CS-73-24

**Faculty of Mathematics**



**University of Waterloo**  
**Waterloo, Ontario**  
**Canada**

The GSYM System  
An Interactive System For  
Displaying Linear Graphs

by

Robert Brien Maguire

Technical Report CS-73-24

## The GSYM System

The purpose of this report is to briefly introduce the GSYM interactive graphics system. GSYM, short for Graph SYMmetry, is designed to allow a user to create, manipulate and display graphs using an IBM 2250 Graphic Display Unit. The report is not intended to be a user manual for the GSYM system. Rather, it will examine several of the features in GSYM in order to see how the system was designed as a tool for investigating the problems involved in generating visual representations of graphs. The latter part of the report contains a discussion of some screen layout problems encountered while designing the graph display format used in GSYM. Appendix B contains a more detailed description of the design and implementation of GSYM, while Appendix A describes the hardware configuration and display capabilities of the IBM 2250 Graphic Display Unit. The discussions found in the report are not limited to the IBM 2250 Display Unit, but could be applied to any graphic display.

GSYM is not to be confused with existing graph processing languages as it is not a programming language per se even though it does make use of macro-like commands for operating on graphs in certain situations. It also contains a list processing subsystem, but is not primarily intended as a list processing tool. It is simply an interactive graphics system especially designed for working on the problem of generating displays of graphs.

The normal operation of GSYM is based on an 'interaction by anticipation' format. The system displays the possible operations

the user may initiate at any given moment and the user selects one of the current options using the light-pen attached to the display. The system is said to be in the 'reset' state when the current display shows the option list illustrated in Figure 1. All system operations begin with the user selecting one of the options in this list with the light-pen. For example, if he chooses the ADD option then the display shown in Figure 2 is placed on the screen. The user then indicates the element to be added to the graph. If this were to be a vertex addition the system would then request the user to position the vertex on the screen.

A similar communication process is used for all the system operations. That is, GSYM displays a list or menu of the options available to the user at the moment and awaits the user's response. This type of format not only guides the user but also eliminates the possibility of system error owing to invalid user input. In order to reduce the frequency of user error most of the option lists contain a brief instructive note defining the nature of the current operation and the user action required. Moreover, should the user change his mind or wish to cancel an operation in the middle of a command sequence he may return to the reset state by pushing one of the programmed function keys on the function keyboard attached to the display. Thus, the system is deliberately intended to be very forgiving, making it feasible for an inexperienced user to become proficient in its use through actual 'hands-on' experimentation.

USE LIGHT-PEN TO SELECT OPTION:

- \* ADD
- \* ALTER
- \* BACKUP
- \* DECLARATION
- \* DELETE
- \* DISPLAY
- \* ENQUIRE
- \* HARD COPY
- \*\*\* HALT \*\*\*
- \* MISCELLANEOUS FUNCTIONS
- \* MODE
- \* MOVE
- \* POINT
- \* ROTATION
- \* TRANSLATION

FIGURE 1 : RESET STATE DISPLAY

USE LIGHT-PEN TO SELECT ELEMENT TO ADD:

\* VERTEX

\* EDGE

\* DIRECTION

\* DIRECTED EDGE

\* LABEL

\*\*\* PFK 30 TO RESET \*\*\*

FIGURE 2 : ADD OPTION DISPLAY

As indicated in Figure 1 the user has a wide choice of operations available to him. The nature of most of these is evident from the name of the command. The basic graph manipulation options are addition, deletion and alteration of graph entities. The graphs are composed of vertices, edges and arrows. An arrow is always associated with an edge, that is, an undirected edge becomes a directed edge when the user adds an arrow (direction) to the edge and vice-versa.

The move, rotation and translation commands allow the user to manipulate the form of the graph as currently displayed on the screen. He is able to move vertices, edges and arrows about the screen or, if he so chooses, he may relocate the whole graph by using the translate command. With the rotation command the graph may be rotated about any point in the 3-dimensional cube in which it is defined. This 3-dimensional cube corresponds roughly to the box-like housing around the 2250 display screen.

The DECLARATION option is used to declare vertex and edge properties. Each vertex and edge in the graph may have associated with it its own property list. The length of the property list depends on the number of properties that have been declared for vertices and for edges. For example, if four vertex properties and five edge properties have been declared then every vertex has an associated property list containing four entries and every edge has an associated property list containing five entries. These property lists are very useful for associating numerical data such as edge weights with individual vertices and edges. This feature is designed for the use of graph-theoretic routines

implemented on the GSYM system.

However, the vertex and edge property lists serve a dual function since they are also used by the list processing subsystem in GSYM to create lists of vertices and edges in graphs. The user declares (through the declaration option) a 'head of list' pointer which points to the first vertex or edge in a list. At the same time he associates one of the previously declared vertex properties and one of the previously declared edge properties with the pointer.

The list is then created using the entries in the property lists of the individual vertices and edges in the list to link to the next element in the list. That is, the head of list pointer initially points to the first vertex or edge in the list. Vertex or edge elements are then added to the list using the POINT option. When an element is added to a list the property in the last element of the current list associated with the head of list pointer is set to point to the element being added to the list. The property entry in the most recent list element (associated with the head of list pointer) is always set to 0 to indicate the last element in the list.

The mode command sets the operational mode for the system. GSYM may function in one of two modes, normal mode or macro mode. Normal mode operation is simply the 'interaction by anticipation' format we have been describing. The user is aided in completing the current operation by a series of displays and system messages. In the macro mode, however, the user must type in a 'macro' which



contains all the information required by the system to perform the next operation. From an execution time standpoint the macro mode is more efficient, but it requires more user time and a more knowledgeable user.

Since GSYM is an interactive system it must be possible to quiz the system at any time concerning the current status of the graph, its properties and any graph related entities such as list pointers. This is the purpose of the ENQUIRE option. Figure 3 shows the option list from which the user may select elements of the graph or system whose status is to be displayed. At his option the user may also have this information printed out on the line printer.

The remaining options shown in Figure 1 are ones which are necessary because of the avowed intention of using GSYM as a tool for implementing routines for generating visual representations of graphs. The first such option is the BACKUP feature. Figure 4 shows the display generated by GSYM when this option is selected. The user is able to SAVE graphs on a peripheral device such as a disk or data cell and RESTORE them as he pleases. Both the graph and the current status of the system are stored so that when the graph is restored the same display is recreated as well as all graph properties and system and graph lists. Such a feature is necessary if a user is to be able to easily save displays created by visual representation routines and also be able to recover them at any time.

