APPENDIX A

VIEWPLOT IMPLEMENTATION

This appendix contains documentation of the implementation of the fast general object space algorithm described in Chapter 3. It contains a program summary, a users' guide and a logic manual.
Viewplot is a program that plots 3-dimensional objects in true perspective in 2 dimensions. It plots only the visible edges and shades the visible portions of the faces as if a light were coming from the north-west.

I call the corners of the object, where edges meet or change direction, vertices. Each vertex is defined by its 3-D Cartesian coordinates. Each edge or border where the surface changes directions is defined by the numbers of the 2 vertices at its ends. Each face or surface is defined by the numbers of the vertices at its corners, in order. You prepare the definition of the object in a file and then read it into Viewplot or type it into Viewplot directly. The edges must be straight lines and the faces flat planes. Also the faces cannot intersect each other. The faces are considered opaque and hide whatever is behind them as seen
by an imaginary viewer.

The objects need not be convex polygons; a ring with a hole is legal. Neither need they be connected; another legal object would be all the buildings in Harvard Yard. Neither need the edges be orthogonal; Gund Hall is a legal object. A useful object would be a set of prisms where the base of each is the outline of a state or county and the height the population. 2 faces of an object cannot intersect one another: the object composed of 2 tetrahedra rotated about the common centre is illegal. In this case you have to find the face intersections by hand and add new edges and split the faces up where they cross. Eventually this restriction will be removed. This will allow making new objects by superimposing old objects.

The viewpoint, or location of the imaginary observer can be anywhere outside the object. If it is far away, the projection is effectively orthogonal, while if close the perspective is very evident. You can also use an inverted perspective. Here the object is projectively transformed normally. But then, instead of seeing the edges and faces closest to you, you see the ones farthest.
When defining the object, you declare the edges to be type 1 or type 2. They are equal in all respects until the time comes to draw any of the them that are visible. Then the type 2 edges are not drawn. This is useful if you are approximating a curved face by several long and narrow flat faces. A face's edges must be defined but you don't want all these close edges drawn. Therefore make them type 2.

Faces are also partitioned into 2 types but here the difference is more important. It may happen that parts of the object are composed of closed polyhedra. They need not be convex, but they must have a definite inside and outside. In this case, faces and edges on the back are guaranteed to be hidden which speeds the calculations a lot. VIEWPLOT cannot detect this case itself (nor will it in the foreseeable future) so you have to tell it. Do this by making all the faces on closed solids type 2 faces and the rest type 1. Naturally one object can have both types of faces.

You can transform an object (change its size, location, orientation etc.). The object can be shifted by a given amount in the X-, Y- and Z- directions. The object's size can be increased by a given factor in each direction. Thus
a cube can be changed to a brick by scaling it by 1, 2, 4 in the X-, Y- and Z-directions respectively. You can rotate an object by a given amount about a given axis. You can scale an object to fit a given box. In general if the object keeps its proportions, it will only touch 2 opposite sides of the box and not the other 4. There is an option to change its proportions so that it will touch 4 or all 6 sides. Say the object is a set of prisms with X and Y units miles and Z units population. Then this command can change this object's probably ridiculous proportions to make its Z extent 30% of the maximum of its X and Y extents by making it fit a box of size 10,10,3 and having only X & Y stay in proportion. In addition, a general linear transformation can be applied to the object. This includes affine transformations which distort squares to look like diamonds. Finally you can do a permanent projective transformation to the object. Then when you plot it, it will be transformed again, for interesting effects.

There is an interactive editor which allows you to add, delete and change the object's vertices, edges and faces.
All the above operations used only one object at a time. There are also some 2 object commands. First you can copy one object to another. Then you can combine 2 objects. This is a basic and easiest way to create complex objects. For example say I want to build an array of 2 by 2 by 2 cubes. I take 1 cube, say its size is 1, and copy it. Then I shift the copy by 1.5 in the X direction and combine the original cube with it. Now I have 2 cubes in 1 object, side by side with a little space between them. Now I copy this object, shift the copy in the Y direction and combine to produce a square of 4 cubes. Repeat once more to produce the final cube of 8 cubes. Once an object is combined, it cannot be split apart except by using the editor to delete all the vertices corresponding to one or the other object. There is also a command to move an object to abut against another before combining them. This is a useful command to make buildings from cubes. It actually abuts boxes around the objects but for convex objects there is little difference and for cubes none. When you move object A to abut the XY face of object B by shifting A in the Z direction, you can choose how to centre A on B by shifts in X and Y. You can centre it, centre it on an edge or put it in a corner. In the latter 2 cases, A can be inside or
outside B (as projected on the XY plane). In all there are 25 ways to centre A on B. Of course if you wish you can do no centring at all. You can also move A to abut on B but not combine it. This is useful if you then want to move A back a little to leave an empty space and then combine it with B. When finished you can write out the new object in an ASCII file.

100 objects are allowed at once. Each object can have up to about 2000 edges. The exact maximum depends on the number of vertices, edges, faces and the sum of the number of vertices in each face. There is no restriction on the total size of all the objects since they are stored in disk when not in use. Further an object has no maximum size when it is created. Only as much storage is used as necessary and if you read in a bigger object or combine another onto this, more space is allocated. If you try to put data (by reading, copying or combining) into a nonexisting object, it will be created.

Initially there are several predefined objects such as 5 of the regular solids, a square ring, a cuboctahedron, a skew cube, prisms of different shaped bases, etc. I am regularly adding new useful looking objects to this base.
VIEWPLOT IMPLEMENTATION

VIEWPLOT Summary

The fastest way to get a plot is to start VIEWPLOT and plot one of these predefined objects thus:

```
.cop = [3111,535]vplt.sav,status.vpt
.run vplt

[VIEWPLOT VERSION OF 10-JULY-76]
*plot icosa
READY TO PLOT

(An icosahedron is plotted now)

*end
```

Maximum sizes: You can now have objects up to about 2500 edges. There is no simple rule for the maximum; it depends on temporary results calculated as the object is plotted.

Running time: I plotted an object without shading composed of a cubic array of 216 cubes that had 2592 edges in under 4 minutes. This was using a poor compiler on a slow computer. A smaller object, an array of 64 cubes with 768 edges took 1:10, while a very small object, an array of
8 cubes with 96 edges, plotted in 10 seconds. The algorithm uses locality in that under reasonable assumptions as to how the average edge length gets smaller as the number of edges get greater, the CPU time used depends on NE**1.25 where NE is the number of edges. In the 3 examples above the time rose even slower than that, in fact not much faster than linear. On the other hand, if the object is a set of pick-up sticks, all interlaced with the edges all going from one side of the screen to the other then that would take a lot longer because each edge would intersect most of the other edges. Since the algorithm depends on knowing which edges intersect other edges (in 2-D projection) short edges take less time to calculate and plot. Plotting with shading will probably take twice as much time as without.

Currently VIEWPLOT takes about 70K of core on a PDP-10. 40K of this is for the code and could be reduced about 20K by overlaying. Another 30K is for an array to hold all temporary data during plotting. Its size depends on the how big an object you want to be able to plot.

VIEWPLOT has an associated file, STATUS.VPT, that contains 2 things: 1) the current values of all parameters, such as the position of the viewpoint and whether plots
should have inverted perspective, and 2) all the existing objects. If you end VIEWPLOT normally, and save STATUS.VPT, VIEWPLOT will read it when you run it next. Thus you continue with all the objects and parameter settings you left off with. As far as the program is concerned, the only difference is that any files you were writing were closed and new copies are now being written. Of course STATUS.VPT can be renamed and copied like any other file. This in fact is how the library objects are implemented. I ran VIEWPLOT with a null status file and read in the library objects. When I ended, I had a status file with the objects predefined that anyone can copy to his disk space and use. The only problem is that is VIEWPLOT should bomb or you type ^C, will be garbage in which case you have to copy it again.

If you want to run VIEWPLOT but lack a status file, you can make a null one by

.cop status.vpt=nul:a

VIEWPLOT will run with default parameters and no predefined objects.
VIEWPLOT IMPLEMENTATION

VIEWPLOT Summary

VIEWPLOT accepts command files. Thus for example you can have a command file that makes a cube of 8 copies of an object and apply it to any object you want. Very big objects take a lot of space (the 64 cube array takes 150 disk blocks in STATUS.VPT and even more in an ASCII file) so it is better to keep a file of the commands you used to create them and then read it as command file.

Things to do:

1. The shading is not finished; I know how to do it but it is messy.

2. Allow input of prisms and their heights. This is fairly simple, only involving an input routine. A more difficult problem is that if VIEWPLOT can handle say 2000 vertices, this means only 10 polygons of 100 vertices. (The factor of 2 is to account for the tops and bottoms). To handle 50 polygons that size requires more modifications. This looks hard.

3. Currently VIEWPLOT plots the result directly on a TEKTRONIX. I will add an option to write the visible lines are polygons on a file with extra info telling which edges and faces they originally came from so that you can write a
program do to any shading you want. This is easy.

4. If VIEWPLOT is to be used with complex objects, add better editing facilities using GIN mode.

5. Convert plotter calls to use BUFTEK.MAC so that the full capabilities of the 44014 are available for shading. This is not too hard.


7. See how VIEWPLOT is used and optimize in that direction.

8. Add clipping so the viewpoint can be inside the object.

9. Add code so that 2 overlapping objects can be combined. This looks hard.

There are 2 other related documents: a description of the algorithm and a program logic manual.
VIEWPLOT

USERS' MANUAL

Wm. Randolph Franklin
Laboratory for Computer Graphics and Spatial Analysis
Harvard University
Cambridge, Mass., 02138
December 19, 1976

Copyright, 1976, by the President and Fellows of Harvard College, Cambridge, Mass., 02138. All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage or retrieval system, without permission in writing.
# TABLE OF CONTENTS

## SECTION | PAGE
---|---
1. Table of Contents | 2
2. Introduction | 3
3. The Perspective Projection | 4
4. Internal Format of Objects | 5
5. Operations on 1 Object | 6
6. Operations on 2 Objects | 7
7. Other Commands | 8
   1. Information Commands | 8
   2. Parameter Setting Commands | 8
   3. I/O Commands | 8
   4. END Command | 8
8. Predefined Objects | 9
9. STATUS.VPT | 10
10. Command Files | 12
11. Maximum Sizes | 13
12. How to Use VIEWPLOT | 14
13. Command Syntax | 15
14. Examples | 27
15. Running Time and Size | 31
16. Errors | 32
17. NEWS.VPT | 32
18. How to Get Help | 33
2.0 INTRODUCTION

VIEWPLOT is a program that plots 3-dimensional objects in true perspective in 2 dimensions. It plots only the visible edges. This manual tells you how to use it on the Harvard University Aiken Computation Lab PDP-10, although it would be similar anywhere else. You need no special knowledge of geometry, architecture or computer science to use VIEWPLOT although a basic acquaintance won't hurt.

Figure 1: 2 Interlocked Rings
3.0 THE PERSPECTIVE PROJECTION

The projection is a true 3 point perspective in Cartesian 3-D space defined by the VIEWPOINT from where you are looking and the PERSPECTIVE PLANE on which the projection falls. This plane is defined by the point on it, called the CENTERPOINT, closest to the viewpoint - the foot of a perpendicular dropped to the perspective plane from the viewpoint.

You can change the default projection by giving new values to the viewpoint and centerpoint. The centerpoint is generally at the centre of the object. The viewpoint can be anywhere outside the object. If it is far away, the projection is effectively orthogonal, while if close the perspective is very evident.

You can also use an inverted perspective. Here the object is projectively transformed normally. But then, instead of seeing the edges and faces closest to you, you see the ones farthest. Alternatively you can consider this to be a projection with the viewpoint farther than infinity. That is the lines of sight diverge instead of converging.
4.0 INTERNAL FORMAT OF OBJECTS

The OBJECT is a set of data in VIEWPLOT. It may represent one 'thing' such as a cube or many such as all the buildings on a block.

I call the corners of the object, where edges meet or change direction, VERTICES. Each vertex is defined by its 3-D Cartesian coordinates. Each EDGE or border where the surface changes directions is defined by the numbers of the 2 vertices at its ends. Each FACE or surface is defined by the numbers of the vertices at its corners, in order. You prepare the definition of the object in a file and then read it into VIEWPLOT or type it into VIEWPLOT directly. The edges must be straight lines and the faces flat planes. Also the faces cannot intersect each other. The faces are considered opaque and hide whatever is behind them as seen by an imaginary viewer.

