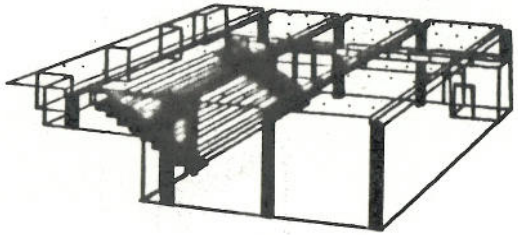
BUILDER

A data-base and display program for computer-aided architectural design.



Bruce Donald
Lab for Computer Graphics
Harvard Graduate School of Design

The BUILDER Program

A System for Representation and display of Design Problems
in Architecture and Urban Design

Contents:

Researcher and Author: Bruce R. Donald

Laboratory for Computer Graphics and
Spatial Analysis
Graduate School of Design
Harvard University
Cambridge, MA

(1981-1982)

The BUILDER Program: An Overview

(This overview was written for a GSD Exhibition by Bruce Donald and Steve Oles).

The BUILDER program was written in the course of research into analytic and computer-aided methods for architecture in the GSD's Laboratory for Computer Graphics. This exhibit features its graphic display capabilities for visual analysis, and illustrates the appropriateness of this approach for quickly generating any number of interior perspectives by varying the station point, center of vision, and lens focal ratio.

A design is represented to the machine as an aggregation of 3-dimensional forms; these forms are positioned in space and have geometric properties (such as length, orientation, and volume) as well as descriptive, functional, or structural properties (such as color, construction materials, bearing surfaces, and connectivity).

Each individual form is represented by a process analogous to "extrusion." First, a "die", or homogeneous cross-section is chosen. These are usually simple 2-dimensional forms that the designer can recognize in plan and section, such as the base of a column or the cross-section of a beam or frame. The "die" is positioned in space and moved a certain distance: in moving it "sweeps out" a volume or surface.

In practice, this process of creating the 3-dimensional representation of the design involves:

1. "Digitizing," or Tracing plan and section representations (projections) of the 3-dimensional forms from a complete set of plan and section drawings of the design project.

2. For each 3-Dimensional element, or for each group of elements (for instance, "all the columns", "all the steps") specifying their position and "thickness".

Functional or structural properties may also be specified to allow such operations as "drawing all the load-bearing walls." More complicated geometric forms may be obtained by "tilting" or "splaying" the faces of selected 3-dimensional objects.

Once the design is entered, any number of 1, 2, or 3-point perspective views may be generated from any station point looking at any center of vision.

The focal ratio of the lens is used to draw the perspectives as they would look with normal, wide-angle, or telephoto lenses of various sizes. The station point and center of vision can be given by "digitizing" a point on the plan or section of the design. (This is done using cross-hairs to pick out the spot where you wish to be standing "in" the design).

BUILDER can draw views with hidden surfaces removed by determining, for a particular viewpoint, what forms can be seen and what forms are obscured. In these views, the value BUILDER computes for visible surfaces is related to the angle of incident light. However, this is a very primitive model of value delineation, and is used only to provide a

"symbolic" representation to make the spatial relationships apparent.

"Realistic", or "true" shading for computer-generated interior perspectives is particularly difficult. More complicated mathematical models of light and shading would take into account the inherent color, reflectance, and specularity of forms, and allow non-uniform, multiple light sources. In addition, texture, shadows, and the light reflected from surrounding objects could modulate the value.

In order to perform hidden surface and shading calculations, the program must be able to compute a geometric model from the design specified by the architect. From this model, it must be able to recognize the surfaces and facets that form the boundary of 3-dimensional objects. The problem can be considerably complicated by intersecting surfaces or forms imbedded within or interfering with other forms.

Hidden surface analysis, and many other sophisticated operations require a geometric model of the design that is free from topological anomalies. Current laboratory research efforts are directed towards detecting and correcting such anomalies. This work would also provide the architect with powerful tools such as interference and connectivity analysis in complex structures. The work exhibited here demonstrates the display capabilities of such an approach for visual analysis.

This research is being conducted by Bruce Donald, a Research Analyst in the Laboratory.

BUILDER Glossary

Bruce Donald, Research Analyst
Laboratory for Computer Graphics and Spatial Analysis
Graduate School of Design
Harvard University
Cambridge, MA
U.S.A.
(617) 495 - 2526

January 31, 1982

Draft - Comments to Author

1.0 CONTENTS

2.0 BUILDINGS AND FRAGMENTS

BUILDER was originally designed for site planning, Thus, each individual form is called a BUILDING in the program. However, for architectural applications, the term "entity", "structure" or "fragment" is more appropriate to describe these sub-entities.