USE LIGHT-PEN TO REQUEST INFORMATION

\* VERTEX PROPERTY LIST

\* EDGE PROPERTY LIST

\* POINTER LIST

\* VERTEX PROPERTIES

\* EDGE PROPERTIES

\* LIST

\* VERTEX LIST

\* EDGE LIST

\*\*\* PFK 30 TO RESET \*\*\*

ANY OTHER BUTTON TO REPEAT ENQUIRY

FIGURE 3 : SYSTEM STATUS OPTIONS

USE LIGHT-PEN TO SELECT BACKUP OPTION

\* CREATE FILE DIRECTORY

\* LOAD FILE DIRECTORY

\* DELETE FILE DIRECTORY

\* SAVE

\* RESTORE

\* RENAME GRAPH

\*\*\* ANY PFK BUTTON TO RESET \*\*\*

FIGURE 4 : DISK/DATA CELL BACKUP

The graphs are saved as members of a file directory which the user must create before saving any graphs. The user types in a name for the file directory when he creates it. There is no limit on the number of file directories he may have. In order to save or restore a graph in a different file directory from the last directory used it is necessary to LOAD the appropriate directory. When a file directory is no longer required the file may be deleted from backup storage by using the DELETE FILE DIRECTORY option. The user must type in a name for every graph he saves or restores. It is possible to replace a graph in a directory with a newer version by doing a save and using the same name as was used for the old version. It is also possible to change the name of an existing graph in a directory by using the RENAME GRAPH option and specifying the old and new names. Notice that the user does not become involved in the actual detailed creation and maintenance of the files used to hold these graphs. In particular, he does not have to worry about the proper command language statements to create new files or reference existing ones.

We also wish to be able to produce copies of graph representations using a plotter. The HARD COPY option provides this facility. When the user selects this option GSYM produces a hard copy of the display of the current graph using a plotter. The plot of the graph cannot be produced directly because the plotter is not on-line in the existing hardware configuration. Instead, a punched card deck is created containing all the information needed by a special program written to recreate the display off-line using the plotter.

We have stated that GSYM is a tool for designing and testing schemes for the visual representation of graphs. But we have yet to indicate how the writer of such a representation routine attaches his program to the GSYM system. The answer is that he uses the MISCELLANEOUS FUNCTIONS option. When this option is selected GSYM replies with the display shown in Figure 5. The user responds by selecting one of the routines. SYS1 to SYS9 are special system routines, the nature and use of which are described in Appendix B. USER1 to USER6, however, are reserved names to be used by user written routines. When a name is selected the corresponding user routine is invoked by the system. Of course, this assumes that the corresponding routine exists and has been incorporated into the system. The actual details for establishing communication between the system and the user routine are described in Appendix B. When the user routine is finished execution control returns to the system. While this feature is intended for use with visual representation routines, a user who is familiar with the internal operations and data structures of the GSYM system may incorporate any type of (graph-theoretic) routine he wishes. For example, he might write a program for finding Hamiltonian paths which, if successful, uses the GSYM list processing subsystem to create a list representing the path and then returns a GSYM pointer to the start of the path.

In the process of designing GSYM it was necessary to solve several problems related to the choice of a display screen layout suitable for graphs. The solution of these problems would have

USE LIGHT-PEN TO SELECT FUNCTION:

\* SYS1

\* SYS2

\* SYS3

\* SYS4

\* SYS5

\* SYS6

\* SYS7

\* SYS8

\* SYS9

\* USER1

\* USER2

\* USER3

\* USER4

\* USER5

\* USER6

\*\*\* PFK 30 TO RESET \*\*\*

FIGURE 5 : SPECIAL SYSTEM/USER ROUTINES

a direct bearing on the form of any display produced by a visual representation routine. For this reason, we shall examine some of these problems and the steps taken to solve or avoid them.

As one might expect, vertices and edges are displayed as points and straight lines, their most natural representation. We have already mentioned the use of arrows to represent directions on directed edges. The user is free to position an arrow anywhere on the screen although one would hope it would be placed on the corresponding edge. In order to indicate the positive and negative ends of the edge the user specifies an arrow orientation. There are eight possible orientations corresponding to the eight major points on a compass, North, South, East, West, Northeast, Southeast, Northwest and Southwest. Figure 6 shows an example of a digraph with the directed edges illustrating the possible arrow orientations. Vertices and edges may also be labelled in GSYM. Labels contain one to four alphanumeric characters and may be positioned anywhere on the screen. Figure 6 also shows several examples of labelled vertices and edges.

The display screen of the IBM 2250 is a two-dimensional entity and is therefore only capable of two-dimensional pictures. If we were restricting our investigation to planar graphs and intended to require that all visual representations of these graphs also had to be planar maps, then such a two-dimensional display system would be quite adequate. However, we wish to generate visual representations for both planar and non-planar graphs. While this does not preclude the possibility of generating visual representations for non-planar graphs with the resultant intersecting edges, there is another more serious problem





in using a two-dimensional system. If a graph had multiple edges connecting the same two vertices then these edges would be displayed as the same line and it would be impossible to tell by looking at the screen which were the multiple edges. Moreover, we do not wish to restrict ourselves to a two-dimensional co-ordinate system because of the constraints this would impose on our visual representation routines. We would not be able to generate a representation treating a graph as a three-dimensional object and this is a natural representation for many graphs. For these reasons all GSYM graphs are defined using a three-dimensional co-ordinate system which forms a cube bounded on one side by the display screen of the 2250 Display Unit and roughly bounded on the other sides by the physical housing around the display screen.

This means that an edge is displayed on the screen as a straight line joining the x and y co-ordinates of the Cartesian (x, y, z) co-ordinates of its end-vertices. This three-dimensional Cartesian representation implies that the user must be able to rotate the graph in order to see more than just one face of the graph. Therefore, GSYM has a rotation option whereby the user can rotate the graph about any point in the cube in which the graph is defined. The user specifies this centre of rotation and also the amount of rotation (in degrees) about the x, y or z axis.

Having just described how GSYM operates on three-dimensional entities we pause to describe the only exception to this rule, a two-dimensional entity called the 'light-pen edge'. The light-pen edge is an alternative representation to the straight line representation