The objects need not be convex polygons; a ring with a hole is legal. Neither need they be connected; another legal object would be all the buildings in Harvard Yard. Neither need the edges be orthogonal; Gund Hall is a legal object. A useful object would be a set of prisms where the base of each is the outline of a state or county and the height the population. 2 faces of an object cannot intersect one another: the object composed of 2 tetrahedra rotated about the common centre is illegal. In this case you have to find the face intersections by hand and add new edges and split the faces up where they cross. Eventually this restriction will be removed. This will allow making new objects by superimposing old objects.

When defining the object, you declare the edges to be type 1 or type 2. They are equal in all respects until the time comes to draw any of the them that are visible. Then the type 2 edges are not drawn. This is useful if you are approximating a curved face by several long and narrow flat faces. A face's edges must be defined but you don't want all these close edges drawn. Therefore make them type 2.

Faces are also partitioned into 2 types but here the difference is more important. It may happen that parts of the object are composed of closed polyhedra. They need not be convex, but they must have a definite inside and outside. In this case, faces and edges on the back are guaranteed to be hidden which speeds the calculations a lot. VIEWPLOT cannot detect this case itself (nor will it in the foreseeable future) so you have to tell it. Do this by making all the faces on closed solids type 2 faces and the rest type 1. Naturally one object can have both types of faces.
5.0 OPERATIONS ON 1 OBJECT

You can transform an object (change its size, location, orientation etc.). The object can be shifted by a given amount in the X-, Y- and Z- directions. The object's size can be increased by a given factor in each direction. Thus a cube can be changed to a brick by scaling it by 1,2,4 in the X-, Y- and Z-directions respectively. You can rotate an object by a given amount about a given axis.

You can scale an object to fit a given box. In general if the object keeps its proportions, it will only touch 2 opposite sides of the box and not the other 4. There is an option to change its proportions so that it will touch 4 or all 6 sides. Say the object is a set of prisms with X and Y units miles and Z units population. Then this command can change this object's probably ridiculous proportions to make its Z extent 30% of the maximum of its X and Y extents by making it fit a box of size 10,10,3 and having only X&Y stay in proportion.

In addition, a general linear transformation can be applied to the object. This includes affine transformations which distort squares to look like diamonds. Finally you can do a permanent projective transformation to the object. Then when you plot it, it will be transformed again, for interesting effects.

There will eventually be an interactive editor which allows you to add, delete and change the object's vertices, edges and faces.
6.0 OPERATIONS ON 2 OBJECTS

All the above operations used only one object at a time. There are also some 2 object commands. First you can copy one object to another. Then you can combine 2 objects. This is a basic and easiest way to create complex objects. For example say I want to build an array of 2 by 2 by 2 cubes. I take 1 cube, say its size is 1, and copy it. Then I shift the copy by 1.5 in the X direction and combine the original cube with it. Now I have 2 cubes in 1 object, side by side with a little space between them. Now I copy this object, shift the copy in the Y direction and combine to produce a square of 4 cubes. Repeat once more to produce the final cube of 8 cubes. Once an object is combined, it cannot be split apart.

There is also a command to move an object to abut against another before combining them. This is a useful command to make buildings from cubes. It actually abuts boxes around the objects but for convex objects there is little difference and for cubes none. When you move object A to abut the XY face of object B by shifting A in the Z direction, you can choose how to centre A on B by shifts in X and Y. You can centre it, centre it on an edge or put it in a corner. In the latter 2 cases, A can be inside or outside B (as projected on the XY plane). In all there are 25 ways to centre A on B. Of course if you wish you can do no centring at all. You can also move A to abut on B but not combine it. This is useful if you then want to move A back a little to leave an empty space and then combine it with B. When finished you can write out the new object in an ASCII file.
7.0 OTHER COMMANDS

7.1 Information Commands

There are some commands that don't change anything. They just list data such as the names of the current objects, the parameter settings or all the vertices, edges and faces of a given object. COMMENT doesn't do anything but let you include remarks in command files.

7.2 Parameter Setting Commands

These are commands that don't change any objects but change parameters that affect the plot. For example you can change the viewpoint or centerpoint. You can have a 10 by 10 grid superimposed on the plot. Timing and other statistics can also be printed during the hidden line calculations for debugging VIEWPLOT.

7.3 I/o Commands

READ inputs an object into VIEWPLOT and WRITE writes objects to a file. This is a command file ready to be read by command CFILE.

7.4 End Command

This returns you from VIEWPLOT to the monitor. It should be used rather than ^C or else files you have written may disappear and STATUS.VPT be rendered unusable.
8.0 PREDEFINED OBJECTS

Initially there are several predefined objects such as 5 of the regular solids, a square ring, a cuboctahedron, a skew cube, prisms of different shaped bases, etc. I am regularly adding new useful looking objects to this base. Do a DIR command when starting VIEWPLOT to list the current objects. Now they include:

TETRA Tetrahedron
CUBE Cube
OCTA Octahedron
DODECA Dodecahedron
ICOSA Icosahedron
CUBOCT Cube-octahedron
SKEWCUBE Skew-cube
3PRISM TRIANGULAR prism
RING 4 sided ring
2RINGS 2 interlocking rings
2RINGSB 2 noninterlocking rings at right angles
5RINGS 5 interlocking rings
6RINGS 6 interlocking rings
LSHAPE An L-shaped figure

The fastest way to get a plot is to start VIEWPLOT and plot one of these predefined objects thus:

```
.cop = [3111,535]status.vpt
.run vpltr[3111,535]
[VIEWPLOT VERSION OF 2-DECEMBER-76]
*plot icosa
READY TO PLOT

(An icosahedron is plotted now)
*end
```

NOTE

Whenever I show indented examples of typing to and from VIEWPLOT, I type in lowercase and the computer responds in UPPERCASE.

These predefined objects are no different from any objects you create. That is you can change and delete them. You can find their size and location with the OBJECT command and change it with the BOX and other commands. You can always get them back by copying [3111,535]STATUS.VPT and
running VIEWPLOT again.

You can also get these objects by copying and reading command files VLIB.DAT, VLIB2.DAT & VLIB3.DAT on [3111,535]. VLIB has the most used objects; VLIB2 the lesser used ones. VLIB3 builds up big objects such as arrays of cubes from 1 cube.

9.0 STATUS.VPT

VIEWPLOT has an associated file, STATUS.VPT, that contains 2 things: 1) the current values of all parameters, such as the position of the viewpoint and whether plots should have inverted perspective, and 2) all the existing objects. If you end VIEWPLOT normally, and save STATUS.VPT, VIEWPLOT will read it when you run it next. Thus you continue with all the objects and parameter settings you left off with. As far as the program is concerned, the only difference is that any files you were writing were closed and new copies are now being written. Of course STATUS.VPT can be renamed and copied like any other file. This in fact is how the library objects are implemented. I ran VIEWPLOT with a null status file and read in the library objects from files VLIB.DAT & VLIB2.DAT on [3111,535]. When I ended, I had a status file with the objects predefined that anyone can copy to his disk space and use.

If VIEWPLOT bombs unexpectedly, STATUS.VPT is probably OK but may be garbage. Type START (at monitor level) and if VIEWPLOT prompts for a command you can continue with the objects and parameter settings you had when if bombed. However if it bombs again immediately, STATUS.VPT is bad and you must get a clean copy (say from [3111,535]). If you end VIEWPLOT with a °C instead of the END command, STATUS.VPT will be bad.

VIEWPLOT uses STATUS.VPT for temporary storage during plotting, enlarging it whenever necessary, but never shrinking it. Thus after the END command, it is probably bigger than it need be. If you wish to create your own permanent STATUS.VPT and disk space is a problem, this is how to create a smaller STATUS.VPT: Run VIEWPLOT with the
old STATUS.VPT. Write all the objects you wish to keep to an ASCII file with the WRITE command. End VIEWPLOT with the END command. Create a null STATUS.VPT. Run VIEWPLOT again. Read the ASCII file with the CFILE command. Set the parameters as you want. End VIEWPLOT without doing any copies, combines or plots. STATUS.VPT is now small. You may want to keep a copy under another name.

Even if you have your own STATUS.VPT, you should keep a copy of important objects in an ASCII file since the format of STATUS.VPT will change as VIEWPLOT is changed and old versions may not work.

If you want to run VIEWPLOT but lack a status file, you can make a null one by

.copy status.vpt=nul:a

or by

.make status.vpt
HTECO.22.6P
*ex$$

VIEWPLOT will run with default parameters and no predefined objects.
10.0 COMMAND FILES

VIEWPLOT accepts command files. Thus for example you can have a command file that makes a cube of 8 copies of an object and apply it to any object you want. Very big objects take a lot of space (the 64 cube array takes 150 disk blocks in STATUS.WPT and even more in an ASCII file) so it is better to keep a file of the commands you used to create them and then read it as command file. The input in a command file is just what you would type in directly - you have to anticipate and include the answers to all questions that will follow the commands. If you leave out a line, things will get out of sync and for a while there will be general chaos. The file can contain any VIEWPLOT commands. However a CFILE command in a command file will cause the rest of that file to be ignored as VIEWPLOT switches to the new one. There is no push-down stack with command files - at least yet.

In fact the only way to read data in from a file is to use the READ command in a command file followed by the data.
11.0 MAXIMUM SIZES

100 objects are allowed at once. Each object can have up to about 2000 edges. The exact maximum depends on the number of vertices, edges, faces and the sum of the number of vertices in each face. There is no restriction on the total size of all the objects since they are stored on disk when not in use. Further an object has no maximum size when it is created. Only as much storage is used as necessary and if you read in a bigger object or combine another onto this, more space is allocated. If you try to put data (by reading, copying or combining) into a nonexistent object, it will be created.

Objects near the maximum may be creatable but not plottable since plotting uses an unpredictable amount of extra space.

If any of the maximum sizes are too small for you, let me know and I'll try to install a bigger version of VIEWPLOT. The problem is that the bigger the limits the more core it uses when running.
12.0 HOW TO USE VIEWPLOT

1. Log on at a TEKTRONIX terminal.

2. Copy file STATUS.VPT from [3111,535] to your own disk area if you don't have one of your own by typing:

   .copy= status.vpt[3111,535]

3. If you don't want to use a predefined object, make one or more files (using TECO, say) defining your new object(s).

4. Set the terminal states by typing:

   .tty form
   .tty no crlf

5. Run VIEWPLOT by typing:

   .run vplt[3111,535]

6. In VIEWPLOT, read some objects from your data files, if you want.

7. Modify them by shifting, combining, etc. if you want.

8. Finally, plot an object.

9. Repeat the reading, modifying and plotting steps (leaving any of these out if you wish) as many times as you want.

10. When done, type an END command.

11. Delete all the temporary files VIEWPLOT created (named FOR27.DAT, FOR28.DAT and FOR29.DAT).

12. Save STATUS.VPT if you want to keep the objects and parameter settings.

13. Log off the terminal - don't just turn it off.
13.0 COMMAND SYNTAX

Most commands prompt for extra info. It may be:

1. A simple yes/no answer. If so, Y or T means yes while N or F means no. A blank or immediate carriage return means don’t change the parameter. Anything else is illegal. Leading blanks are not allowed.

2. (for the BOX command) 3 yes/no answers in 3 consecutive characters on 1 line such as YYN. Leading blanks are not allowed.

3. The name of an object. Type the object name — any 8 characters without blanks, followed by a carriage return. Leading blanks are not allowed. If you type nothing but a carriage return, the last object named is used again.

4. A file name. This is 1-5 characters; the compulsory extension is DAT. Leading blanks are not allowed.

5. One or more numbers. There are no fixed field sizes: use a blank, tab or comma to separate them. Extra blanks before, between and after numbers are allowed.

6. A location which is 3 real numbers on 1 line representing Cartesian coordinates. Extra blanks before, between and after numbers are allowed.

For most of the command names, only the first 4 characters are used. The exceptions are where this would be ambiguous such as ABUT2 which you must type completely to distinguish it from ABUT.

You must type command names in uppercase if the terminal has both cases — for example on the 4014 set the tty lock switch. For object names, upper and lower case are distinguished: ‘A’ and ‘a’ are 2 different objects.