The language will be changed to reflect this and make both terms acceptable.

3.0 CONVENTIONS

All commands to BUILDER are terminated with a semicolon:

```
DRAW SECTION;

PLOT PERSPECTIVE ALL;
```

Values (numbers, strings/text, filenames, etc.) must be preceded by a colon:

```
MAKE FOCALRATIO: 4 ;

MAKE CENTEROFVISION HEIGHT: 55;

CREATE IMAGE FILENAME: 'SECTION.DSK';
```

Strings/text must be in matched quotes. Tuples, or sequences of numbers, must be enclosed in matched parentheses:

```
MAKE STATION POINT: (30,100,3000.3);
```

At any time in BUILDER, you may type a "question mark" (?) to see what are valid words to say next. (Eg: " DRAW ?" will show you what you can say after DRAW). If you must type a value, BUILDER will respond "<value>".

In this document, optional portions of commands are enclosed within brackets. Thus, in the command:

```
CREATE IMAGE [ FILENAME: 'image.dsk' ] ;
```

The notation implies that the sentences:

CREATE IMAGE; and CREATE IMAGE FILENAME: 'image.dsk'; are *both* valid.

Most BUILDER parameters are "Evaluable Variables" -- that is they may be used in value expressions. For example, if you have specified the center of vision height, and wish to make the station point at the

same height, you say:

```
MAKE STATION HEIGHT:  CENTEROFVISION HEIGHT;
```

More complicated operations with "evaluable BUILDER variables" are possible:

```
MAKE STATION HEIGHT:  CENTEROFVISION HEIGHT + 50;

MAKE STATION HEIGHT:  CENTEROFVISION HEIGHT * 2 + 5;

MAKE STATION POINT:  CENTEROFVISION POINT - (100,0,200);

OPEN INPUT FILE:  OUTPUT FILE;  -- Open as input the current output file.

MAKE STATION POINT: 2*(CENTEROFVISION POINT - STATION POINT) +
                        (4, 50, STATION HEIGHT/2);

IF: (BUILDING BASE - BUILDING TOP) < 5 ; DRAW BUILDING; FIN;
```

Any BUILDER word or command may be abbriviated to an unambiguous "short form". Thus the sentence "DRAW PERSPECTIVEVIEW ALL;" can be written "DR PER AL;"

3.1 Evaluable Variables, And Defined Variables

The Novice BUILDER user may wish to skip to the next section, and learn about Evaluable varibles and defined BUILDER variables at a later time.

The BUILDER user can "define" his own variables, which may be of any kind. This is analogous to the "LET" statement in BASIC.

```
DEFINE HERE:  STATION POINT + (0,0,50);

DEFINE THERE:  (40,500,0);

DEFINE SCALE:  88.8;

DEFINE INVSCALE:  1/scale;

DEFINE THISFILE:  "RICK.BAS";

DEFINE THATFILE:  INPUT FILE;
```

Thee defined lexemes may be used in value/variable expressions:

```
DEFINE FLOOR:  103;

MaKE BUILDING BASE:  FLOOR;

MAKE BUILDING TOP:  2* floor + 5;

MAKE STATION POINT:  HERE/2;

MAKE BUILDING EXTRUDEDFACE XTILT:  2 * SCALE/16;

DEFINE FLOOR:  FLOOR - 15;
IF: BUILDING LENGTH< Maxpts & Building top < MaxHeight; Copy Building; Fin;
```

4.0 DIALECTS

There are two dialects in BUILDER, DRAWING and DIGITIZING. In DRAWING, you can have an input and output file open at once. The Input fragments (or "BUILDINGS") are processed in a local buffer, or storage space, one at a time. Then they can be written out, or COPIED to the output file. Once copied, they remain in the buffer, and can be transformed, flipped, edited, and so forth, and then written out again as new buildings (instances of the original form).

In the DIGITIZING mode, you can create fragments ("BUILDINGS") from scratch. No Input file can be opened in digitizing, ut if you swith into this mode, your current input file remains in its same status when you go back into DRAWING mode.

The modes are to distinguish between editing an exising supply for the buffer (the file) and new data-- the graphic crosshairs as input.

Modes are switched with the DO command.

5.0 VERBS IN BUILDER

Most verbs are valid in both modes.

5.1 DO

Used to Switch operating modes:

```
DO DRAWING;
```

DO DIGITIZING;

The input file in drawing mode is maintained, though unaccessible, throughout DIGITIZING mode.