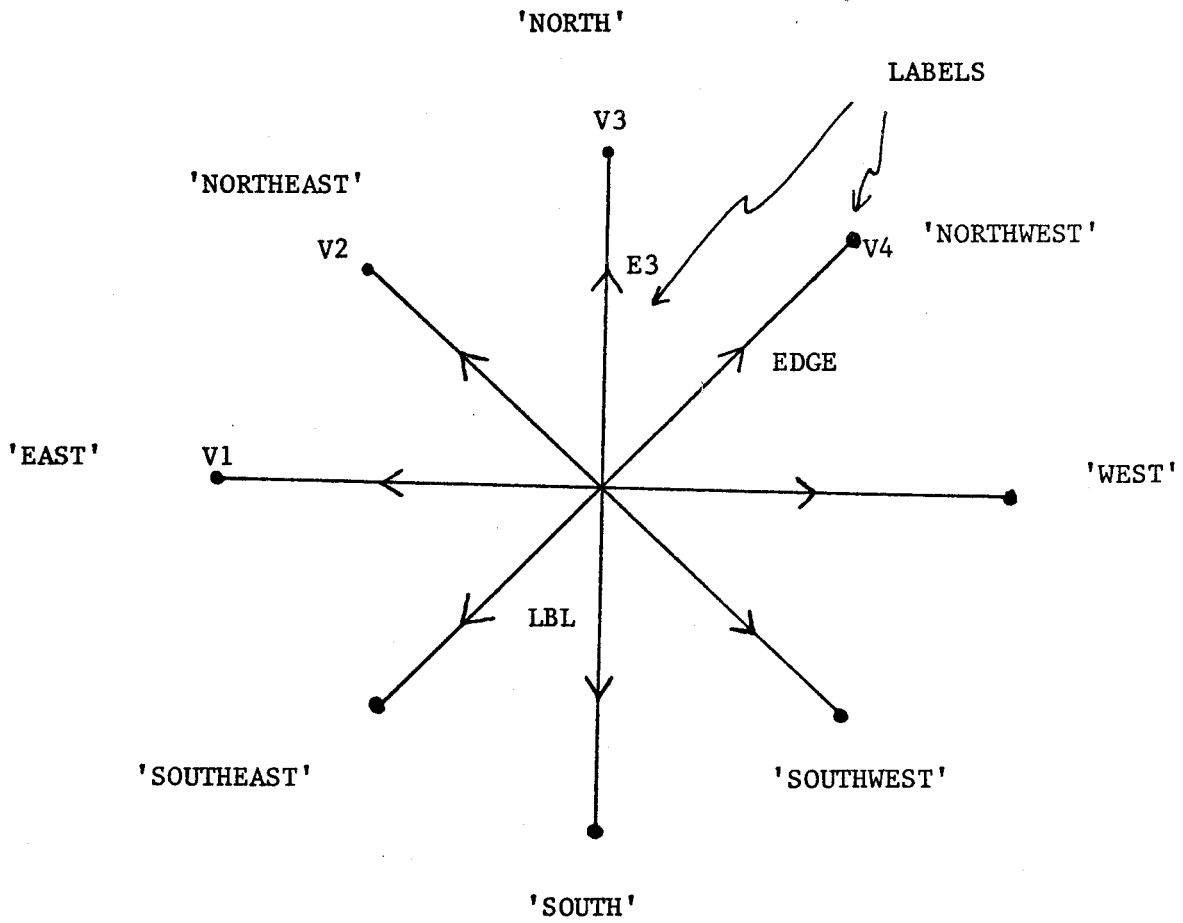


FIGURE 6 : ARROW ORIENTATIONS

of edges. Instead of having the system create a straight line for an edge the user may specify that he wishes to draw a freehand representation of the edge using the light-pen. The edge is then displayed as a sequence of short straight line segments tracing the path followed by the light-pen. Notice that this edge type provides an alternative solution to the problem of displaying multiple edges. For example, multiple edges could be drawn as curved lines between the end-vertices.

There are really two problems involved here. The first problem is that of displaying multiple edges connecting the same two vertices of a graph. On a two-dimensional display screen straight line representation of multiple edges causes a somewhat thicker or brighter line than normal to be displayed between the vertices. That is, it appears as if there is only one edge joining the vertices. This problem can be solved by drawing the edges as curved lines. The GSYM system, however, does not check for the existence of multiples edges between vertices. It is left entirely to the user to recognize and handle such cases of multiple edges. The light-pen edge feature provides a solution should the user wish to make use of it.

The second problem is the fact that, while graph edges are typically drawn as straight lines, there are many instances when curved lines are more appropriate. In other words, there will be graph representations where the ability to use curved edges as provided by the free-hand drawing facility of light-pen edges will be necessary to create a good visual representation of the graph.

Thus, the real purpose of this facility is that it allows the user greater freedom in altering the visual representations produced by representation routines. Straight line edges may be replaced by user drawn edges which improve (in the user's opinion) the visual representation of the graph. These edges should not be used other than for this purpose because, unlike all other elements of the graph, they cannot be treated as three-dimensional entities. It is impossible to draw a curve in 3-space within the present context of the IBM 2250 graphics system. When the user draws a light-pen edge, he is only able to draw the edge as it would be seen from the display screen side of the cube in which the graph is defined. For example, should a graph containing light-pen edges be rotated, the light-pen edges must be redrawn between the new positions of their end-vertices. Figure 7 shows an example of a graph containing all the various elements which may be found in GSYM graphs: labelled and unlabelled vertices and edges, directed edges and light-pen edges (both directed and undirected).

For more detailed information on the GSYM system the reader should refer to Appendix B.

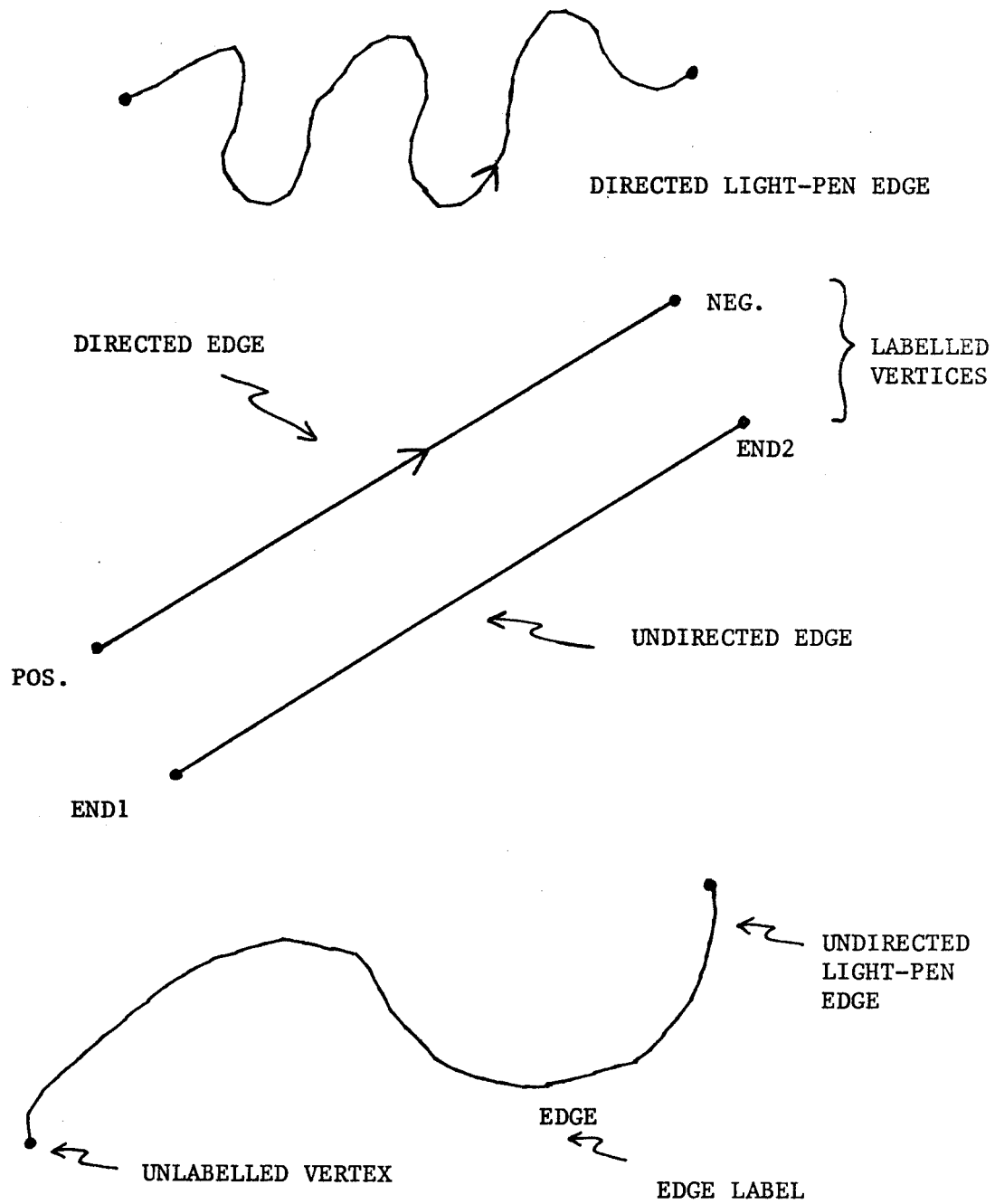


FIGURE 7 : ELEMENTS OF GSYM GRAPHS

APPENDIX A

## APPENDIX A : HARDWARE DESCRIPTIONS

Hardware Descriptions

The IBM S/360-2250 Model 1 Graphic Display Unit is the graphic device on which the GSYM system was implemented. The 2250 Display Unit has been designed to operate in combination with an S/360 computer; in this case the 2250 was connected to an IBM 2911 switching unit which, in turn, was linked to an S/360 Model 75 and an S/360 Model 50. Thus, the 2250 Display Unit and the GSYM system could be used with either machine. The following description briefly describes some of the features of the 2250. It will not discuss the S/360 machines except in their relation to the 2250.