The commands are:

13.1 ABUT

Questions: 1) The object to abut onto
2) The object to move against it
3) The side of the first object to move against: one of XL, XH, YL, YH, ZL or ZH
4) How the second object is to be centred on the first: 0, 11, 12, 13, 14, 15, 21-25, ... 51-55
Move the second object up against the first and then add it onto the first. Actually the second is moved until boxes around the objects touch, but usually there is little difference. The second is moved against 1 of the 6 sides of the first's box: XH means the higher of the 2 sides with equation \( X = \text{CONSTANT}, \ YL \text{ means the lower } Y \text{ side, etc.}

Assume you have chosen side XH. Object 2 is shifted in the X direction until its lowest X coordinate equals the highest X coordinate of object 1. Now it must be centred on object 1 in the Y and Z directions. A centring code of 0 causes no centring. Else the code is 2 digits, each 1-5. The first controls centring in the Y direction, the second in the Z thus: The big square is object 1 and the little squares possible positions for object 2.

If you are centring on the Y plane, the other 2 dimensions are Z and X respectively, and for Z, X and Y. Code 1 always means low in that direction and 5 high. Thus for centring on the XL, YL or ZL planes, the above diagram should be mirror reversed. Code 3 causes object 2 to be centred on object 1 in that dimension. Codes 1, 2, 4 & 5 cause object 2 to be aligned against the inside (2 & 4) or outside (1 & 5) of object 1.

After object 2 is added to 1, it is moved back to its former position.
13.2 ABUT2

Questions: (The same as for ABUT)

ABUT2 acts almost the same as ABUT. However the location of object 2 is permanently changed and it is not added to 1 which is not altered. One use is to abut several objects in different locations of the same side of 1 without changing 1's box by combining them with it. Combine them all at the end with the COMBINE command. ABUT2 also allows you to move object 2 back a little before adding it.

13.3 BOX

Questions: 1) An object
   2) The new centre of its box.
   3) The new size of its box in each direction (3 real numbers).
   4) Which dimensions are to remain in proportion? (3 yes/no answers on 1 line).

Expand or contract an object in the X, Y and Z directions to fit a box with a given centre and size. Those of the 3 dimensions for which you reply Y or T remain in proportion to each other, the rest are expanded to touch the sides of the box. Thus to keep the X and Y dimensions in proportion to each other but expand the Z dimensions as much as possible independently, type YYN.

13.4 CENTERPOINT

Question: The new location.
Default: (0,0,0)

Change the location of the centre of the perspective plane. This should generally be inside or near the object.

13.5 CFILE

Question: File name.

Read commands from a file as if you were typing them at the terminal and execute them. They are listed as read except for data for the READ command since it is so long.

At the end, control returns to your terminal. This end of file must occur between commands and not between a command and its questions. For commands that have
questions, follow each command, on a new line, by the answers. The command file can itself contain a CFILE command: This switches to the new file and the rest of the old file is never read. VIEWPLOT makes no attempt to correct for errors while reading the command file; it blithely continues reading regardless.

The most important use of command files is to read in objects since the READ command expects data from the terminal. Here you put the READ command and its data in a command file.

For examples, look at file VLIB3.DAT[3111,535].

13.6 COMBINE

Questions: 2 objects

Combine the 2 objects to make a new object replacing the second. If the second doesn't exist, instead create it containing a copy of the first. It doesn't matter that the second object is now bigger, provided it stays under the max size for objects.

13.7 COMMENT

The rest of the line following 'COMMENT' is printed and nothing is done but read the next command. This is useful for annotating command files.

13.8 COPY

Questions: 2 objects

The first object is copied onto the second, deleting its former contents. If the second object doesn't exist it is created. The second object may be bigger or smaller. If it doesn't exist, it is created.

13.9 DEBUG

Questions: 1) A routine name

2) a debugging code for it
3) Another routine name

4) Its debugging code

5), 6) ... Defaults: 0 for the debugging codes for all routines.

This causes the given routines to start or stop printing info as they execute and is used to debug VIEWPLOT. Routines are prompted for until you just type \(<CR>\). For the code, 0 means print nothing, 1 means print a little as the routine executes and 2 means print more.

13.10 DELETE

Question: An object

Delete the object. Its data is lost unless it has been copied or combined to another object.

13.11 DIR

List the names of the current objects.

13.12 DIR2

Print a directory of all existing objects giving their names and various internal info. At any time, each object is either in core or on disk. (This is transparent to you: objects are swapped in when they are needed and out when space is needed). For the objects in core further info is listed in a table. Important variables are

- The object's number which gives its name in the first list,
- NV The number of vertices
- NE The number of edges
- NE1 The number of type 1 edges
- NF The number of faces
- NF1 The number of type 1 faces
- NLF The total number of vertices in all the faces
- WORDS The number of words of storage the object takes.
13.13 END

Exit from VIEWPLOT to the monitor. If you ^C instead of this, files being written will not be closed and STATUS.VPT will be useless.

13.14 GRID

Question: Yes/no
Default: No

Should all plots have a 10 by 10 grid superimposed?

13.15 HELP

List all the command names.

13.16 INVERT

Question: Yes/no
Default: No

Should inverted perspective be used for future plots? This means that distant objects appear bigger than close ones. Lines of perspective will converge near you instead of on the horizon.

13.17 LOCATION

Questions: 1) Size of the plot on the screen (1 real)
2) lower left corner of the plot (2 coords)

This alters the size and location of the plot as displayed on the screen. The size and location assume the screen is 10 wide by 7.56 high. The plot corner is relative to the lower left corner of the screen. This defines a square window that the plot fills. Thus corner = (2.5,1.3) and size = 5 makes the plot half the width of the screen and centred in it.
13.18 MORE

List the commands each with a one line description.

13.19 NUMBER

Question: Yes/no
Default: No

Should the vertices on the plot be numbered? This is useful for debugging an object.

13.20 OBJECT

Question: An object

For the given object list: the numbers of its vertices, edges (each type), faces (each type), total vertices in the faces, the size in words and its bounds (low and high X, Y and Z).

13.21 OBJECT2

Question: An object

list detailed info about the object, useful if you are doing hand calculations. For each edge give the length and direction vector. If it is too short, give the direction as (0,0,0). This is not ambiguous since the length of the direction vector normally 1. For each face, give the equation. If you wish more detailed data, or want this data written to a file so one of your own programs can use it, see me.

13.22 PARAMS

List the values of all user settable parameters such as the viewpoint and whether a grid is to be drawn. Also list the unit numbers and internal parameters.
13.23 PERT

Question: An object

This does a perspective projection on the object. It is transformed so that it will look from viewpoint = \( (0,0,0), \) infinity \( \) and centerpoint = \( (0,0,0) \) the way it now looks from the current viewpoint and centerpoint. Thus parallel lines will converge to a point, etc. If you then plot the object, it will be transformed again for the plot, using the values of viewpoint and centerpoint in effect at that time.

13.24 PLOT

Question: An object

Calculate the hidden lines etc. and plot the given object. If the STATS switch is on, various statistics will be printed as the plot is calculated. When VIEWPLOT types \( \text{READY TO PLOT} \), type a carriage return to get the plot. A bell indicates that the plot is finished. You can now get a hard copy. The plot will remain until you type another carriage return. Then VIEWPLOT will prompt for another command. You should not leave the same plot on the screen for longer than 15 minutes at a time.

13.25 PRINT

Question: An object

List the given object’s data on the screen, nicely formatted.

13.26 READ

Questions: 1) An object
2) MV, ME, MF, MLF
3) The object’s data

Read the given object. MV, ME, MF & MLF are upper bounds on the number of vertices, edges, faces and total vertices in the faces. Giving these too high (below the maximum size) does nothing but slow VIEWPLOT down a little as it reads the object. Thereafter the object only gets as much space as it needs. If you omit ME, MF and MLF, reasonable values calculated from MV are used. If these values are too small, the first extra vertex, edge or face
causes an error. The object is deleted but VIEWPLOT continues.

If you type the command at the terminal, VIEWPLOT will prompt you for each vertex, edge and face. If this command appears in a command file, the object data will not be listed - you can always use the PRINT command later.

As for the data, there are no fixed columns; input is free format. First give the vertices: on each line the coordinates of 1 vertex, followed by an optional positive integer identifier. If this id is omitted for the I-th vertex, #I is used. Follow the last vertex with a line with just ‘9999’. There must be fewer than MV vertices in all. Then give the edges, 1 per line. For each, give the ids of the 2 vertices at its ends. Give all the type 1 edges, followed by a line with just ‘9998’, followed by all the type 2 edges and ended by a line with just ‘9999’. If there are no type 2 edges, the 9998 but not the 9999 may be omitted. However if there are no type 1 edges, the 9998 must still precede the type 2 edges. Finally give the faces, 1 per line. For each face, give the ids of the vertices at its corners, in order. Each face can have up to 20 vertices. Give the type 1 faces, followed by ‘9998’, followed by the type 2 faces, followed by ‘9999’. The type 2 faces must be oriented positively (anti-clockwise) as seen from the outside. Remember that type 2 faces belong to a solid with a well-defined inside and outside.

This is how a tetrahedron is read in:

```
read
tetra
4  6  4  12
1.000 1.000 1.000 1.000
1.000 -1.000 -1.000 -1.000
-1.000 1.000 -1.000 1.000
-1.000 -1.000 1.000 1.000
9999.  9999.  9999.
1  2
1  3
1  4
2  3
2  4
3  4
9999  9999
9998  9998
1  3  2
1  2  4
1  4  3
2  3  4
9999  9999
```
You can get more examples by writing objects to a file and looking at it.

13.27 ROTATE

Questions: 1) An object
   2) The angle of rotation, in degrees
   3) A point on the axis of rotation
   4) Another point on that axis

   Rotate the object about an axis through the 2 points. The rotation is right-handed looking from the first to the second.

13.28 SCALE

Questions: 1) An object
   2) The fixed point
   3) The amount to stretch in the X, Y and Z directions.

   Change the object's size on each direction by the given factor. The stretching is about the fixed point. You can reflect the object by making one factor negative. Don't do this if the object has any type 2 faces for then they will be oriented the wrong way.

13.29 SHIFT

Questions: 1) An object
   2) The amount to shift in X, Y & Z.

   Shift or translate the object by adding the given amounts to each coordinate.

13.30 STATS

Questions: 1) Yes/no
   2) A unit number if the first answer is yes.

   Should statistics be printed as VIEWPLOT calculates? They are useful for debugging VIEWPLOT and also to see how things are progressing for big, slow plots. They are printed on the given unit (5 for the terminal).
13.31 STELLATE

Questions: 1) An object
2) A real number, say R.

Create a pyramid on the outside (or inside) of each type 2 face of the object. R determines the height of each pyramid. If R=1 and the pyramid's base is a regular polygon, then the height is such that the slant edges are as long as the base edges. If the base is not a regular polygon, the height is such that the root mean squared slant edge length is equal to the root mean squared base edge length. If R=2, the height is twice that, etc. If R=-0.5, the pyramid is half that high and points inward, not out. If R is big (say >10) you will hardly be able to see the original object for the pyramids. If the object is not one solid, but say an array of cubes, it is your responsibility to see that the pyramids are not so high that they intersect. If so, the plot will be wrong even though there will probably be no error message. This is also a problem if R is negative; if R<1, the inward pointing pyramids will intersect and if R is about -1, they might.

13.32 TIME

Questions: 1) Yes/no
2) If yes, a routine tracing code (0 or 1).

Should frequency and timing stats be kept for each subroutine? If so, they are listed on the same unit as given in the STATS command. If the tracing code is 1, all subroutine calls and returns are listed.

13.33 TRANSFORM

Questions: 1) An object
2) A 4 by 3 real matrix, 4 elements per line.

Transform the given object. If the matrix is T, for each vertex X, its new coordinates, Y, are (for i=1,2,3):

\[ Y = X \cdot T + X \cdot T + X \cdot T + T \]
\[ I \quad 1 \quad 1I \quad 2 \quad 2I \quad 3 \quad 3I \quad 4I \]

Since this preserves straight lines, the edges and faces are still defined.

Examples: The following matrix is the identity and doesn't alter the object:
This matrix:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
\end{bmatrix}
\]

does the transformation:

\[
X <- X + Y \\
Y <- Y \\
Z <- Z \\
\]

Thus it changes a square in the XY plane into a rhombus. The `TRANSFORM` command does anything the `ROTATE`, `SHIFT` and `SCALE` commands do, but not as conveniently.