5.2 DRAW

The generic DRAW command is used to set up parameters for drawing, and to actually draw objects and data bases.

5.2.1 DRAW Parameter Commands -

1. DRAW PLAN; -- Draw (in the future) Plan projections.
2. DRAW SECTION or DRAW ELEVATION; -- draw (henceforth) sections or elevations.
3. DRAW PERSPECTIVEVIEW; Draw Perspective views .
4. DRAW [NO] LABELS; -- Draw identifier labels.
5. DRAW [no] CONNECTINGENDSEGMENTS; -- Command to connect the first and last die points. Globally specifies line or polygonal dies.
6. DRAW [NO] AWAYFACES; In hidden surface views, draw the faces the point away from the eye.

5.2.2 DRAW Commands That DRAW Something -

1. DRAW BUILDING; -- Draw the current BUILDING or fragment in the buffer.
2. DRAW BUILDING: 13;

   Search the input file for building 13, and draw it. Used to get a specific building/fragment into the local buffer.
3. DRAW NEXT; -- Draw the next building/fragment in the input file.

4. DRAW ALL; -- Draw all buildings/fragments in the data base (input file).

5. DRAW TEMPLATE ALL; -- Draw all fragments and buildings in the template file.

6. DRAW GRID; -- Draw a grid. Also :

   DRAW GRID: (x - spacing , y- spacing) ;

5.3 DIGITIZE

In any mode, you may digitize the station point and center of vision point. In DIGITIZING Mode, you may also digitize a new fragment die, thus filling the building buffer and deleting whatever was in the local buffer before.

DIGITIZE STATION POINT;

DIGITIZE CENTEROFVISION POINT;

DIGITIZE BUILDING; -- in DIGITIZING mode, this command lets you digitize a new building/fragment die, in either plan or section plane, depending on the setting of BUILDING PLANE. The graphic crosshairs are used to digitize successive die points. The last point is indicated by a "Q" (for Quit Digitizing). If the DRAW CONNECTINGENDSEGMENTS; command has been given, the first and last points of the die will be connected to create a polygonal die. (This is also the default mode). A polygonal die will extrude a volume; a "line die" (DRAW NO CONNECTINGENDSEGMENTS;) will extrude a surface (a "curtain").

5.3.1 Digitizing Notes -

The DIGITIZE command can be used to the exclusion of the MAKE command to specify the station point and centerofvision point in 3-dimensional space. If the DRAWINGMODE is PLAN, BUILDER takes the (X, Y) coordinates of the station/centerofvision point from the graphic input. If the DRAWINGMODE is SECTION or ELEVATION, it reads the (Y, Z) coordinates. Thus, to specify both points in 3-dimensional space relative to the existing plan and section, you can either:

1. DRAW PLAN ALL;

   DIGITIZE STATION POINT, CENTER POINT;

   MAKE STATION HEIGHT: z-value, CENTEROFVISION HEIGHT: z-value;

   or

2. DRAW PLAN ALL;

   DIGITIZE STATION POINT, CENTEROFVISION POINT;

   DRAW SECTION ALL;

   DIGITIZE STATION POINT, CENTEROFVISION POINT;

5.4 DEFINE And DELETE

Used to define and delete variables for BUILDER operations. Example:
DEFINE RICK: "THIS IS RICK'S FILE";

DEFINE SHIFT: (-5.4, 500);

Define length: 88.4# shift;

5.5 OPEN

Command used to open files.

OPEN INPUT FILENAME: ' Name ';

OPEN TEMPLATE FILENAME: ' Tname ';

The Input file buildings/fragments can be brought into the local buffer for modification and copying, as well as drawing. The Template file is a standard builder file or polygon file, but can only be drawn: it is analogous to a "Graphic overlay" and is used to compare current work to a template. The most useful command referencing the template is DRAW TEMPLATE ALL; which draws the template WITHOUT ERASING THE SCREEN, and leaves the local building buffer intact and the input file position and status unchanged. Thus a structure may be carefully fitted into a data base, by comparing it again and again to the template as it is honed. The Template is drawn with the same parameters as the input file.

On friendly machines (VAX) the input file and template file may be the SAME file, thus allowing new items to be created and compared to the input file being copied with them.

5.6 MAKE

The MAKE command is used to assign values to BUILDER parameters. The SHOW command is its converse-- anything you can MAKE can be SHOWN.

MAKE sets values to change views, coordinates, positions, and so on:

1. MAKE BUILDING NUMBER: 33 ; -- Assign a label to the building/fragment for identification.

2. MAKE BUILDING PLANE HORIZONTAL or VERTICAL; -- select the plane on which the die is located, i.e., from which extrusion will occur.