IBM 2250 Display Unit

The IBM 2250 Display Unit is a programmable cathode ray tube (CRT). Although the 2250 is a computer with its own control unit and memory, it is normally expected to be operated as an I/O device attached to a channel of an S/360 computer. The channel is usually a multiplexor channel as the 2250 is basically a low speed device.

The CRT consists of an electron gun and a phosphor-coated screen. The electron gun focuses an electron beam on the screen which causes the phosphor to glow briefly at the point of contact. The movement of the electron gun is controlled by graphic orders placed in the 2250 memory.

Thus, the user programs the 2250 to trace images on the screen, placing graphic orders in the 2250 buffer to create a display. The CRT screen is divided into a 1024 by 1024 grid, with each intersecting point being addressed as an ordered pair  $(x,y)$  where  $0 \leq x,y \leq 1023$ .

The 2250 at the University of Waterloo contains most of the optional features which are available to the Model 1 2250 Display Unit. These include the following:

1) Buffer Memory

The buffer for the 2250 consists of between 4K and 32K bytes of storage in 4K increments. The University of Waterloo 2250 has an 8K buffer. This buffer can be accessed by either the 2250 or the main computer with an access time of 4.2 microseconds per byte. The 2250 uses the buffer as a memory for storing the instructions and data which control the movement of the electron gun. The main computer uses the buffer to pass data and instructions to and from the 2250. While the 2250 executes the graphic orders in the buffer the main computer can continue processing. Thus, the screen may contain a display while the main machine is preparing the next.

2) Absolute Vectors

Without this feature the 2250 could only draw straight lines horizontally, vertically or at a  $45^\circ$  angle. Any other straight line would be drawn as a combination of such lines. With this option, however, the 2250 can draw straight lines between any two points on the screen grid.



3) Character Generator

This option is a hardware device which draws alphanumeric characters on the screen. The characters are stored as one byte data items in the 2250 buffer and when they are to be displayed the character generator directs the movement of the electron gun so as to trace the character. Without this option characters must be generated by software using a sequence of vectors.

4) Function Keyboard

The function keyboard consists of a box with 32 pushbuttons or keys on it. The keyboard is used primarily for interrupting the main computer. An interrupt is sent to the main computer which then requests data from the 2250 to identify the key used. In the GSYM system the keyboard is used mainly to control light-pen operations.

5) Alphanumeric Keyboard

This keyboard is very similar to a typewriter keyboard. Its primary purpose is to type in alphanumeric data which is stored in the 2250 buffer. This data is displayed on the screen as it is typed. A special underscore character is displayed on the screen beneath the position where the next character typed will be placed.

6) Light-Pen

The light-pen is another user control device. It is a pen-like stylus which is attached via a flexible cord to the 2250.

The pen can detect the beam of the electron gun as it passes over the screen. If the pen is enabled when it detects this beam it halts the display and sends an interrupt to the main computer. Information is then sent to the main computer specifying the type of data detected (line or character), the location of the byte accessed in the 2250 buffer and the x,y co-ordinates of the beam. The pen is enabled and disabled under program control. When it is enabled an interrupt occurs as soon as the beam is detected. Otherwise, the pen switch must be activated by pushing the tip of the pen gently against the screen.

Further details on the use and programming of the 2250 Display Unit may be obtained from the IBM manuals listed in the Bibliography (i , ii).

A P P E N D I X B

## APPENDIX B : MORE DETAILS ON GSYM

This appendix contains a more detailed description of the data structures and internals of GSYM than was feasible in the body of the report. It is not a user's guide. However, a thorough understanding of the following material should permit a user to follow and comprehend listings of internal GSYM routines. It should also enable the user to begin to write and connect his own 'user functions' to the system.

GSYM is a collection of both PL/1 (iii) and Assembler (v) programs. PL/1 was chosen because of the built-in list processing features of the language. The majority of the list processing code required for creating and maintaining the graphs and their properties is written in PL/1. While this is more expensive in terms of execution time than Assembler code would be, it has the advantage of being easily written and just as easily changed. Changes to existing routines and new routines can be coded, tested and debugged much faster using the high level language. This is highly desirable because GSYM is intended as a tool for testing various visual representation schemes. In addition, it is expected that the graph theorists for whom this tool was primarily developed are more likely to understand (or will be more willing to learn) a high level language like PL/1. Finally, the facility of being able to signal PL/1 on conditions from Assembler routines for graphic display control and the PL/1 graph processing routines.

All the routines for communicating with the graphic display are coded in Assembler language. The basic attention handling facilities of the Graphic Programming Services for the IBM 2250 are used to control the display.(i) Most of the non-graphic I/O operations of the GSYM system are also coded in Assembler. Finally, many service routines such as the routine for handling system status enquiries are written in Assembler in order to make them more efficient and faster than would be possible using PL/1.

The following figures show the PL/1 data structures used for the various graph entities. In most cases the lists of graph elements have been implemented using PL/1 based variables.(iii)

Figure B.4 shows how the 2250 buffer is used to store the graphic orders which produce the display of the current graph. The free area is used by system routines for creating the various option lists and other displays the user sees while working with GSYM. The graphic orders for these displays are recreated and placed in the buffer each time they are shown so that the free area is also available to user routines. A user creating his own displays must, however, know what portion of this free area is used by any system routines his routines may invoke. For example, a user routine which calls the light-pen tracking routine must not violate the area this routine will use. Moreover, the user routine must restore the status of the buffer to what it was before it took control.