13.34 VIEWPOINT

Question: A point

Relocate the viewpoint, the location of the imaginary observer.

13.35 WRITE

Questions: 1) An object
            2) A file name
            3) Any number of lines of comments

Write the given object to the file, formatted so that a `READ` command can read it. If the file name is blank, this object is added to the same file as the last `WRITE` command. The file is written as a command file with comment lines, a `READ` command and the data. You are prompted for comment lines until you type just a carriage return. They are useful for identifying the object. Each comment line can be up to 72 characters long. The `WRITE` command is a safer way to store objects than saving STATUS.VPT because it is in ASCII and because the format of STATUS.VPT may change with different versions of VIEWPLOT. You must end VIEWPLOT with an `END` command if the file being written is to be preserved.

There can be only 1 open WRITE file at a time. Any time you give an explicit file name in this command, the old file is closed (if one is open) and a new one opened. If a file already exists with this name, it is deleted. Thus if
you give the same file name twice, the first version is lost.
14.0 EXAMPLE

The following protocol from login to logout shows a few VIEWPLOT commands:

.log 6001,543
JOB 1 Harvard 5.07B-43 TTY40
Other jobs same PPN
2350 05-Dec-76 Sun

System down Thursdays 0800-1200 for preventive maintenance.

/cop=[3111,535]status.vpt
.run vplt[3111,535]

STARTING VIEWPLOT, 5-Dec-76 23:50, (VERSION OF 28 NOVEMBER 76)
HAVE YOU SET TTY FORM AND NO CRLF?
LIST FILE NEWS.VPT FOR NEW FEATURES (28-NOV-76).
RESTORING OLD VPLT STATUS.
*DIR2

NAMES OF EXISTING OBJECTS:

1  RING   -1  60  118  -1
2  TETRA  -1  24  42   -1
3  OCTA   -1  30  60  -1
4  CUBE   -1  36  66  -1
5  DODECA -1  72 138  -1
6  ICOSA  -1  48 114  -1
7  THINRING -1  60 118  -1
8  6RINGS -1  300 618  -1
9  2RINGS -1 108 218  -1
10 2RINGSB -1 108 218  -1
11 5RINGS -1 252 518  -1
12  AXES  -1  24  36  -1
13  LSHAPE -1 102 198  -1
14  GHD1   -1  48 102  -1
15  GHD2   -1  48 102  -1
16  GHD3   -1  48 102  -1

OBJECTS NOW IN CORE: ( 3 OF 20000 WORDS USED)
# MV NV SV ME NE NEL SE MF MLF NF NF1 NLF WORDS
*PRIN
OBJECT? CUBE
PRINT OBJECT, TITLE=CUBE
VERTICES (LOC IN I/RSP= 3 # 4 SIZE= 30 WORDS.
  INT BASE= 3), # OF VERTICES= 8 # OF TYPE 1= 8 MAX #= 8
1: 1.0000 -1.0000  1.0000 2: 1.0000  1.0000 1.0000
3: 1.0000  1.0000 -1.0000 4: 1.0000 -1.0000 -1.0000
5: -1.0000 -1.0000  1.0000 6: -1.0000  1.0000 1.0000
EXAMPLE

7: -1.0000 1.0000 -1.0000 8: -1.0000 -1.0000 -1.0000
EDGES  (LOC IN I/RSP= 33 # .4 SIZE= 30 WORDS.
INT BASE= 4), # OF EDGES = 12 # OF TYPE 1= 12 MAX #= 12
1) 1 2(2) 2 3(3) 3 4(4) 4 1 5(5) 5 6(6) 6 7(7) 7 8(8) 8 9(9) 9 1
10) 2 6(11) 3 7(12) 4 8(8)

FACES  (LOC IN I/RSP= 63 # 4 SIZE= 37 WORDS.
INT BASE= 6), # OF FACES = 6 # OF TYPE 1= 0 MAX #=

# BASE NVRT  VERTICES
1:13 4: 1 2 3 4
2:17 4: 1 4 8 5
3:21 4: 1 5 6 2
4:25 4: 2 6 7 3
5:29 4: 3 7 8 4
6:33 4: 5 8 7 6

*OBJE

OBJECT? CUBE

THE OBJECT CUBE HAS:
8 VERTICES,
12 EDGES, THE FIRST 12 BEING TYPE 1,
6 FACES, THE FIRST 0 BEING TYPE 1,
24 VERTICES IN Faces AND
97 WORDS STORAGE.
ITS BOUNDS ARE
X: -1.0000 TO 1.0000
Y: -1.0000 TO 1.0000
Z: -1.0000 TO 1.0000

*BOX

OBJECT?

NEW CENTRE OF OBJECT? 0 0 0
NEW SIZE IN X, Y AND Z DIRECTIONS? 1 1 1
FOR WHICH OF THE 3 DIMENSIONS ARE, DISTANCES TO REMAIN
IN PROPORTION? (TYPE 3 LOGICAL VARIABLES) TTT

*PRIN

OBJECT?

PRINT OBJECT, TITLE=CUBE

VERTICES  (LOC IN I/RSP= 3 4 SIZE= 30 WORDS.
INT BASE= 3), # OF VERTICES= 8 # OF TYPE 1= 8 MAX #= 8
1: 0.5000 -0.5000 0.5000 2: 0.5000 0.5000 0.5000
3: 0.5000 -0.5000 0.5000 4: 0.5000 0.5000 -0.5000
5: -0.5000 0.5000 -0.5000 6: 0.5000 -0.5000 -0.5000
7: 0.5000 0.5000 -0.5000 8: 0.5000 0.5000 0.5000

EDGES  (LOC IN I/RSP= 33 # 4 SIZE= 30 WORDS.
INT BASE= 4), # OF EDGES = 12 # OF TYPE 1= 12 MAX #=
1) 1 2(2) 2 3(3) 3 4(4) 4 1 5(5) 5 6(6) 6 7(7) 7 8(8) 8 9(9) 9 1
10) 2 6(11) 3 7(12) 4 8(8)

FACES  (LOC IN I/RSP= 63 # 4 SIZE= 37 WORDS.
INT BASE= 6), # OF FACES = 6 # OF TYPE 1= 0 MAX #=

# BASE NVRT  VERTICES
1:13 4: 1 2 3 4
2:17 4: 1 4 8 5
3:21 4: 1 5 6 2
VIEWPLOT Users' Manual

EXAMPLE

4:25 4: 2 6 7 3
5:29 4: 3 7 8 4
6:33 4: 5 8 7 6
*COPY
OBJECT TO GO FROM?
OBJECT TO GO TO? A

*ROTATE
OBJECT?
ANGLE OF ROTATION IN DEGREES? 45
ONE POINT IN AXIS?
ANOTHER POINT ON THE AXIS? 0 0 1

*SHIFT
OBJECT?
AMOUNT TO SHIFT? 1 1 1

*ADDON
OBJECT TO GO FROM? CUBE
OBJECT TO GO TO? A

*[

%ILLEGAL COMMAND

*BOX
OBJECT? ICOSA
NEW CENTRE OF OBJECT?
NEW SIZE IN X, Y AND Z DIRECTIONS? 1 1 1
FOR WHICH OF THE 3 DIMENSIONS ARE DISTANCES TO REMAIN IN PROPORTION? (TYPE 3 LOGICAL VARIABLES) YYY

*OBJE
OBJECT? A

THE OBJECT A HAS:
16 VERTESES,
24 EDGES, THE FIRST 24 BEING TYPE 1,
12 FACES, THE FIRST 0 BEING TYPE 1,
48 VERTESES IN FACES AND
175 WORDS STORAGE.
TS BOUNDS ARE
::  -0.5000 TO 1.7071
::  -0.5000 TO 1.7071
::  -0.5000 TO 1.5000

SHIFT
OBJECT?
MOUNT TO SHIFT? 1 1 -2.5

*COMBINE
OBJECT TO GO FROM? ICOSA
OBJECT TO GO TO? A
*PKD

EXIT FROM VPLT

CPU TIME: 13.47 ELAPSED TIME: 5:55.88
NO EXECUTION ERRORS DETECTED

EXIT

 files deleted:
 FOR28.DAT
 FOR29.DAT
 STATUS.VPT
 104 Blocks freed

 .k/f

 Other jobs same PPN
 Job 1, User [6001,543] Logged off TTY40  2357  5-Dec-76
 Saved all 16 files (950. Disk blocks)
 Another job still logged in under [6001,543]
 Runtime 0 Min, 16.30 Sec
*PLOT
OBJECT?

READY TO PLOT.

*END

EXIT FROM VPLT

CPU TIME: 13.47 ELAPSED TIME: 5:55.88
NO EXECUTION ERRORS DETECTED

EXIT

.del for28.dat,for29.dat,status.vpt
Files deleted:
FOR28.DAT
FOR29.DAT
STATUS.VPT
104 Blocks freed

.k/f
Other jobs same PPN
Job 1, User [6001,543] Logged off TTY40 2357 5-Dec-76
Saved all 16 files (950. Disk blocks)
Another job still logged in under [6001,543]
Runtime 0 Min, 16.30 Sec
15.0 RUNNING TIME AND SIZE

I plotted an object without shading composed of a cubic array of 216 cubes that had 2592 edges in under 4 minutes. This was using a poor compiler on a slow computer (the F40 compiler on the Aiken PDP-10). A smaller object, an array of 64 cubes with 768 edges took 1:10, while a very small object, an array of 8 cubes with 96 edges, plotted in 10 seconds. The algorithm uses locality in that under reasonable assumptions as to how the average edge length gets smaller as the number of edges get greater, the CPU time used depends on NE**1.25 where NE is the number of edges. In the 3 examples above the time rose even slower than that, in fact not much faster than linear. On the other hand, if the object is a set of pick-up sticks, all interlaced with the edges all going from one side of the screen to the other then that would take a lot longer because each edge would intersect most of the other edges. Since the algorithm depends on knowing which edges intersect other edges (in 2-D projection) short edges take less time to calculate and plot. Plotting with shading will probably take twice as much time as without.

Currently VIEWPLOT takes about 58K of core on a PDP-10. 30K of this is for the code and could be reduced about 20K by overlaying. Another 20K is for an array to hold all temporary data during plotting. Its size depends on the how big an object you want to be able to plot.
16.0 ERRORS

VIEWPLOT has different types of errors. Some occur when you try to do something illegal such copy from a nonexistent object. After these it prints a message, doesn't execute the command and prompts for the next command. If the error is something like trying to create an object that is too large, not only is the new version of the object not created but the old version, if there was one, is deleted.

Others are due to internal VIEWPLOT errors. These probably print an error number and return to monitor level. You can try to restart VIEWPLOT immediately by typing 'START'. If VIEWPLOT prompts for a command, you have succeeded and probably have all your objects and parameter settings as before. But if VIEWPLOT bombs, STATUS.VPT is bad and you must get a clean copy.

Some user errors such as trying to read a nonexistent file also trap to monitor level. Try restarting as mentioned above.

If an object has errors, the only way to correct it at this point is to write it out and TECO it. Eventually there will be an interactive editor.

17.0 NEWS.VPT

There is a file NEWS.VPT[3111,535] that has additions and changes to VIEWPLOT since this manual.
18.0 HOW TO GET HELP

In case of problems, contact:

Wm. Randolph Franklin
Conant Hall #11,
36 Oxford St.,
Cambridge, Ma, 02138
(617) 498-4207

or

Laboratory for Computer Graphics and Spatial Analysis
Gund Hall 520,
Harvard University,
CAMBRIDGE, Mass, 02138
(617) 495-2526

or

send ARPANET mail to FRANKLIN@HARV-10.

What is MAIL?

This is a way to send messages from one user to another on the system. For detailed info, type 'HELP MAIL' at monitor level. Briefly, at monitor level, type 'MAIL FRANKLIN'. When prompted for a subject, type 'VIEWPLOT'. Then type your message, comment or criticism on as many lines as needed. End it with an escape character.