   These planes (horizontal or vertical) are the plane or X-Y plane, and a Section, or Elevation (Y-Z) plane.

3. MAKE BUILDING TOP or BASE: 25 ; -- Define where extrusion will start and stop. Measured from the die plane.

4. MAKE BUILDING MODELFACE or EXTRUDEDFACE XTILT or YTILT: -10 ;

   Specify how the modelface and extruded faces are to be tilted after extrusion takes place. This allows slanting rooves and splayed walls. You can show all tilting with the command SHOW BUILDING TILT;

   For example, to create a "chalet" roof effect, you could digitize the plan for a building envelope, extrude it to a certain height, and then "tilt the roof:"

   DO DIGITIZING;

   MAKE BUILDING PLANE HORIZONTAL;

   DIGITIZE BUILDING;

   DEFINE FLOOR: 0, ROOF: 300;

   MAKE BUILDING BASE: FLOOR, TOP: ROOF;

   MAKE BUILDING EXTRUDEDFACE XTILT: 20 ;

   These commands tilt the "top" (extruded) face at 20 degreed from the horizontal.

5. MAKE BUILDING LENGTH; -- Specify the number of points in the die.

6. MAKE BUILDING COORDINATE [: 3] VALUE: (22.5, 55);

   Make the value of the third coordinate of the die the tuple shown. The current value of the x coordinate and y coordinate are in the evaluable lexemes XVALUE and YVALUE. The tuple (XVALUE, YVALUE) is in the evaluable tuple VALUE.

7. MAKE POINT COORDINATE: 10; --

   This command is used to load up the evaluable lexemes XVALUE, YVALUE, and VALUE (the tuple) with the coordinates of die coordinate 10. Thus, to multiply the third coordinate of a building die by C, you would say:

   MAKE POINT COORDINATE : 3;

   MAKE BUILDING COORDINATE VALUE: VALUE * c ;

   Note that the last statement if equivalent to:

   MAKE BUILDING COORDINATE: 3 VALUE: C * (XVALUE, YVALUE);

   These commands are useful for designing dies based on equations, for modifying dies pointwise, or addressing individual points of a die.

   Scaling can also be done in this manner.

8. MAKE INPUT SCALEFACTOR: 4.5; -- Scale all input fragments/buildings by 4.5;

9. MAKE OUTPUT SCALEFACTOR: 10; -- Scale all fragments and buildings by a factor of 10.

10. MAKE STATION POINT: (10,20,300);

    Specify the observer's station point as a point in 3-space.

11. MAKE STATION HEIGHT: 300;

    Specify the Z (height) coordinate of the station point.

12. MAKE CENTEROFVISION POINT: (0,0,20);

    Specify the center of vision point in 3-space.

13. MAKE CENTEROFVISION HEIGHT: 20;

    Specify the Z coordinate of the center of vision point. (The height at which one is looking).

Note: the lexemes CENTEROFVISION POINT and HEIGHT, and STATION POINT and HEIGHT are all evaluable, that is, they may be used in expressions such as:

MAKE STATION HEIGHT: STATION HEIGHT + 20;

MAKE CENTEROFVISION POINT: (30, 40, STATION HEIGHT);

MAKE STATION POINT: 1.4* (STATION POINT-CENTEROFVISION POINT) + (30,10,50);

The DIGITIZE command allows these parameters to be specified graphically with the crosshairs.

Note that if the STATION HEIGHT and the CENTEROFVISON HEIGHT are the same, BUILDER will draw 2-point perspectives. Similarly, if 2 of the three coordinates for these points are symmetrically the same, a 1-point will be drawn.

14. MAKE FOCALRATIO: 1.6;

Specify the focal ration for the view. The default is 4 (telephoto), and the human eye is about 1.5-2.

The focal ratio is defined as the ratio of the distance to the picture plane and one half the width of the picture plane. Thus it is closely related to the viewing angle (width of cone of vision) and lens size of conventional photography.

NOTE: the DIGITIZE CENTEROFVISION POINT, DIGITIZE STATION POINT (in both plan and section) and then the MAKE FOCALRATIO commands all affect the perspective view: they are its parameters.

15. MAKE TERMINALSPEED: 960;

This sets the terminal speed in characters per second. Necessary on fast devices so BUILDER will not clear the screen too quickly.

16. MAKE HITHERCLIPPING DISTANCE:100;

MAKE YONCLIPPINGDISTANCE:6000;

Specify the hither and yon clipping distances for hidden surface views.