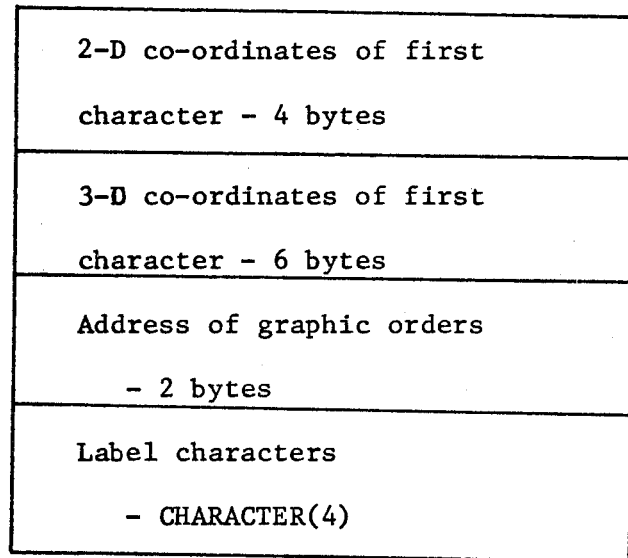
The following descriptions are of the format of the graphic orders corresponding to the various graph entities. The user is referred to the IBM Graphic Programming Services Manual for further details on the meaning and use of the different commands.(i)

Previous vertex 'POINTER'
Next vertex 'POINTER'
Internal name - CHARACTER(4)
Label 'POINTER!'
Address of graphic orders - 2 bytes
Property List pointer - 2 bytes
2-D co-ordinates - 4 bytes
3-D co-ordinates - 6 bytes
Invalence - 2 bytes
Pointer to incident indirected edges - 2 bytes
Outvalence - 2 bytes
Pointer to incident outdirected edges - 2 bytes
Valence - 2 bytes
Pointer to incident undirected edges - 2 bytes

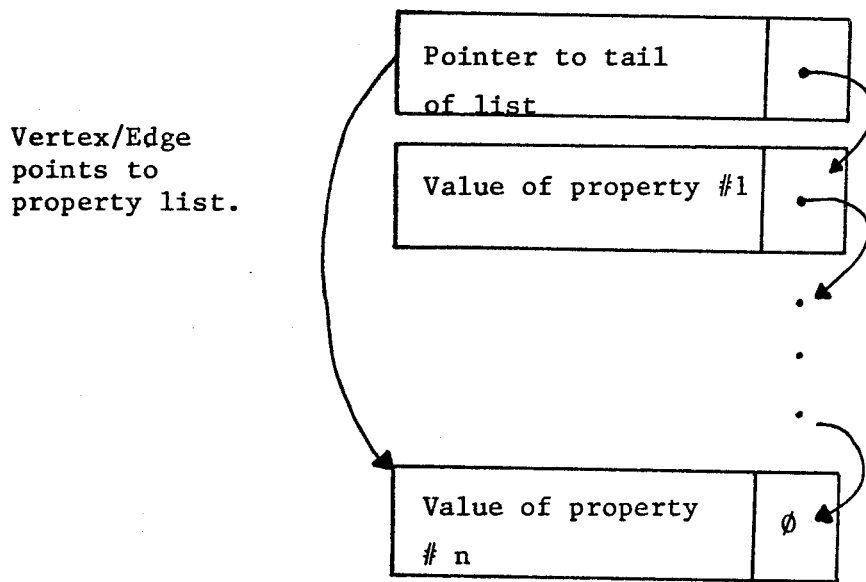
FIGURE B.1 : VERTEX DATA STRUCTURE

Previous edge 'POINTER'
Next edge 'POINTER'
Internal name - CHARACTER (4)
Label 'POINTER'
Address of graphic orders - 2 bytes
Property list pointer - 2 bytes
(Positive) vertex 'POINTER'
(Negative) vertex 'POINTER'
2-D co-ordinates of arrow - 4 bytes
3-D co-ordinates of arrow - 6 bytes
Address of arrow graphic orders - 2 bytes
Arrow orientation - 2 bytes
Edge type - 2 bytes

FIGURE B.2 : EDGE DATA STRUCTURE



Label Data Structure



Vertex/Edge Property List Format

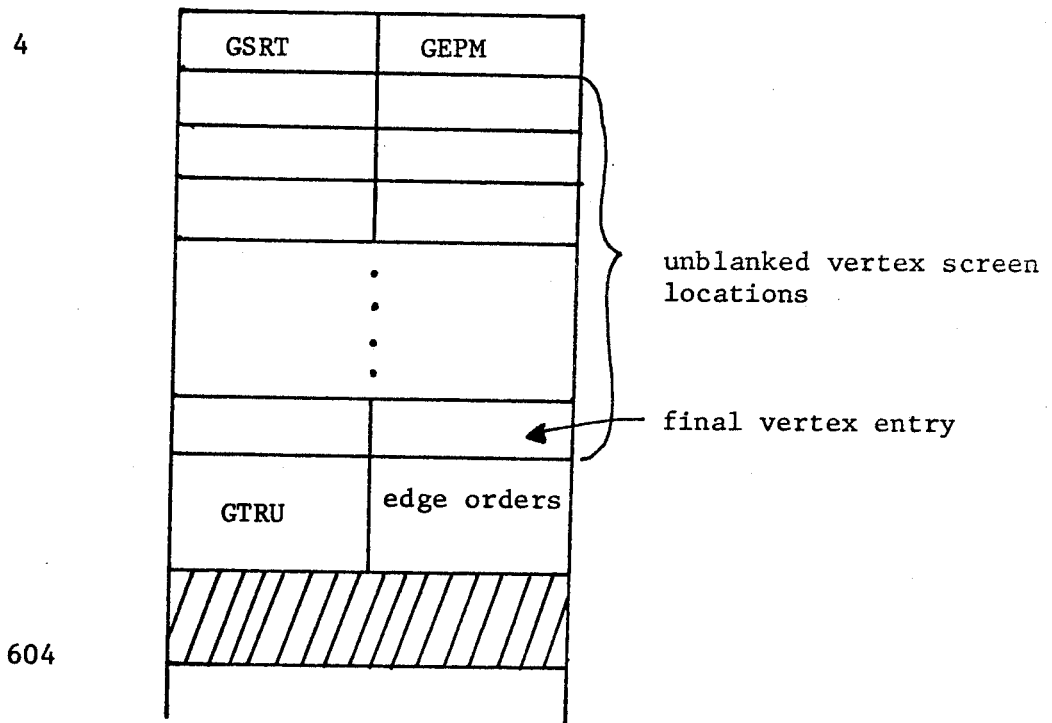
FIGURE B.3




OFFSET		
0	Reserved	
4	GSRT	GEPM
8	GTRU	'604'
12	Vertex Data Area	
604	GNOP2	GEVM
608	GTRU	'1404'
612	Edge Data Area	
1404	GTRU	'3208'
1408	Label Data Area	
3208	GTRU	'4598'
3212	Direction Data Area	
4598	GTRU	'6156'
4602	Light-Pen Edge Data Area	
6156	GTRU	'4'
2032 bytes	Free Area	
	6160	
	8192	

NOTE: Contents of buffer shown as it exists before any elements are added to the graph.

FIGURE B.4 : STRUCTURE OF 2250 BUFFER

Vertex Graphic Orders

There is sufficient room in the vertex buffer area for a maximum of 148 vertices. The buffer areas reserved for the different graph entities may easily be changed by restructuring the buffer and making slight changes in the routines for adding and deleting the various elements. However, in order to increase the size of any one buffer area without decreasing the size of any of the others means decreasing the size of the free area. This is quite feasible as none of the existing GSYM routines use more than a small fraction of this area at any given time. However, the current buffer allocations should be sufficient to contain the maximum amount of data that can be displayed on the screen without encountering display regeneration problems (i.e. screen flicker).

4	GSRT	GEPM
8	$(x_1, y_1), u$	
12	$(x_2, y_2), u$	
	⋮	
596	$(x_{148}, y_{148}), u$	
600	GTRU	'604'
		

Vertex buffer area as it would appear when full.