Please send me errors and suggestions for improving VIEWPLOT or this manual.
VIEWPLOT

Program Logic Manual

Wm. Randolph Franklin

DATE: 12-jul-76 3:52:08
### 1.0 TABLE OF CONTENTS

1. Table of contents
2. Routine summary
3. Routine descriptions
4. Storage allocation in core
5. Object data format
6. Units
7. Varda records
8. Sblock units
9. Conventions
10. Dec fortran dependencies
11. Overlaying
12. New variable descriptions
13. Long term suggestions for improvement
14. Old variable descriptions
15. Old storage allocation in ala
16. Calling tree
17. Routine cross reference
**2.0 ROUTINE SUMMARY**

<table>
<thead>
<tr>
<th>number</th>
<th>name</th>
<th>arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>main</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>block data</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>addon (vf, ef, ff, vt, et, ft, ier)</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>addon2(name1, name2, ier)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ala (kv, ke, kf)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>alc (kres, kob, pt)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>alloc (name, kv, nv, ke, ne, kf, nf, l1f)</td>
<td>fn</td>
</tr>
<tr>
<td>7</td>
<td>areaav (kv, kp, n)</td>
<td>fn</td>
</tr>
<tr>
<td>60</td>
<td>bepair(kv, ke, kb, nbep, ichan)</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>bfpair(kv, sv, kf, sf, kbf, ichan, nfb)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>bound (kv, b, key)</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>box (kv, cent, siz, keep, actual)</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>clipl (kv, ke, kf, ceg)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>clump (kv, ke, kf, kk, ndim)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>copsen(vf, ef, ff, vt, et, ft, ier)</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>cortim</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>cross (kv, na, nb, lyes, xsect)</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>dattim(dt)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dacos(x)</td>
<td>dble prec fn</td>
</tr>
<tr>
<td>12</td>
<td>draw (ichan)</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>dumpio(locno, len, seq, kode)</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>dump2 (blocno, base, len, lwrite)</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>edgfac(ke, kf)</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>edit</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>eepair(kv, ke, ichan, se)</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>es (kv, kv2, pt)</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>exchan(base, len1, len2)</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>facqn (kv, kf)</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>fndedg(ke, na, nb)</td>
<td>int fn</td>
</tr>
<tr>
<td>62</td>
<td>getobj(na, nb, kob, extv, exte, extf, extlf, keep, alloc)</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>gridit</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>groups(kk)</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>hatch (kv, np, darkns, slope)</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>htsin (kv, numcom, pt)</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>ibiners(base, n, elt) fn</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>idmat (a)</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>indata(kv, ke, kf, numcom, pt, ier)</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>inters(kv, kf, kbf, nfb)</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>invper(kk, n)</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>ipack (1h, rh) fn</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>ipnpol(kv, kp, n, z)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>irunti(dummy) macro fn</td>
<td></td>
</tr>
</tbody>
</table>
VIEWPLOT PROGRAM LOGIC MANUAL
ROUTINE SUMMARY

land (i, j) fn
lor (i, j) fn
lsh (iword, iamt) fn
matmul(a, b, ab)
movesp(frbase, tobase, len)
nedfac(kf)
obset(kv, jv, mv, nv, sv, ke, je, me, ne, nel, se, kf, jf, mf, mlf, nf, nlf, sf)
onface(kv, kf, z) int fn
outsen(kv, ke, kf, ititle)
pert (kv1, kv2)
plagra(kv, ke, kk, pt, ier)
prisin(kv, ke, kf, numcom, pt, ier)
ptin(kv, n, z, ier)
punch(kv, ke, kf, name)
putobj(kob)
sblock(rec, kode, loc, icahn)
sblock2(rec, kode, loc, icahn)
show(kv, kf, kbf, z, lyes, nfb)
soldel(ke, kf, kk)
srtedg(ke, kk)
ostsen(kv, ke, kf, kvis, khid, lsto, ier)
swed(ke, na, nb)
tessel(kv, ke, kf, ier)
times(name) totmac(tland, tlor, tlash, tipack, tunp)
trans(kv, tra, px, qx, ard, key)
unpack(lhrh, lh, rh)
varda(block, len, recno, kode)
vccross(x, y, z)
veclen(v) fn
verr(i)
yntf(ll, l2, l3, n)
zedqn(kv, ke)

Obsolete Routines:
dataset(array, nall, nuse, unit, name)
edgnn(kv, ke, key)
efbou(kv, ke, kf, key, kk, ll)
srtver(kv, ke, kf, kk)
3.0 ROUTINE DESCRIPTIONS

3.1 MAIN

If this is a new core image, set all i/o units (except 5) to device DSK:. In any case, zero routine times and negate HIBLOC so VARDA will open its file again.

Then alternately call ALC to read and execute commands (except plotting) and ALA to calculate hidden lines and plot. ALC returns a code saying whether ALA should be called next or MAIN should return to the monitor.

3.2 ALC(KRES, KOB, PT)

On entry, assume that ISP is empty (i.e. that all objects have been swapped out). If LINIT = TRUE which means this is the first call to ALC, try to use file STATUS.VPT to restore the objects and parameters. If VARDA returns LOLDDA = FALSE, the file was the wrong format (probably a null file), so start with default parameters (as defined in BLOCK DATA) and no objects. Else read in common COM1 which has all the parameters and the object headers. VARDA also reads a few variables to init itself. If the variable COMLEN (length of common) read in is different, stop since the common read is out of date but by now the default common has been smashed.

Read commands and execute all of them except plotting. Call routines for many commands. PT sets the verbosity of the messages. On a PLOT command, swap out all the objects, leaving a copy of the object to be plotted in core. It is #KOB. The flag KRES is set and control returns. On an END command, timing stats are printed if desired. Set KRES to another value and return.

COPYING OBJECTS: Insure that the 'from' object is in core and also up to date on disk. Put the 'to' object's name in array OBJ if it doesn't yet exist. Then make it point to the existing 'from' object in core and mark the latter object as not in core.
3.3 ALA(KV, KE, KF)

Do preliminary calculations for and plot object based at (KV, KE, KF): Call ES to rotate and scale object so that VIEWPOINT= (0, 0, 0) and CENTERPT= (0, 0, 1). Call PERT to do the perspective transformation. Call BOX to make the object fit a 10 x 10 window. The above 3 transformations are also done on an object consisting of only the 3 axes. This object provides a reference point so that the cross-hatch shading can be parallel to the original axes.

Allocate a temporary array, KK, of NV elements. Call CLUMP to cluster close vertices and to delete unused vertices and duplicate vertices and edges. The unused space is left at the end of KV and KE for now. Free KK. Allocate space for the face equations, based at KFQ, and for a temporary array of NE elts, based at KK. Call EDGFAC to add the edges on each face to the face array. Call FACQN to calculate the face equations. Call SOLDEL to delete the back facing type 2 faces and edges not on a remaining face. Call NEDFAC to delete the edge numbers from the face array.

Copy KV, KE, KF, KFQ to disk. Call EEPAIR to produce a file of (E, E) pairs. Mark ISP as empty. Call BFPAIR to calculate (B,F) pairs and leave them, KV, KF and KFQ in core. Call INTERS to read the file of (B,E) pairs and to leave a file of visible segments. Call DRAW to plot them.

3.4 Addon(vf, Ef, Ff, Vt, Ee, Ft, Ier)

Concatenate the object defined by (VF, EF, FF) onto the end of the object defined by (VT, EE, FT). The latter must be big enough for both. Return IER=0 if there was no error, -1 if there was.

3.5 ADDON2(NAME1, NAME2, IER)

NAME1 and NAME2 are INT(2). Catenate the object named NAME1 onto NAME2. Return IER=0 if there are no errors; else return 1 to 3 depending on the error.
If there is room for object 1 and the new bigger object 2 in core together, call ADDON to do the job. Else get object 1 to find out how big it is. Then swap in the second, allowing extra space in its arrays for object 1. Finally read in object 1 in pieces into object 2 and correct its edges and faces.

3.6 ALLOC(NAME, KV, NV, KE, NE, KF, NF, NLF)

Allocate space for an object in ISP. If NV > 0, allocate an array of NV vertices and return KV as its base. Likewise for edges and faces. NAME is an id code that is stored in the arrays.

3.7 BEPAIR(KV, KE, KB, NBEP, ICHAN)

IN: KV, KE, ICHAN
OUT: NBEP, KB

Assume the first end of each edge is negative if the edge is interior. Assume ISP contains KV and KE but can be overwritten. Divide the screen into B by B square buckets where $B = \text{BUCKET} \times \text{NE}^{2/3}$. Initially, BUCKET = .5. Later it can be fine-tuned. Use a fudge factor so that an edge that passes very close to a bucket is considered to pass through the bucket. IF an edge is more horizontal than vertical, run along it horizontally. It can only cover 1 or 2 buckets vertically per horizontal bucket. If it is more vertical, do the reverse.

Produce a file on SBLOCK channel ICHAN of (B,E) pairs for all the buckets the edges defined by KV, KE pass through. Pack each pair into 1 word before writing not to save core but because 1) it means 1/2 the disk I/O and 2) sorting is much faster. Set NBEP to the number of pairs. Make all the edges' endpoints positive. Swap them out without destroying the core image. Then move them to the front of ISP, on top of the vertices. Then swap the (B,E) pairs in after the edges, set their base to KB, and sort them by bucket. Return with the edges and (B,E) pairs in core. Leave in core.
Important variables:
IBLO    Return 1-D bucket #(1-IB) for a real coord.
IBLO2   Return 2-D bucket #, for 2 1-D ones.
IB     # buckets across screen.
PER    Width of each bucket.

3.8 BFPAIR(KV, SV, KF, SF, KBF, ICHAN, NFB)

IN:  SV, SF, ICHAN
OUT: KV, KF, KBF, NBF

Assume that ISP is empty. Swap in the vertices and faces and set KV and KF. Divide the screen into C**2 buckets where C = FBUCK * NF**.333. Initially FBUCK = 1. Then determine which buckets each face partially or wholly covers. For now just the box around the face. Assume that by now the backward type 2 faces have either been deleted or their first vertex number zeroed. Write (B,F) pairs.

Then swap the pairs in and sort them by bucket number. Transform them from a sequence of (B,F) pairs to a sequence of NFB**2+1 dope pointers indexing into a following sequence of faces. Assuming an average of more than 1 face per bucket, this will take less space. Reset KBF since the array is now single dimensional.

Next swap in the face equations and set KFO. Eventually I plan to sort the faces in each bucket by the distance of the centre of each bucket projected on each face from the viewpoint. Return with the vertices, faces, face equations and (B,F) pairs in core.

3.9 BOUND(KV, B, KEY)

Find the coordinates of a box around the vertices based at KV and return in B which is REAL(KEY, 2). Use a 3-D box if KEY=3, 2-D if KEY=2. Although the vertices are single precision to save space, B is calculated and returned in double precision for increased accuracy.
3.10 BOX(KV, CENT, SIZ, KEEP, ACTUAL)

Transform the vertices based at KV so that they fit a box whose centre is CENT and size is SIZ. If the object does not touch all 6 sides of the box, KEEP controls whether some dimensions should be expanded more than others. ACTUAL returns the expansion factors actually used.

Those dimensions for which KEEP(I)= TRUE are kept in proper proportion to each other so that in general vertices touch the sides of the box for only 1 of these dimensions. The scale is adjusted independently for the other dimensions until vertices touch the opposite sides of the box in them.

3.11 CLUMP(KV, KE, KF, KK, NDIM)

Sort vertices and edges. If any vertices are very close in NDIM (=2 or 3) dimensions, change all refs to the second vertex to refer to the first. 2 vertices that are close in 2-D may be separated by 1 or more vertices that are close in X but distant in Y or Z. Delete duplicate vertices and edges and unused vertices. Delete edges with both ends the same. Compress out deleted elements (without decreasing the sizes of the arrays). KK bases a work array of NV elements.

3.12 Copsen(vf, Ef, Ff, Vt, Et, Ft)

Concatenate the object defined by (VF, EF, FF) onto (VT, ET, FF). The receiving object must be big enough.

3.13 CORTIM

Call TIMES 2000 times and set RTCORR to the average time used per call, in msec. Zero RNCALL.
3.14 CROSS (KV, NA(2), NB(2), LYES, XSECT(2))

Test whether the edge whose endpoints are in NA intersects the edge in NB and set LYES TRUE or FALSE accordingly. If yes, return the intersection point in XSECT. If the closest end of NB is farther than the farthest end of NA by at least TOLER, then they are defined not to intersect. Note that close to the viewpoint means high Z. However they do intersect if the end of one is on the other.

The method is that if NA and NB intersect, then the ends of NA must be on opposite sides of NB (extended if necessary) and vice-versa. CROSS is usually called several times with the same NA while NB varies. Thus some info on NA such as the edge's equation need not be recalculated. If NA(1) = -1, NA is assumed to be the same as before.