17. MAKE HIDDEN TOLERANCE: (0,1,1); -- specify the tolerance "box" for hidden surface processing in case there are overlapping/interpenetrating faces. Reduces hidden failure

cases--analagous to WHIRLPOOL tolerance.

18. MAKE HEIGHT or WIDTH SCREENSIZE: <value> ;

    Change the screen size. Don't do it!

19. MAKE INPUT TYPE POLYGON FILE;

    MAKE INPUT TYPE BUILDERFILE;

    This specifies the type of input file. A BUILDERFILE is a file created as an output file by BUILDER. A POLYGONFILE is the generic term for an ODYSSEY polygon, chain, or line file, which BUILDER can read.

    The POLYGONFILE will contain the cartographic representations of dies on a plane. They then must be assigned extrusion parameters and used to create 3-d forms. (These parameters include: BUILDING PLANE, TOP, BASE, TILT, etc.)

    POLYGONFILES almost always need to be scaled with the MAKE INPUT OFFSET and MAKE INPUT SCALEFACTOR commands.

    To see how to scale it, DRAW BUILDING: <n>; and then SHOW BUILDING COORDINATES.

    The builder and polygon files, in general, contain dies--lines, points, and polygons, that create 3-d forms. The polygon/chain/line files from ODYSSEY have no information on how to create 3-d forms from them-- how to extrude them. But BUILDERFILES do.

5.7 SHOW

The SHOW command is used to display parameters and status within BUILDER. "Anything you can make you can show". There are also :

1. SHOW BUILDING HEADER, COORDINATES;

2. SHOW BUILDING TILT;

3. SHOW FILESTATUS;

4. SHOW ANNOTATION;

5. SHOW DRAWINGMODE;

6. SHOW BUILDING LENGTH;

7. SHOW POINT COORDINATE; -- See which die point has been loaded into the evaluable BUILDER variables XVALUE, YVALUE, and VALUE.

8. SHOW VALUE, XVALUE, YVALUE;

   (Note that VALUE = (XVALUE, YVALUE) of the current POINT COORDINATE).

5.8 RESTORE

The RESTORE command is used to reset BUILDER parameters to their initial, or "default" values. You can restore most things you can make.

Default values are used so that you can "get" views and information without having to set a lot of parameters. You then change them to get the views (etc.) that you want.

5.9 MOVE

The MOVE command moves one point in a die, or else moves the whole building/fragment in 3-d. To edit a die, you must be in the same drawing-mode (i.e., DRAW PLAN or DRAW SECTION) as the die.

1. MOVE POINT; -- using the graphic crosshairs, move a point on the die somewhere.

2. MOVE BUILDING; -- using the crosshairs, move the whole structure somewhere. You can move fragments in plan or in section.

5.10 KILL

The command KILL POINT; is used to delete one point in the die with the crosshairs. If the point you specify is ambiguous, it will ask you which point you mean (which coordinate pair).

5.11 FLIP

The FLIP BUILDING; command switches the extrusion process to the other plane (section to plan, or vice versa), while keeping the mass centroid in 3-D in the same position.

5.12 CREATE

The CREATE command is used to create new files. BUILDER can create BUILDERFILES, which are in the standard BUILDER data base format for BUILDER to read and draw, or "image files" (plot files) containing graphics commands to specific devices. You specify a file name:

```
CREATE OUTPUT FILENAME: 'HOUSE.BAS';
CREATE IMAGE FILENAME:  'PLOT.DSK';
```

By leaving off the filename, you get the default file, or the last filename you specified. (Eg., "CREATE IMAGE;" will create a file called "IMAGE.PLT".)

5.13 END FILE Or END IMAGE

The END FILE command (END FILE;) is used to close an output file or an image file prior to writing (creating) another, or leaving BUILDER. Essentially, you CREATE an output file, COPY buildings into it from either the local buffer (editor/digitizer) or the input file, and then END the FILE.

In writing image files, you CREATE an image file with the command CREATE IMAGE;. Then, you DRAW to it--all plotting commands will go to the file, not to the screen. Then you END IMAGE;

5.14 COPY

The COPY command is used to copy the input file fragments/buildings, and/or the building currently in the local processing BUILDER buffer to the output file.

1. COPY BUILDING;

   Write the building currently in the buffer to the output file. Once written, its number (the BUILDING NUMBER) is incremented, but the building, die, extrusion parameters, and 3-d representation is unchanged, and it may now be

instanced, transformed, etc., and then COPIED again.