Edge Graphic Orders

604

GNOP2	GEVM
blanked beam to first vertex	
-----	
unblanked beam to second vertex	
<p style="text-align: center;">⋮</p>	
GTRU	label orders

1404

604

GNOP2	GEVM
$(x_{11}, y_{11}), b$	
-----	
$(x_{12}, y_{12}), u$	
-----	
$(x_{21}, y_{21}), b$	
-----	
$(x_{22}, y_{22}), u$	
<p style="text-align: center;">⋮</p>	
-----	
$(x_{91}, y_{91}), b$	
-----	
$(x_{92}, y_{92}), u$	
GTRU	'1404'


1400

Buffer area as it appears when full.

The edge buffer area contains sufficient space for a maximum of 99 edges. This area is not used for light-pen edges, just for the usual straight line edges. The edge orders in the area represent both directed and undirected edges as the graphic orders for displaying the arrows on directed edges are stored in their own buffer area.

Label Graphic Orders

1404

GEVM	blanked beam to label co-ordinates (x,
y)	GECF B
4 character label	
GEVM	x co-ordinate
y co-ordinate	GECF B
4 character label	
• • •	
GTRU	arrow orders
	

3208

1404

GEVM	x co-ord.
y co-ord.	GECF B
label # 1	
⋮	
GEVM	x co-ord.
y co-ord.	GECF B
label # 150	
GTRU	'3208'

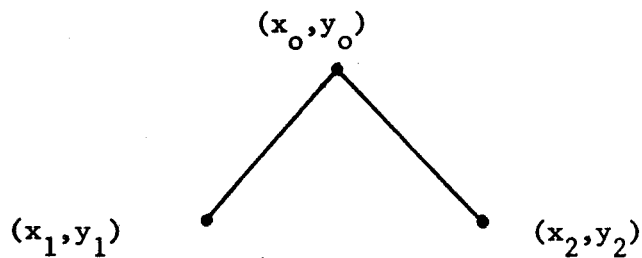
3204

3208

Label buffer area as it appears when full.

The label buffer area is large enough to allow for 150 labelled vertices and/or edges in the graph. Therefore, if the graph has a combined total of more than 150 vertices and edges, it follows that not all of them can be labelled.

#### Edge Direction Orders



3208

GEVM	blanked beam to ( $x_0$ ,
$y_0$ )	GEVI2
unblanked beam to ( $x_1, y_1$ )	blanked beam to ( $x_0, y_0$ )
unblanked beam to ( $x_2, y_2$ )	GEVM
GTRU	light-pen orders
4598	
4600	

↖ may be on  $\frac{1}{2}$   
word boundary

The incremental mode is used for the arrow orders because this form only requires 14 bytes per arrow whereas the equivalent absolute mode orders would require 18 bytes per arrow. Using the incremental mode also means that the graphic order will be the same for all arrows with the same orientation. Note that by setting up different pairs of values

for  $(x_1, y_1)$  and  $(x_2, y_2)$  we can generate the different orientations using the same graphic orders. There is sufficient buffer space for a maximum of 99 edge arrows. This corresponds to the maximum number of straight line edges. But light-pen edges may also be directed so the total number of directed edges regardless of type cannot exceed 99..

3208

GEVM	$(x_{10},$
$y_{10}),b$	GEVI2
$(x_{11},y_{11}),u$	$(x_{10},y_{10}),b$
$(x_{12},y_{12}),u$	GEVM
	• • • •
	GTRU
'4598'	

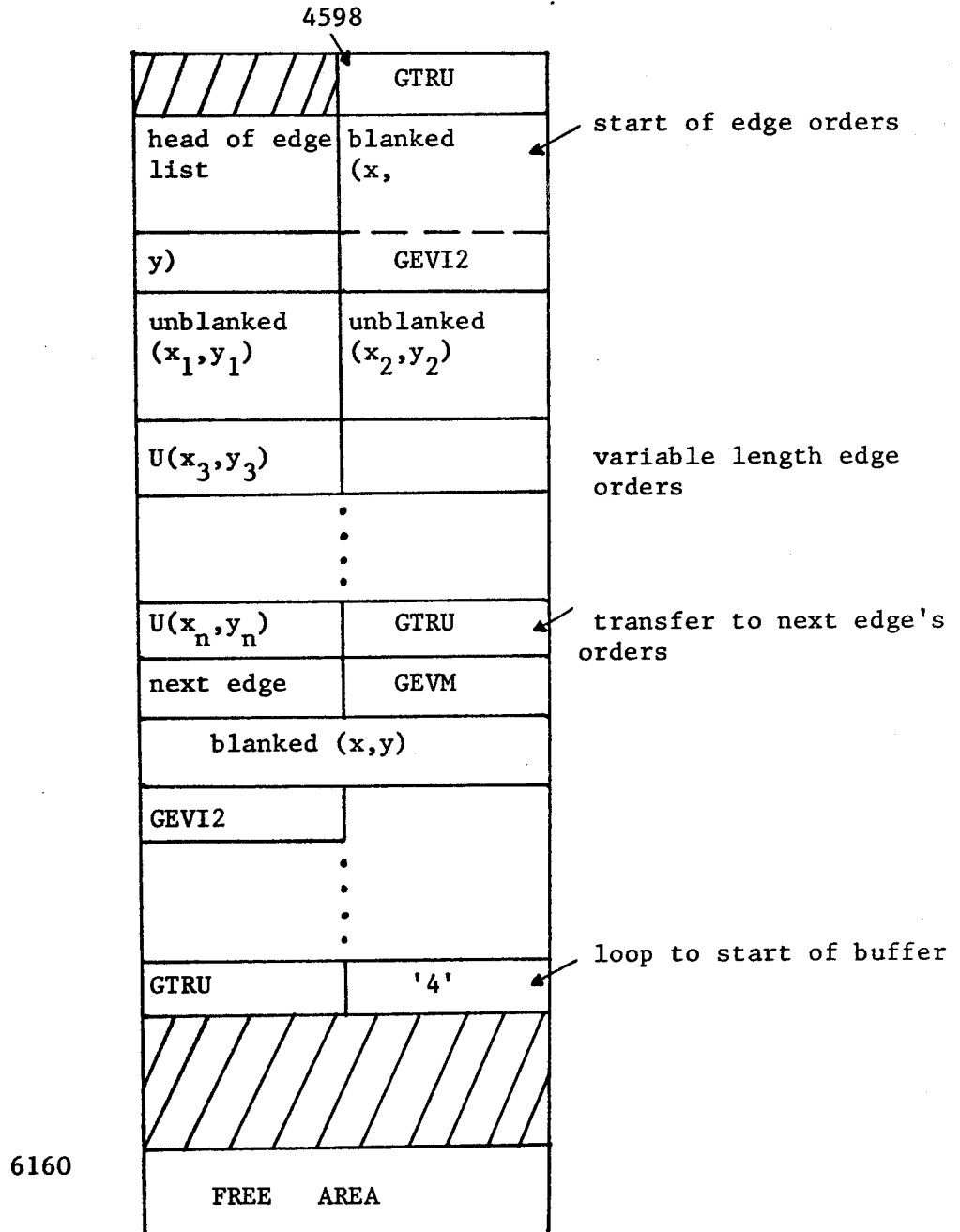
4592

4596

Arrow buffer area as it appears when full.