CROSS keeps several counters on how many edge pairs have intersected etc. Calling it with NA(1) = -2 zeroes these. NA(1) = -3 causes them to be printed. In either case, no intersection is calculated for that call.

3.15 DACOS (X)

Return the double precision arccosine of double precision X.

3.16 DATTIM(DT)

Return the current date and time in 5a4 thus:
12-mar-76 12:34:56.7

3.17 DRAW(ICHAN)

Read the coords of the endpoints of the visible segments from SBLOCK channel ICHAN and plot them. Currently DRAW will bomb if you do any of the following: replot, save plots, shade, number.
3.18 DUMPIO(LOCNO, LSEG, ISEG, KODE)

Read or write data from common/cstc/ or an array arg to disk. Data is in core in arbitrary many discontiguous segments. On disk, the data is 1 variable length record of a direct access file. The length of data read from a given record must equal the length written. However the lengths of individual segments in core need not correspond.

First a call is made to init the dsk record. then calls transfer the data, 1 segment per call. For writing, a final call closes the record. Thus each record is equivalent to a sequential file, except that the average overhead is 1/2 block instead of 7 blocks. When a record is overwritten, the new record may be longer or shorter. The record may be any length from 1 word to the disk space available.

Reading an eof is a terminal error, although this could be easily changed to return the actual length instead.

The parameters are:

LOCNO if kode=1,2,3,4,5 (initializing a record), this is the record #.
     if kode=6 (transfer next segment), this is the loc in core of the next segment:
     >=0:  segment is (isp(locno+i), i=1,lseg)
     =-1:  segment is (iseg(i), i=1,lseg)

LSEG used only for data transfer when is length of segment.

ISEG int array. used only for data transfer when see above.

KODE operation:
   1  init to read
   2  init to read and then delete record
   3  delete record
   4  init to write over null record
   5  init to write over record (that may or may not be null)
   6  read/write next segment
   7  write eof. KODE is set to 7 when a record is finished (eof read/written or record deleted).

The major local variables are:
3.19 DUMP2 (RECNO, BASE, LEN, LWRITE)

Read or write a DUMPIO record that is exactly 1 segment in ISP.

RECNO  (int) # of DUMPIO record.
BASE  (int) base in ISP.
LEN  length of record in words.
LWRITE (log) TRUE => write it. FALSE => read it.

3.20 EDGFAC(KE, KF)

To the face array, add which edges are on each face. Let $K = ISP(KF + ISP(KF + 4) + I) + J$. Then MOD(K, ISPLIT) is the J-th vertex of face #I and K/ISPLIT is the corresponding edge, from the J-th to the J+1-st (or the first) vertex of the face. The vertex and edge numbers are packed in one word to save space.

3.21 EDIT

Prompt for an object’s name, then prompt for additions, changes and deletions to its vertices, edges and faces.

3.22 EEPAIR(KV, KE, ICHAN, SE)

Assume that there is nothing in ISP that cannot be overwritten. Call EEPAIR to leave the edges and sorted (B,E) pairs in core on top of KV, KE. Write the (E,E)
pairs, each packed into 1 word, for all pairs of edges in the same bucket. However, don't write a pair if the second edge is interior. Swap the pairs in and sort them. Delete any duplicates. Swap the (E,E) pairs out to SBLOCK channel ICHAN in the following form: A record (N, 0) meaning that this edge has N crossings. A record (END1, END2) giving the endpoints of this edge. N records (CEND1i, CEND2i) giving the endpoints of the N edges crossing this edge. This is repeated for every edge. Each of these records is packed into 1 word before writing.

Before writing the records for any edge, compare the edge number with the number of the last edge for which pairs were written. A gap indicates edges with no possible intersections. Hence they are not mentioned in the (E,E) pairs. Nevertheless write sets of records for these edges. Do the same check after handling the last edge in the (E,E) pairs, then write a (-1,0) record to mark the end.

3.23 ES (KV, KV2, PT)

Transform the object with vertices based at KV from a right hand coordinate system with VIEWPOINT and CENTERPT as in common to a left hand system with VIEWPOINT=(0, 0, 0) and CENTERPT=(0, 0, 1). Scale Z to change angle of vision from VISANG degrees to 90. Do calculations in double precision for greater accuracy.

The method: Translate to make VIEWPOINT=(0, 0, 0). Dilatate about origin to make ABS(VIEWPOINT - CENTERPT) =1. Rotate about Z-axis to make CENTERPT=(1, 0, 0). Rotate about X-axis to make CENTERPT=(0, 0, -1). Negate Z-coords to make CENTERPT=(0, 0, 1). Multiply Z-coords by TAN(VISANG/2) to make angle of vision 90.

If KV2 > 0, do the same transformations to the vertices it bases. If PT= TRUE, write some debugging info.

3.24 EXCHAN(BASE, LEN1, LEN2)

Exchange ISP(BASE +I), I=1,LEN1) and (ISP(BASE +LEN1+I), I=1,LEN2) by reversing the 2 groups separately
then reversing them together. This moves every element thrice. A messier method moving every element 1 to 3 times depending on GCD(LEN1, LEN2) exists.

3.25 FACQN(KV, KF)

Calculate the equations of the faces defined by KV and KF and return them in a preallocated array based at KFQ. DO the calculations in double precision. Calculate the equation for each face from the first 3 non-collinear vertices on it.

3.26 FNDEDG(KE, NA, NB)

Return which edge has endpoints (NA, NB). return 0 if none. Assume edges are sorted within types 1 & 2. Use a binary search.

3.27 GETOBJ(NA, NB, KOB, EXTV, EXTE, EXTF, EXTLF, KEEP, LALLOC)

If the object with name (NA, NB) is in core and all of EXTV, EXTE, EXTF and EXTLF are zero, return its number in KOB. If NA and NB are blank, use the last referenced object. If the object is in core and 1 or more of them are not zero, call PUTOBJ to write the object; then act as if it is not in core (since it is not any longer). If the object is not in core, read it in, leaving space for EXTV extra vertices, EXTE edges, EXTF faces and EXTLF vertices faces. Besides leaving extra space at the ends of the arrays, the displacements of the starts of the faces must be changed. If there is not enough space for the object, swap out other objects until there is. However do not swap out object #KEEP, if KEEP .NE. 0. If there is still not enough space, this is a terminal error. if the object does not exist and LALLOC = TRUE, allocate it with size EXTV, ... Eventually, objects might be expanded in core without having to be written to disk first. Return KOB =0 on any error, such as not enough space.
3.28 GRIDIT

   Draw a grid on the screen.

3.29 GROUPS(KK)

   You initialize by calling with KK=-1. This causes GROUPS to read a line of integers that stands for a larger finite sequence of integers. A single positive number stands for itself. A pair I J with I>0 and J<0 stands for the numbers from I to -J, inclusive. GROUPS returns the next number in sequence every time it is called (including the initialization call). When there are no more it returns 0.

3.30 IBINSR(BASE, N, ELT)

   Binary search (ISP( BASE+I ), I=1,N) for ELT. The array is assumed sorted. Return ELT’s location or 0 if it is not found. Before starting the search, check whether ISP(BASE + ELT) is the answer.

3.31 IDMAT (A)

   Make the 4 by 4 double precision matrix A an identity matrix.

3.32 INDATA(KV, KE, KF, NUMCOM, PT, IER)

   Read an object from unit INUNIT into KV, KE and KF after reading and printing NUMCOM comment cards. If PT is TRUE, list the data as read. Return IER as 0 or -1 depending on whether the data was correct or not (e.g. too large).
3.33 INTERS(KV, KF, KBF, NFB)

Using the (E,E) pairs, calculate the visible line segments:

Assume the vertices, faces, face equations and (B,F) pairs are in core. KBF bases the (B,F) pairs. There are NFB face buckets on a side. Loop on the edges: Read an edge (if any edge is too short ignore it) then its putative crossing edges from SBLOCK channel 1. Up to 15 crossing edges are allowed. Call CROSS to determine which edges actually cross and return the crossing points. Sort the points along the edge. The NSEG points divide the edge into NSEG+1 segments each of which is entirely visible or entirely hidden. Since CROSS returns only the 2-d intersection points, the Z-coords must be calculated. For both the sorting and this, the distance of a point along an edge is measured by the sum of the absolute values of X and Y distances from the first endpoint. This is the easiest way to avoid having to treat horizontal or vertical edges as special cases.

Take the midpoint of each segment and call SHOW to see if it is visible. If it is, write it to SBLOCK channel 2 for later plotting.

Contents of ISP during INTERS: Assume you are plotting an array of N by N by N cubes.

<table>
<thead>
<tr>
<th>type</th>
<th>number</th>
<th>storage, each</th>
<th>storage, tot</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertices</td>
<td>NV=7*N**3</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>faces</td>
<td>NF=3*N**3</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>face equations</td>
<td>NF</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>(B,F) pairs</td>
<td>NBFP+NFB**2= 1</td>
<td>6</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>6<em>N**2+2</em>N**3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If N=6, this totals 54*216 = 12K. This could possibly reduced by a fancier algorithm such as writing the segments to disk, then later swapping them in, sorting them by face bucket and determining whether they are visible. NBFP is quadratically sensitive to the relative size of the face buckets and the faces.
3.34 IPNPOL(KV, KP, N, Z) INT FN

Is the 2-D point Z inside the polygon whose N vertices are listed based at KP?

-1 Outside
0 On the border
1 Inside

3.35 IRUNTI(DUMMY)

This MACRO function ignores its argument and returns the total CPU time since the start of the job, in milliseconds. It can be dummied out since it is only used to collect statistics.

3.36 Invper(kk, N)

Assume that NEW(I) = ISP(KK + OLD(I)) is a permutation of N elements. Invert it without using any extra core.

3.37 IPACK (LH, RH)

Pack LH and RH into 1 word and return it. This is a MACRO function.

3.38 LAND (I, J)

This is a MACRO function to return the bitwise logical AND of I and J.

3.39 LOR (I, J)

This is a MACRO function to return the bitwise logical OR of I and J.
3.40 LSH (IW, IAMT)

The result of this MACRO function is IW shifted by IAMT bits (left if IAMT > 0).

3.41 MATMUL (A, B, C)

A, B and C are double precision 4 by 4 matrices. Make C the matrix product of A and B.

3.42 MOVESP(FRBASE, TOBASE, LEN)

Move (ISP(FRBASE+I), I=1,LEN) to ISP(TOBASE+I), I=1,LEN). The ranges may overlap.

3.43 NEDFAC(KF)

Delete edge numbers stored in face array, since they are no longer needed.

3.44 OBSET

(kv, jv, mv, nv, sv, ke, je, me, ne, nel, se, kf, jf, mf, mlf, nf, nfl, nlf, sf)

Given KV, KE and KF set the other variables. If any of these 3 is 0 on entry, the variables corresponding to it are not set. All of the result variables that are not wanted may be defaulted by the same dummy variable; do not use a constant since it might be changed. The variables are mostly obvious except that MV etc. refers to # of elements and SV etc. to # of words.

Print the object defined by (KV, KE, KF) with the title TITLE (10A4). If any of KV, KE or KF is 0, don’t print its array.
3.45 OUTSEN (KV, KE, KF, ITITLE)

Print the object based at KV, KE, KF with the title ITITLE (18a5). If any of KV, KE or KF is 0, omit that part of the object. If the output unit is the terminal, vary the format according to the object so as to pack the listing into as little space as possible.

3.46 PERT (KV1, KV2)

Do a perspective inversion to change the vertices based at KV1 from VIEWPOINT= (0, 0, 0) to VIEWPOINT= (0, 0, infinity). Straight lines and flat planes remain straight and flat. If KV2 is not 0, change those vertices also. The transformation for every point (X, Y, Z) is:

\[
\begin{align*}
X & \leftarrow X/Z \\
Y & \leftarrow Y/Z \\
Z & \leftarrow 1/Z
\end{align*}
\]

3.47 PUTOBJ(KOB).

If object #KOB exists and is in core and the version in core is different from the version on disk (or there is no version of this object on disk), then compress out all unused space and write it to disk. When the faces are compressed, the pointers to the start of each face must be changed. The compression is not to save space on disk, but to save space when the object is read in again. Mark the object as no longer in core. Move down the rest of ISP after the object and update the pointers in array OBJ. I assume that the object is contiguous in ISP with the vertices first. Any other pointers to addresses in ISP will be invalidated. To delete an object from ISP without writing it to disk, mark it as unchanged in OBJ.