2. COPY REST;

   Copy the building currently in the buffer, and the "rest" of the input file to the output file. The buffer will contain the last building in the input file after this operation

3. COPY UNTIL BUILDING: 15;

   Copy the building in the buffer, and all buildings in the input file to the output file, until building 15 is encountered. Place building 15 in the buffer and return to the user for commands.

NOTE: the COPY command is valid only in DRAWING mode. In DIGITIZING mode, use the END BUILDING; command to write the current buffer fragment out to the output file:

END BUILDING;

Write the building currently in the buffer to the output file. Once written, its number (the BUILDING NUMBER) is incremented, but the building, die, extrusion parameters, and 3-d representation is unchanged, and it may now be instanced, transformed, etc., and then written out (with the END BUILDING; command) again.

5.15 REWIND

The REWIND command is NOT like HOMER's!! In BUILDER, input files are sequential, and you frequently want to get back to a building/fragment that was earlier in the file. The REWIND command will close the input file and re-open it at the begining. Thus, when in the middle of a file, you can draw the first building/fragment (and bring it into the buffer) with the commands:

REWIND; DRAW NEXT;

Similarly, to search the whole file for building/fragment 66, you say:

REWIND; DRAW BUILDING: 66;

Note that some commands perform an automatic rewind: any command with ALL in it, LIST INPUT, and HIDE.

5.16 ROTATE

The ROTATE command is used to rotate an entire fragment/building die about an arbitrary point. the command form is: ROTATE BY: 45;

The Angle specified may be negative or positive. BUILDER will compute the die's centroid, and plot it for you. You can then use it, or any point, to serve as the center of rotation.

5.17 LIST INPUT

The LIST INPUT; command lists the headers of all fragments/buildings in the input file.

5.18 PLOT

PLOT is exactly the same as DRAW, except that it instructs BUILDER to plot on the Houston Instruments plotter instead of the screen.

5.19 TRANSFORM

The TRANSFORM BUILDING; command is used to transform (rubber sheet) a building die with up to 10 control points. You digitize the input "window" control points, and then the points to which they are stretched. The MAKE TRANSFORMATION POWER: <m> ; command will affect the transformation in certain complex ways (see Morehouse's memo on "Rubber Sheeting with Inverse Power Constants", which addreses the problems of perverting the manifold).

The TRANSFORM BUILDING; command will take the original "window points" (up to 10) and stretch them towards theeir new positions. The die points near the window points (control points, in some terminologies) will be "pulled" along with them, as a function of the transformation power and their distance to the control points. This command is useful for cases where you want to "twist" a part of a die somewhere. It can lead to non-planar results.

5.20 INSERT

The INSERT SEGMENT; command is used in editing Dies. Using the crosshairs, the first and last points of the inserted segment are fitted to the nearest points in the previous die. The rest of the new segment points replace the old points they surround (topologically). The segment may have more or fewer points than the segment it replaces, and thus can either "add on" or "remove" a wing to a structure.

5.21 HIDE

This command is used to generate hidden surface perspective views.

The view is generated from the current station point, with the focal ratio and center of vision unchanged. The command is EXPERIMENTAL--it is still being debugged. The following commands influence the HIDE; command:

1. USE AED512; or USE T4027;

   Specifies the raster device to use.

2. MAKE HITHERCLIPPINGDISTANCE or YONCLIPPINGDISTANCE

   Specifies the hither and yon clipping planes

3. MAKE HIDDEN TOLERANCE: (dx, dy, dz);

   Specifies a tolerance "box" (topologically a ball) for data with overlaps.

5.22 USE

The use command specifies the device to draw on. To plot on the Houston Instruments plotter, simply employ the PLOT command wherever you would say DRAW. For raster devices, say:

USE T4027; -- use the Tektronix 4027 terminal

USE AED512; -- use the AED 512 raster terminal

5.23 QUIT

Exit from BUILDER with the QUIT; command.

6.0 BUILDER MACRO-OPERATIONS: PROGRAMMING IN BUILDER

BUILDER is actually a programming language. You can write "operations", or "macro-operations" which contain lists of commands for BUILDER to execute. These commands can be written either with the system text editor, or from within BUILDER.

The most useful commands pertaining to operations are:

1. BEGIN OPERATION -- start defining (typing a BUILDER "program", or operation).
2. END OPERATION; -- used to mark the end of an operation.
3. PERFORM OPERATION -- perform (do) a BUILDER operation that is previously defined.
4. IF / ELSE -- Conditional statement for operations.
5. IF/GOTO -- Conditional branch within an operation
6. LIST OPERATION -- display (type out) a previously defined operation.