Light-pen Edge Graphic Orders



It is impossible to estimate the average length of the orders for each edge since this will vary for each user. There is, however, room

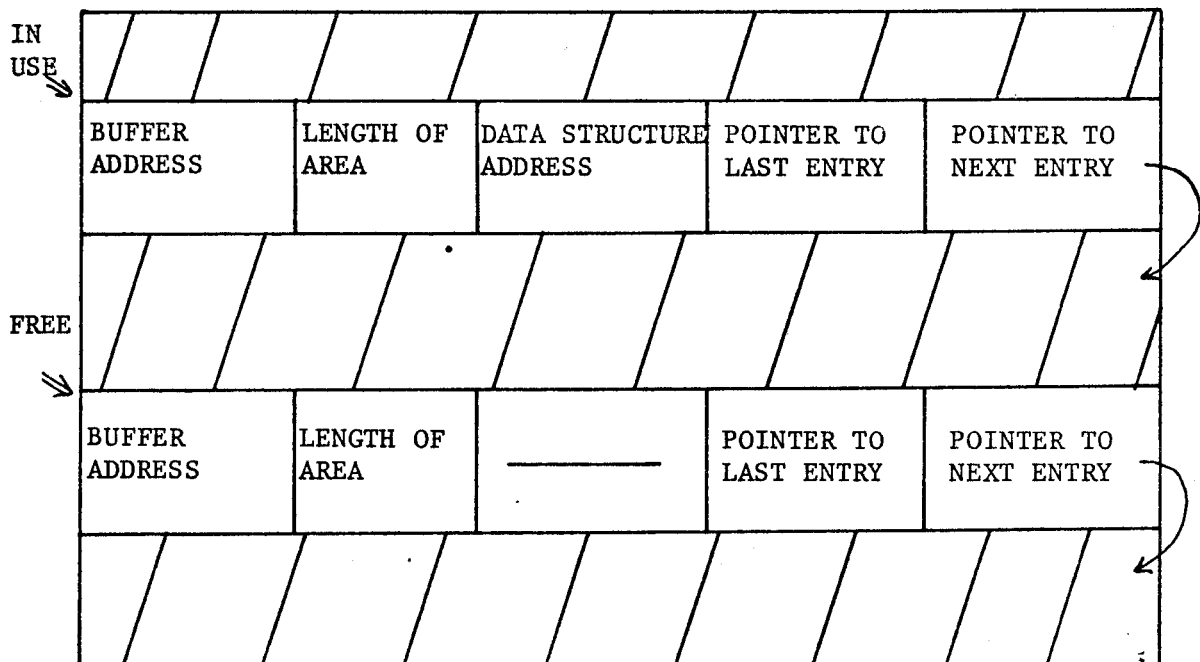
for approximately 30 edges if each edge takes about 20 line increments. But then it is also possible to draw an edge so long that it will not fit into the light-pen edge buffer area even if this area was previously empty.

#### Addition and Deletion Buffer Operations

All routines for altering the graph by adding and deleting graphic orders from the 2250 buffer operate in essentially the same manner. The sole exception is the light-pen edge routine. For all other graph entities, if an element is to be added to the graph, the routine first checks whether there is space left in the buffer for the element. If there is, then the graphic orders required to display the new element on the screen are generated and placed at the bottom of the corresponding buffer area. These routines maintain a pointer to the most recent entry in each buffer area. If there is no room for the element the user is simply informed that the buffer is full.

When an element is to be deleted the routine is passed the address of the graphic orders to be deleted. Each time an element is added to the graph the address of the graphic orders generated for the element is saved in the data structure corresponding to the element. The routine then copies the orders for the most recent entry of this type into the space which is no longer needed. Thus, each time an element is deleted the buffer area is compressed, avoiding the problem of buffer fragmentation. Of course, if the element to be deleted is the most recent entry the pointer to the most recent entry is simply reset to point to the second most recent entry.

The above scheme will not work for light-pen edges since the graphic orders for the different edges differ in length. That is, it is not possible to keep the buffer area intact after a deletion without recopying much of the buffer. Thus, we cannot avoid the buffer fragmentation problem. The table below is used to keep a record of each light-pen edge entry in the buffer. The table is initialized so that the only entry is a free area entry representing the whole light-pen edge buffer area. When the first light-pen edge is added to the graph this entry is split into an entry representing the buffer area used for the edge orders and an entry representing what was left of the free area. Thus, as long as edges are added to the graph a linked list of 'in use' buffer areas is built up and there remains one entry representing whatever free area is left. Each entry contains the starting address and length of the orders for the corresponding edge.



When a light-pen edge is deleted the linked list of 'in use' entries is searched until the correct entry is found. This entry is then moved to the tail of the 'free' list. The free list contains only one entry until the user starts deleting light-pen edges. Therefore, the usual situation will be that the light-pen edge buffer area is partitioned into holes of 'in use' areas and 'free' areas. When an edge is added its buffer area is taken from the first entry in the 'free' list with enough space to hold the orders generated for the new edge. Any space that is not used is returned to the 'free' list.

Eventually, the light-pen edge buffer area will become fragmented into 'in use' areas and 'free' areas, the 'free' areas being quite small. This situation is allowed to continue until there is an edge to be added and none of the 'free' entries are large enough to hold it. The buffer area is then compressed by moving all the 'in use' areas to the start of the buffer area (i.e. the orders are recopied) and the buffer table is recreated. The 'free' areas are thusly transformed into one large 'free' area at the bottom of the light-pen edge buffer area. Should this area still not be large enough for the new edge the user is informed that the buffer area is full.

#### System -- User Routine Linkage

For each function name listed in the miscellaneous functions option list (see Figure 5) for which no actual routine exists, there is a dummy entry point in the routine SYS1. Suppose for example that a user wishes to add a routine called USER6 (only names allowed are those in the miscellaneous functions option list) to the system. He must remove the ;

USER6 entry point from SYS1 and recompile and relinkedit SYS1. He then inserts a subroutine call to USER6 in the routine called 'GSYM'. 'GSYM' must also be recompiled and relinkedited. The user must then change the job control language for running GSYM so that the new routine is 'included' when all the system load module libraries are linkedited together at run time. Finally, he must compile and linkedit the new routine USER6 and place the load module obtained in one of the system load module libraries. The following is a list of the existing system and user routines which may be invoked by using the miscellaneous functions option.

#### SYS5 -- Light-Pen Edge Curve Fit

This routine allows the user to perform a curve fit on a specified light-pen edge. The purpose of this facility is to be able to create a smoother version of an edge that was drawn using the light-pen. This is occasionally necessary because of the poor resolution of the light-pen. In order to be able to draw a smooth curve it is necessary to use a relatively small light-pen target. However, the smaller the target the more likely the pen is to strike the target at random points on the circumference of the target as it moves about the screen. Thus, this routine was written in an attempt to improve through software what is essentially a deficiency in the graphics hardware.

#### SYS6 -- Adjust Edge Labels and Arrows

This routine will move an arrow associated with a directed edge to the centre of the edge (except for light-pen edges) and display the arrow with the proper orientation. If the edge is labelled the label is positioned just off the centre of the edge. The user may select the

edge or may specify that this operation is to be performed on all directed edges in the graph. This routine is intended to make it a little easier for the user to position arrows and labels on edges.