3.48 SBLOCK(REC, KODE, LOC, ICHAN)

Block a sequential file with short records. Do not scan records over block boundaries. You are responsible for
setting SBUNI, SBLLEN and BLKSIZ in common. You must write
an EOF between writing and reading. You cannot append after
writing EOF. If writing the whole file at once, you must
set SNREC.

Reading an eof is a terminal error, although this could
be easily changed. For speed, I handle 1 word records
separately if KODE= 1 or 2.

Parameters and common variables:

KODE
1  Read REC. Set KODE to -1 at EOF.
2  Write REC.
3  Write EOF.
4  Read whole file into (ISP(LOC+I), I=1,SNREC(ICHAN)).
    Set SNREC(ICHAN) to length of file.
5  Write whole file into the same.
6  Zero the file.

ICHAN  Logical channel number.  1 or 2.
SBLLEN(2) Logical record length.  1<= SBLLEN(ICHAN) <= BLKSIZ.
    You set it.
BLKSIZ  Physical block size set by you.
SBUNI(2) Unit numbers set by you.
SNREC(2) Number of records in each file.
SIREC(2) Number of blocks that have been read when reading.
    Set by program.

Even for KODEs 5 & 6, SBLLEN is used and the whole block
is not filled. Thus the file can be written with KODE =2,
say and read with KODE =4.

3.49 SBLOC2 (REC, KODE, LOC, ICHAN)

Do SBLOCK operations on 2 word records that can be
packed into 1 word on disk. REC is INT(2). SBLLEN(ICHAN)
must be 1.

KODE OPERATION
1  Call SBLOCK to read the next record, unpack it into
    REC and return it.
2  Pack REC into 1 word and call SBLOCK to write it.
4  Read the whole file into core, delete it from disk,
sort it in core and unpack it into (ISP( LOC+ 2*I +J), J=1,2), I=1, SNREC(ICHAN)).

Call SBLOCK directly to do anything else such as writing EOF.

3.50 SHOW (KV, KF, KBF, Z, LYES, NFB)

If Z(1) = -1.23, init certain vars and, if debugging has been requested, test all faces to see whether they fit their equations. This call must be made first for every object.

For the other calls, determine whether 3-D point Z is visible. First find which face bucket Z is in, then test it against all faces in the bucket. If it is behind any face’s plane extended, test whether it is inside the face’s projected 2-D (on the X-Y plane) polygon. If so it is hidden by that face. Set LYES= TRUE if Z is visible.

3.51 SOLDEL(KE, KF, KK)

Assume the face equations have been calculated and are based at KFQ. KK bases a work array of NE elements. Delete any faces the first component of whose equation is TYPIGN. Delete all backward facing type 2 faces and all edges not on a remaining face. ne an interior edge to be one on at least 1 forward type 2 face but on no backward type 2 faces. Flag these edges by negating their first endpoints. COMPRESS out deleted edges and faces.

3.52 SORT(KB, RECLEN, NREC, KK)

Shell sort an integer array of NREC records, each of length = RECLEN (<=10). The array is based at KB. Even when RECLEN =1, KB bases a 2-D array. If KK >0, it bases an array that is returned with the new location of each old element. For speed, handle the case RECLEN=1 separately. This goes a lot faster since comparing and moving records is just 1 operation instead of a DO loop.
3.53 TIMES (N)

Keep track of how many times subroutine #N is called, and how much time it uses. TIMES must be called on entry (CALL TIMES (N)) and on exit (CALL TIMES (-1)). The routines may be nested up to 99 deep. Also, optionally trace calls to and returns from timed routines.

Bump RNCALL everytime TIMES is called. Then then when IRUNTI returns the CPU time, lower that time by RNCALL * RTCORR to hopefully negate the time used in calling TIMES itself.

3.54 TOTMAC (TLAND, TLOR, TLSH, TIPACK, TUNP)

This MACRO subroutine returns in its arguments the number of times the functions LAND, LOR, LSH, IPACK and UNPACK have been called. This is faster than having them call TIMES.

3.55 TRANS (KV, TRA, PX, QX, ARD, KEY)

Transform the vertices based at KV according to KEY:

KEY OPERATION
1 Transform with matrix TRA.
2 Stretch by factor QX about fixed point PX.
3 Shift by PX.
4 Rotate by ARD degrees about an axis through PX and QX.
5 Calculate a transformation matrix for the above rotation and return it in tra.

The calculations are done in double precision and TRA, PX, QX AND ARD must be double precision, although the vertices are single precision.
3.56 UNPACK (LHRH, LH, RH)

This MACRO subroutine splits the 2 halves of LHRH into LH and RH, extending the signs.

3.57 VARDA (BLOCK, LEN, RECNO, KODE)

Implement a direct access file with variable length records of no maximum length. Any record can be overwritten by a new record of different length (longer or shorter). The record is handled 1 physical block at a time. a block has BLKSIZ-1 data words and 1 link word that the caller does not alter. RECNO is the number of the record. When writing, LEN= -1 for all but the last block. For the last block, LEN= the number of data words in this block. 0<= len < blksz. When reading, len is set like this. KODE is changed to 7 when the first block of the record is handled, and to 0 when the last block is processed. When writing or deleting, block is overwritten. If KODE=8 or 9 so that 1 block is read or written, the link field is not changed. KODE is returned as -1 on errors such as illegal block number.

If HIBLOC=0 on call, VARDA initializes everything. This should happen on the first call. If HIBLOC<0, call DEFINE FILE to open the file again, then set HIBLOC = -HIBLOC. This should happen on reentering after an error.

If KODE=11, the current table block is written, then FREE, FREE2 and HIBLOC are written to the end of the last table block. Then if you stop VIEWPLOT before calling VARDA again; and call VARDA with KODE=12 when next you run VIEWPLOT, as far as VARDA is concerned it will be as if there was no interruption. This is useful for saving lots of big objects without writing and reading them again.

KODE: 1 Read
       2 Read & delete
       3 Delete
       4 Write over null record
       5 Write over null or nonnull record
       6 Print tab & link fields of blocks
       7 Continue last operation (kode=1,2,4,5) with next block.
VIEWPLOT PROGRAM LOGIC MANUAL
ROUTINE DESCRIPTIONS

8 Read 1 block
9 Write 1 block
10 Zero the file
11 Write the important VARDA variables to the file.
12 Read these variables in.

TAB table of first block no, last block no for each rec.
          0 = empty record.
TMAX # table blocks
TNOW current table block in core.  0=None.
RMAX highest rec # allowable.  0< recno < rmax.
HIBLOC highest block # written so far.
BLOCK current file block. last word, if >0, is # of next block of rec.
          if <=0, this is the last block of the rec, and the negative of this is
          the # data words in this block. block is destroyed after being written.
          it must be supplied for and is destroyed by deleting a record.
FREE ptr to first of chained free blocks. always points to a block.
          the free list is always at least 1 block long.
FREE2 ptr to second block in free list. if only 1 block
          in free list, free2=hibloc+1.
KBLOCK number of first block of file, except next block when reading.
PER # record entries per table block.
LAST # of last block of record when deleting.

3.58 VCROSS (X, Y, Z)

X, Y and Z are double precision(3). Return Z as the vector product of X and Y.

3.59 VECLEN (V)

Return the length of the double precision(3) V.
3.60 VERR (KODE)

Print the error code KODE with a message and stop. This is a convenient termination for errors that should never happen. There are 2 formats for the digits of KODE: The old is XXYY where XX is the routine number in a table in VERR. The new is XX0YY where XX is the routine number in array ROUTIN in common. In either case, YY is the error number.

3.61 YNTF(L1, L2, L3, N)

Accept N (<=3) chars and set L1, L2 and L3 accordingly: If Y or T set TRUE. If N or F set FALSE. IF blank, con’t change. If anything else, print an error message and try again.
4.0 STORAGE ALLOCATION IN CORE

All the significant arrays are allocae from one big common array, ISP. It is equivalenced to a real array, RSP. This allows me to use variable size arrays, allocate and equivalence arrays at will, and to efficiently pass these arrays to subroutines.

ISP has 3 header words, 2 equivalenced to scalars:

- ISP(1) = a code identifying ISP.
- ISP(2) = NISP = the highest used word in ISP.
- ISP(3) = MISP = the size of ISP in words.

Arrays in ISP are referred to by their dope pointer, e.g., KV for a vertex array. This pointer is generally 1 word below the start of the array, except for simple multidimensional arrays such as the face equations. Thus for this 4 by NF array, KFQ the pointer is 5 below the start. This is for ease of addressing: the j-th coord of the i-th equation is

\[ RSP(KFQ + 4*I + J) \]

This does not apply to the vertex, edge and face arrays. Although they seem multidimensional, they contain various info at the front, such as the number of edges of type 1, so that the whole array is considered single dimensional. Therefore KV is 1 before the start of the vertex array.

Whenever an array in ISP has internal pointers and subarrays of its own, these pointers are always relative to the start of the array, not to the start of ISP. This is so the array can be moved around as a whole.

Arrays are frequently swapped between disk and core using routines DUMP0, DUMP2, SBLOCK and SBLOC2.

You can get space for patching VIEWPLOT in DDT from the end of ISP. BE sure to adjust MISP however, since routines such as BFAIR may temporarily allocate ISP from the end forward.
5.0 OBJECT DATA FORMAT

Each object is composed of 3 arrays which are parts of ISP. There is 1 each for the vertices, edges and faces. The vertices array, since it is real is referenced in RSP which is equivalenced to ISP. Each array is referred to by its base location, commonly the variables NV, NE and NF. Thus the edges array, say, is ISP(KE+1), ISP(KE+2), ... The format for the arrays is:

VERT: 1 Type
2 Name
3 Size (in words)
4 Displacement for first vertex (=3)
5 Number of vertices in use
6 (Unused)
7 X-coord )
8 Y-coord ) first vertex
9 Z-coord )
10,11,12 Second vertex
...

EDGE: 1 Type
2 Name
3 Size (in words)
4 Displacement for first edge (=4)
5 # of edges in use
6 # of type 1 edges in use
7 # of vertex at first end of first edge
8 # vertex number at second end of first edge
9,10 Edge #2
...

FACE: 1 Type
2 Name
3 Size (in words)
4 Displacement to pointers (=6)
5 # faces in use
6 # type 1 faces in use
7 rel base of first face (=K, say)
8 rel base of second face
...
K+1 first vertex of first face
K+2 second vertex of first face
...
The type is a code that says what type of array this is. It is different for vertex, edge and face arrays. The name field tells which object this array belongs to. I use the above 2 occasionally for error checking. The size is the number of words this array takes in ISP. It and the displacement can be used to tell the maximum number of vertices or whatever the array can hold. The displacement is relative to the start of the array. For vertices and edges it is used to address the actual vertices and edges thus:

Coordinate #J of vertex #I:
RSP(KV + RSP(KV+4) + 3*I + J)

End #J of edge #I:
ISP(KE + ISP(KE+4) + 2*I + J)

For all objects created to date, the displacement is constant as given. It is made variable to allow for future fields in the arrays. The face displacement is to a sub-array giving the base of each face. As always, the displacements are relative to the start of the faces array. The sub-array of displacements is 1 before the start of each face. Thus:

Vertex #J of face #I:
ISP(KF + ISP(KF+ISP(KF+4)+I) +J)

Number of vertices on face #I:
ISP(KF+ISP(KF+4)+I+1) - ISP(KF+ISP(KF+4)+I)

The number of words each array takes is:

NV = # of vertices
NE = # of edges
NF = # of faces
NLF= Sum of the number of vertices in each face

WV = Minimum size of vertex array, in words
WE = " " " edge " " "
WF = " " " face " " "

WV = 3 * NV + 6
WE = 2 * NE + 6
WF = NF + NLF + 7
In the edge and face, the type 1 edges or faces are all listed before any of the type 2 faces. The edges are listed with the smallest endpoint first. An edge or face is usually marked deleted by zeroing the first vertex in it.