6.1 BEGIN OPERATION

This command is used to start (define) an operation to BUILDER. You type in the operation, and end with a "control-Z" (The "control key" is like the "shift key" and is used to create special characters for special operations. You send the character "control-z" by holding down the "control" ("CTRL") key, and typing Z while "CTRL" is still depressed). The BEGIN OPERATION command can take the name of the operation (a filename) as its argument:

BEGIN OPERATION ; -- use the default, or last operation name.

BEGIN OPERATION: 'INSTANCE'; -- begin a named opeation.

6.2 END OPERATION

The END OPERATION; command MUST be contained within an operation. It may occur anywhere, and causes BUILDER to "pop" from the operation, back to the previous command level.

6.3 LIST OPERATION

The LIST OPERATION command is used to list (display, or type out) the operation you have just defined or executed, or else a perviously defined operation. The LIST OPERATION; command with no argument lists the last operation you defined (or the default operation, MACRO.MAC). If you have just executed (PERFORMED) an operation, then LIST will list it.

The argument to LIST OPERATION is the filename of a previously defined operation. LIST will display its contents.

LIST OPERATION;

LIST OPERATION : 'INSTANCE'; LIST OPERATION: 'REFLECT';

6.4 PERFORM OPERATION

The perform operation command causes BUILDER to execute a command procedure (operation). PERFORMS may be "nested" (on operation may PERFORM another).

There are three arguments to PERFORM. If any are unspecified, the operation is performed in the last way specified, or in the dafault way.

The parameters are: what operation to perform, what selection criteria to use for buildings/structures to be input to the procedure, and a list or range of buildings/fragments for which the operation will be performed. The input file is searched (sequentially, from the current position) for the specified set of buildings.

6.4.1 The Selection Mode -

You can perform operations just one time, for a "set" (list) of BUILDING NUMBERS (identifiers), or for a "range" of BUILDING NUMBERS (identifiers). These modes are specified as:

```
PERFORM OPERATION [: "name" ]  ONCE ;
                               FOR BUILDING SET [ : (20,32,1,4,55,6,12,3) ] ;
                               FOR BUILDING RANGE [: (1,100) ] ;
```

Note, as always, that the portions of commands in brackets ([,]) indicate optional phrases. If these portions left out, the operation is performed with the last parameter used, or else with the default values.

The following SHOW operations are useful:

1. SHOW OPERATION; -- display the current operation name

2. SHOW BUILDING SELECTIONMODE; -- show the mode for selecting buildings/ fragments on which the operation will be performed.

3. SHOW BUILDING SET; and SHOW BUILDING RANGE; -- Show the set or range of buildings to be used for the SELECTIONMODE: RANGE or SET modes.

6.4.2 PERFORM OPERATION ONCE -

The ONCE selection mode is useful for performing a command procedure one time only. Such procedures might set up views or open files for views, or contain a list of commands to be applied selectively.

6.4.3 PERFORM OPERATION FOR BUILDING SET Or RANGE -

The building/fragment SET or RANGE indicates a set (list) of building identifiers, or an inclusive "range" (min, max) for which the operation will be applied. This selection mode is most useful for specifying an operation or procedure which will be applied to at least several buildings/fragments in the input data base, for example: "make all the columns in the data base 4 feet shorter", "instance all the steps on both sides of the auditorium, and copy them to the output file."

6.5 Examples Of PERFORM Commands

```
PERFORM OPERATION :  'NAME' ONCE;

PERFORM OPERATION FOR BUILDING RANGE:  (1,20);

PERFORM OPERATION:  'SHIFT' FOR BUILDING SET:  (5,6,22,3);

PERFORM OPERATION:  'ROTATE' FOR BUILDING SET;

PERFORM OPERATION;
```

6.6 IF/ ELSE/ GOTO

The IF statement can be used in operations to specify conditionally performed sets of commands. the format is:

```
IF : <logical expression> ;
 <list of commands to be performed if the expression is true>
 [ELSE;
 <list of commands to be performed if the expression is false>]
 FIN;
```

The ELSE clause is optional. The FIN; ("end if statement") command is not. If statements may be nested (inside another IF clause). The list of commands to be evaluated when the expression is clause is true or else false may be any BUILDER commands, in the usual format, ending with semi-colons, etc. There may be as many as you like.