#### SYS7 -- Trace

This routine allows the user to set up a selective trace of the internal operations performed by the system. This is expensive and should only be used as a debugging tool.

#### SYS8 -- Buffer Dump

This is another debugging tool which allows the user to have part or all of the contents of the 2250 buffer printed out. This routine is very useful in the debugging of user routines which manipulate the 2250 buffer.

#### SYS9 -- Graph Input

When this routine is invoked a description of a graph is read from a user specified (through job control language) disk file and displayed on the screen. The graph is displayed without regard to its structure. That is, the vertices are arbitrarily placed in rows on the screen and the edges inserted between the appropriate vertices. This routine allows the user to bring graphs into the system without having to create them element by element.

#### USER1 -- Automorphism Group Calculation

This routine consists of automorphism group calculation and graph display routines currently being investigated using the GSYM system.

System Error Messages

The following is a list of error messages that may be generated by the system when an error is detected. Note that these are recoverable errors, that is, the user has made an error such as typing a name wrong or a similar easily corrected error. It is also possible for the system to terminate abnormally due to a graphic I/O error. However, most of the GSYM graphic I/O routines have been purposely designed to retry the I/O operation several times before abending. Thus, such errors are extremely remote unless there is a serious system hardware fault.

INTERNAL NAME ERROR  
UNDEFINED NAME USED  
END NAME MISMATCHED  
INVALID ORIENTATION  
INVALID PROPERTY  
USE ALTER OPERATION TO CHANGE NON-NULL LABEL  
INVALID NUMBER  
INVALID LIST TYPE  
INVALID OPTION  
USE ADD OPERATION  
NULL LABEL IMMOVABLE  
ELEVENTH PROPERTY  
NAME ALREADY IN USE  
POINTER LIST FULL

INVALID AXIS

LAST INTERNAL NAME

INVALID BUFFER ADDRESS

2250 BUFFER AREA FULL

### Macro Data Tables

We shall conclude this appendix with a list of the data required for the various internal operations performed in manipulating the graph currently on display. Before any system operation begins all the data required to complete the operation is collected in the PL/1 structure shown in Figure B.5. The data is typed in by the user if the system is in macro mode. However, if the system is in normal mode then the system derives this information from the usual 'display-user response' sequence. For example, when the user positions the light-pen target on the screen in order to create a new vertex the screen co-ordinates of the target centre are placed in this structure. The rest of the appendix shows what values are required for each operation. These tables are necessary for anyone trying to invoke system routines from his own user routines.



```
1 MACRO,  
  2 TYPE,  
  2 PROP#,  
  2 CORD1, (3 X, 3 Y, 3 Z)  
  2 CORD2, (3 X, 3 Y, 3 Z)  
  2 PLUS,  
  2 NEG,  
  2 LBL,  
  2 LBL-TYPE,  
  2 ARROW,  
  2 LIST(10),  
  2 LTPEN
```

FIGURE B.5

	CORD1	CORD2	PLUS	NEG	LBL
V	vertex location	label location			label
E		label location	end 1 ptr.	end 2 ptr.	label
DIR	arrow location		undirected edge ptr.	positive end ptr.	
DE	arrow location	label location	positive end ptr.	negative end ptr.	label
VLBL		label location	vertex ptr.		label
ELBL		label location	edge ptr.		label

	LBL_TYPE	ARROW	LTPEN	PROP#	LIST
V	label type			# of properties	property values
E	label type		ltpen flag	# of properties	property values
DIR		arrow orientation		edge type	
DE	label type	arrow orientation	ltpen flag	# of properties	property values
VLBL	label type				
ELBL	label type			edge type	

NOTE: In all tables values not shown are undefined. This also applies to elements of 'MACRO' which are not listed.

Delete Operation

	PLUS	LBL_TYPE	ARROW	LTPEN
V.	vertex ptr.			all flag
E	undirected edge ptr.		edge type	all flag
DIR	directed edge ptr.		edge type	all flag
DE	directed edge ptr.		edge type	all flag
PRTY.		vertex/edge code	pointer to property in vertex or edge property list	all flag

NOTE: If the 'all flag' is on no other data is required for the deletion.

Alter Operation

	PLUS	NEG	LBL	LBL_TYPE	ARROW	LTPEN
DIR	directed edge ptr.				new arrow orntn.	reverse direction flag
VLBL	vertex ptr.		label	label type		
ELBL	edge ptr.		label	label type		
PRTY	vertex or edge ptr.	vertex or edge ptr. or numeric value		vertex/ edge code	locn. of prop. in list	
PTRVAL	vertex or edge ptr.			vertex/ edge code	locn. of prop. in list	

Point Operation

PLUS - pointer to the first vertex or edge in the list.

NEG - pointer to the second element in the list.

LBL\_TYPE - location of head of list 'POINTER' in GSYM pointer  
list or location of property associated with list.

ARROW - location of property in vertex list.

LTPEN - turned on if LBL\_TYPE points to 'POINTER'.

PROP# - # of property lists to alter.

LIST - pointers to the remaining elements in the list.

Move Operation

	CORD1	PLUS	NEG	LIST
V	new location	ptr. to vertex		
E		ptr. to edge	ptr. to new end 2	ptr. to new end 1
DE		ptr to directed edge	ptr. to new neg. end	ptr. to new pos. end
VLBL	new location	ptr to vertex		
ELBL	new location	ptr. to undirected edge		
DELBL	new location	ptr. to directed edge		
ARROW	new location	ptr. to directed edge		

Declaration Operation

	PLUS	LBL	LBL_TYPE	ARROW	LTPEN
VPRTY		property name			
EPRTY		property name			
PTR	vertex or edge ptr.	pointer name	location of property in vertex list	location of property in edge list	defer flag
ASSOC	vertex or edge ptr.		location of property in vertex list	location of property in edge list	defer flag



Translation Operation

CORD1 - the x, y and z increments by which the co-ordinates of all the elements in the graph are to be shifted.

Rotation Operation

CORD1 - the co-ordinates of the centre of rotation.

LBL TYPE - rotation axis code.

PROP# - amount of rotation in degrees.

B I B L I O G R A P H Y

## BIBLIOGRAPHY

- ( i) IBM System/360 Operating System Graphic Programming Services for the IBM 2250 Display Unit, Form C27-6909-5.
- ( ii) IBM System/360 Component Description IBM 2250 Display Unit Model 1, Form A27-2701-2.
- (iii) IBM System/360 Operating System PL/1 (F) Language Reference Manual, Form C28-8201-2.
- ( iv) IBM System/360 Operating System PL/1 (F) Programmer's Guide, Form GC28-6594-7.
- ( v) IBM System/360 Operating System Assembler Language, Form GC28-6514-6.
- ( vi) IBM System/360 Operating System System Control Blocks, Form GC28-6628-4.
- (vii) IBM System/360 Operating System System Programmer's Guide, Form C28-6550-6.
- (viii) IBM System/360 Operating System Supervisor and Data Management Macro Instructions, Form GC28-6647-3.
- ( ix) IBM System/360 Operating System Supervisor and Data Management Services, Form C28-6646-2.