Other arrays in ISP are usually simpler: they have no information words at their start. For multi-dimensional arrays, the base is set so the array can be directly addressed. Thus for the 4-dimensional face equation array, the equation for the I-th face is:

\[
\text{rsp} (kfg+4*i+1) * x + \text{rsp} (kfg+4*i+2) * y + \\
\text{rsp} (kfg+4*i+3) * z + \text{rsp} (kfg+4*i+4) = 0.
\]
6.8 UNITS

TTYUNI permanent tty unit (5)
INUNIT current input unit for commands (either TTYUNI or INUNI2)
O current output unit for msgs & plots
PUNUNI current output unit for data files (26)
INUNI2 command file input unit (27)
SBUNI(2) temporary seq files used by sblock (28, 29)
IOSTAT output unit for statistics and run times (25)
DIRUNI direct access unit for VARDA (9)
.0 VARDA RECORDS

Vertices of object being plotted.

Edges of object being plotted.

Faces of object being plotted.

Face equations.

COM1.

01-200: Objects. Object #I is record #I+100. The format is 3 words giving the number of words in the vertices, edges and faces respectively, then the 3 arrays with the vertices, edges and faces. They have all unused space compressed out.
8.0 SBLOCK UNITS

SBLOCK is currently implemented to block short records into sequential files with 2 units. They are used thus:

1. (B,E) pairs (packed) and (E,E) pairs (packed).
2. (B,F) pairs and the visible segments.
9.0 CONVENTIONS

1. The first line of every routine is c>>name

2. The date is near the top of each routine in the form C<TAB>DATE:<DATE>.

3. The routines called are given thus: C<TAB>ROUTINES CALLED: A, B, ...

4. The variables in common used are given: C<TAB>COMMON USED: A, B, ...

5. A description of the routine is not given at the top of the routine, but in VPLT2.RNO.

6. The common to be inserted is delimited by c$scom & c$ecom

7. At the start of each file, give the date and the routines it contains.
10.0 DEC FORTRAN DEPENDENCIES

Some of these would only affect a pretty poor compiler and most of them would be easy to change although it would be difficult to make the file handling machine independent particularly as the FORTRAN standard does not support direct access.

1. Use of $ in FORMAT statements in ALC.
2. Use of I and F format items without field widths in ALC, edit and GROUPS.
3. Use of END= in ALC reads.
4. A word must hold at least 4 characters.
5. Use of IFILE and OFILE in ALC.
6. The value of IBIG is the biggest integer (set in DAT).
7. Use of subscripts more general than C*V+C, altho nested subscripts used only in VERR and TIMES.
8. Direct access I/O used in VARDA.
9. VPLMAC.MAC contains MACRO routines. Some return the runtime and set the devices of all units except 5 to DSK:. Others do logical and, or and shift. These are used only to format the current date and could be dummied. IPACK and UNPACK pack and unpack 2 words into 1. They could be written in FORTRAN.

10. The unit numbers may be too big or wrong for some systems.
11. If you set BLKSIZ bigger than 127, you must increase fixed arrays in VARDA and DUMPIO.
12. I equivalence real and integer arrays and sometimes copy reals as ints.
13. If during a certain routine invocation I won't use a logical or array parameter, I usually dummy the argument with scalar 0. This will cause a type fault in say, WATFIV.
14. Use if A5, not A4 in OUTSEN and use of literal constants in arguments to OUTSEN.
11.0 OVERLAYING

Altho I have never tried to overlay, the routines are set up so that you could save a lot of storage. DUMPIO, SBLOCK and VARDA are the only routines that have local variables that must always be preserved from call to call. Also CROSS, SHOW and GROUPS must be preserved from call to call within one object, but they can be swapped out between objects.
12.0 NEW VARIABLE DESCRIPTIONS

MOBJ    Maximum number of objects.
OBJ(6, NOBJ) Short description of objects:
   1,2: Name. Blank if this object doesn't exist.
   3: KV. -1 of object not in core
   4: KE
   5: KF
   6: If object is in core: 0 if it has not been
       modified, -1 if it has.
KOBOld  Number of last object accessed in ALC.
13.0 LONG TERM SUGGESTIONS FOR IMPROVEMENT

1. NEWS.VPT
2. DRAW, HTSIN, PRISIN, TESSEL - TMPUNI
3. SOS for VPLT2
4. sort (B,F) pairs by face distance.
5. GIN mode to give vertex numbers.
6. Calculate inverse transformation for vertices.
7. Carry vertex ids through calcs.
8. Make file of segs, polys
14.0 OLD VARIABLE DESCRIPTIONS

rsp  the real storage array
isp  the integer storage array
misp size of isp
typver  type of vertices array
typedg  type of edges array
typfac  type of faces array
typpoi  type of array of 2-d points
typlin  type of array of 2-d lines
o  output unit for messages
inunit  input unit for data
eye  coordinates of viewpoint
viewpt  coordinates of centerpoint
toler relative fudge factor. this should be larger than
the largest error in viewplot due to roundoff or
loss of significance. in the pdp-10 this is at
least 3e-5. (it could be lowered by double
precision in critical calculations).

visang  visual angle (degrees)
lhid  should hidden lines be plotted?
ltes  should faces be coloured?
big  largest real number
ibig  largest integer
lplot  should result be plotted?
lprin  should result be printed?
lvis  should visible lines to be plotted?
nob  number of objects
perinc  number of lines per inch on printer
tmpuni  unit for temporary storage
pltsiz  size of printer-plot (inches square)
lgrid  should a grid be drawn on the plot?
keb  base of an array of edge bounds
kfb  base of an array of face bounds
keq  base of an array of edge equations
kzq  base of an array of "z-equations" which return the
    z-coordinate of a point on an edge given the x- and
    y-coordinate.

k nep  see tessel
kep  see tessel
mep  see tessel
kp ol  see tessel
mp ol  see tessel
kang  see tessel
lb tes  should various info be printed in tessel?
iplane  see tessel
VIEWPLOT PROGRAM LOGIC MANUAL

OLD VARIABLE DESCRIPTIONS

kvax  see tessel & ala
shine  see tessel
kvexi  see inters
typign code inserted in face equation if face to be ignored
kseg  see inters
mseg  see inters
lstere  should stereo pictures be plotted?
stesep  stereo separation (angle subtended by viewpoints at
centerpoint) in degrees
device  1 - tektronix.  2 - ards
iye  (1 or 2) which viewpoint of the stereo picture is
    being processed
nshad  # words of shading info written on tmpuni
routin  names of routines
ltime  should timing stats be taken?
rtime  time spent in routines so far in milliseconds
rtime2 time spent in routines & in routines called by these
    routines
rnumb  number of times routines have been called
nrouin  number of routines
machin  1- pdpl0 2- other
munits  highest unit no. allowed
plunit  unit to save several plots on to be combined
15.0 OLD STORAGE ALLOCATION IN ALA

storage allocation: (an array is referred to by its base variable)
1. allocated by stosen and used thruout ala & descendent routines:
   real:    kv, kvax, kfq, kfb
   int:     kf, kvis, khid
2. allocated by stosen (ke) or ala (others) for inters and freed after it.
   real:    keq, kzq, keb
   int:     ke, kevi, kseg
3. allocated by ala for duration of plagra & clump:
   real:    keg, keb
   int:     kk
4. allocated by ala for duration of tessel:
   real:    kang
   int:     knep, kep, kpol
16.0 CALLING TREE (OUT OF DATE)

main   alc   addon
       alloc
       box
       bound
       trans
copsen
datast
draw   alloc
       hatch
       alloc
       pltlcg
       numlcg
       pltlcg
       plot
       pltlcg
       window
htsin
indata
obset
outsen
pix
prisin
punch
trans
ynf

ala

alloc
clamp
draw   alloc
       hatch
       alloc
       pltlcg
       numlcg
       pltlcg
       plot
       pltlcg
       window
edgfac
edgqn
efbou
es
       trans
       idmat
       matmul
       veclen
facqn
       areav
       veclen
       vcross
gridit
       pltlcg
       numlcg
inters
cross
show
nedfac
outsen
pert
plagra
cross
edgqn
swed
efbou
pltset
plots
window
scalev
bound
trans
soldel
srtedg
stosen
alloc
tessel
ptin
ipnpol
areav
onface
ipnpol
trans
vcross
zedqn
srtver
dumpio
varda
(everything)
times
irunti

internal plotting routines are not listed.
17.0 ROUTINE CROSS REFERENCE

<table>
<thead>
<tr>
<th>routine</th>
<th>called from</th>
</tr>
</thead>
<tbody>
<tr>
<td>addon</td>
<td>alc</td>
</tr>
<tr>
<td>ala</td>
<td>main.</td>
</tr>
<tr>
<td>alc</td>
<td>main.</td>
</tr>
<tr>
<td>alloc</td>
<td>ala alc draw hatch stosen</td>
</tr>
<tr>
<td>areaav</td>
<td>facqn tessel</td>
</tr>
<tr>
<td>bound</td>
<td>box</td>
</tr>
<tr>
<td>clump</td>
<td>ala</td>
</tr>
<tr>
<td>copsen</td>
<td>alc</td>
</tr>
<tr>
<td>cross</td>
<td>inters plagra</td>
</tr>
<tr>
<td>dataset</td>
<td>alc</td>
</tr>
<tr>
<td>draw</td>
<td>ala alc</td>
</tr>
<tr>
<td>dumpio</td>
<td></td>
</tr>
<tr>
<td>edgfac</td>
<td>ala</td>
</tr>
<tr>
<td>edgqnn</td>
<td>ala plagra</td>
</tr>
<tr>
<td>efbou</td>
<td>ala plagra</td>
</tr>
<tr>
<td>es</td>
<td>ala</td>
</tr>
<tr>
<td>facqn</td>
<td>ala</td>
</tr>
<tr>
<td>fnndged</td>
<td>edgfac</td>
</tr>
<tr>
<td>gridit</td>
<td>ala</td>
</tr>
<tr>
<td>hatch</td>
<td>draw</td>
</tr>
<tr>
<td>htsin</td>
<td>alc</td>
</tr>
<tr>
<td>idmat</td>
<td>es</td>
</tr>
<tr>
<td>indata</td>
<td>alc</td>
</tr>
<tr>
<td>inters</td>
<td>ala</td>
</tr>
<tr>
<td>ipnpol</td>
<td>onface ptin</td>
</tr>
<tr>
<td>irunti</td>
<td>times</td>
</tr>
<tr>
<td>main.</td>
<td></td>
</tr>
<tr>
<td>matmul</td>
<td>es</td>
</tr>
<tr>
<td>nedfac</td>
<td>ala</td>
</tr>
<tr>
<td>numlcn</td>
<td>draw gridget</td>
</tr>
<tr>
<td>obset</td>
<td>alc</td>
</tr>
<tr>
<td>onface</td>
<td>tessel</td>
</tr>
<tr>
<td>outsen</td>
<td>ala alc</td>
</tr>
<tr>
<td>pert</td>
<td>ala</td>
</tr>
<tr>
<td>plagra</td>
<td>ala</td>
</tr>
<tr>
<td>plot</td>
<td>draw pltlcg</td>
</tr>
<tr>
<td>pltlcg</td>
<td>draw gridget hatch numlcn</td>
</tr>
<tr>
<td>prisin</td>
<td>alc</td>
</tr>
<tr>
<td>ptin</td>
<td>tessel</td>
</tr>
<tr>
<td>punch</td>
<td>alc</td>
</tr>
<tr>
<td>show</td>
<td>inters</td>
</tr>
<tr>
<td>soldel</td>
<td>ala</td>
</tr>
<tr>
<td>srtedg</td>
<td>ala</td>
</tr>
</tbody>
</table>
VIEWPLOT PROGRAM LOGIC MANUAL
ROUTINE CROSS REFERENCE

srtver
stosen
swed
tessel
times
alc
box
edgqn
hatch
onface
punch
swed
trans
varda
cross
veclen
verr
window
yntf
zedqn
ala
plagra
ala
alc
box
edgqn
hatch
onface
punch
swed
trans
varda
cross
veclen
verr
window
yntf
ala
ala
addon
ALLOC
arenv
bound
addon
ALLOC
BOUND
box
edgfac
edgqn
efbou
es
cross
fndedg
gridit
hatch
inidata
inters
ipnpol
medfac
onface
pert
plagra
prisin
ptin
punch
soldel
srtedg
srtver
stosen
swed
trans
zedqn
dumpio
obset
alc
ala
box
es
facqn
tessel
alc
clump
ccross
draw
edgfn
facqn
hatch
inidata
inters
onface
outsen
ptin
plagra
ptin
show
soldel
swed
alc
alia
ala