The logical expression is an expression which is either true or false: for example:

```
IF:  A = B;

IF:  BUILDING BASE < 100;

IF:  BUILDING TOP - BUILDING BASE > 1005;
```

The following operators can be used in ANY logical expression:

```
&       --      AND
%       --      OR
```

-- NOT

Example: IF: A=B & B + Building base > 50 ;

6.6.1 IF / GOTO -

There is also an IF/GOTO statement in BUILDER. The form is:

IF: <logical expression> GOTO LINE: "START" ;

Here, the "line" parameter is a text label. In BUILDER operations, Comments and Statement labels begin with a dollar-sign ($). The statemet label start would be somewhere within the operation as:

$ START:

Statement labels must be capitalized, in a line beginning with a dollar - sign, and IMMEDIATELY followed by a colon.

Remember to end your if statments with a FIN; This is crucial, since if the IF evaluates to FALSE, then no staments will be executed until a FIN (or an ELSE) is reached.

However, the statements WILL be checked for validity. If an error occurs, BUILDER will "pop" the macro operations, and return to interactive command mode.

6.7 The ECHO Command

Ordinarily, BUILDER will "echo", or type back the response to a command. Thus the command "Ma sta p: st p/2;" [i.e., make the station point half the vector that it is now] will be echoed as:

.-. MAKE STATION POINT : (34,556.5, 70.25) ;

You can turn the echo off at any point, with the ECHO OFF; command. ECHO ON; will turn it back on.

This is useful if you have an operation which draws, and you don't want to see the commands fly by you, overwriting the drawing.

6.8 Screen "Flashing" And Waiting.

The Command DRAW PERSPECTIVEVIEW or HIDE will erase the screen before drawing. DRAW SECTION, PLAN, or TEMPLATE will not, even if a perspective of the template is to be drawn (this is useful, for example, to compare the local buffer fragment/building, or the input file data base, in perspective to a template).

After any plan, section, hidden surface, or perspective drawing, BUILDER waits for a carriage return before asking for more commands with the BUILDER "prompt" (the question mark: "?"). However, when performing a macro-operation, or when plotting on the plotter, BUILDER does not wait, since in a repetitive drawing macro this would require frequent intervention to proceed, and since the prompt will not mar a plot.

6.9 Examples Of BUILDER Macro-operations.

_DNA1:[DONALD.HOOTI.BUILDING]BEAU.DAT;1

```
$ MACRO-OPERATION BEAU
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED  9DEC81
$
do dr;
open in fi:'dn0:[studio.beau]theatre.bas';
ma s p:(730,701,247), c p:(426,397,87), fo:1.5;
end op;
```

_DNA1:[DONALD.HOOTI.BUILDING]BRUCE.DAT;1

```
do dr;
per op:'setr';
op in fi:'dn0:[studio.bruce]theatre.bas';
end op;
```

_DNA1:[DONALD.HOOTI.BUILDING]CIRCLE.DAT;4

```
$ MACRO-OPERATION CIRCLE
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 23NOV81
$
Define Numpt:0;
Define R2: r*r;
Define Delta: 2*r/Numpts;
Define XX: -r - Delta;
$ LOOP:
        Define Numpt: Numpt + 1;
        Define XX: XX + Delta;
        Define YY: (R2 - XX*XX)^(.5);
        Make Building Coordinate: Numpt
                Value: (XX + r, YY + r);
```

```
        If: XX < r Goto Line: 'loop';
         Fin;
 End Op;
```

_DNA1:[DONALD.HOOTI.BUILDING]COPY.DAT;1

```
 $ MACRO-OPERATION COPY
 $ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 18DEC81
 $
 DO DR; REW; DR NE; COPY REST;
 DO DIG; END OP;
```

_DNA1:[DONALD.HOOTI.BUILDING]CUBE.DAT;1

```
 $ MACRO-OPERATION CUBE
 $ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 16DEC81
 $
 $ MAKE A CUBE AT COORDINATES: (X1,Y1) OF HEIGHT "CUBE"
 $
 MA BU LEN:4, BASE:Z1, TOP:Z1 + CUBE;
 MA BU COORD:1 VAL:(X1,Y1);
 MA BU COORD:2 VAL:(X1+CUBE,Y1);
 MA BU COORD:3 VAL:(X1+CUBE,Y1+CUBE);
 MA BU COORD:4 VAL:(X1,Y1+CUBE);
 END OP;
```

_DNA1:[DONALD.HOOTI.BUILDING]DEMO.DAT;2

```
 $ MACRO-OPERATION DEMO
 $ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 22NOV81
 $
 DEFINE XMIN:200,XMAX:700,F:.009;
 DEFINE NUMPTS:20,NUMPT:0, YOFF: 0,XOFF:450,;
 PERFORM OPERATION:'BUILD:PARA' ONCE;
 DRAW PLAN BU;
 END OP;
```

_DNA1:[DONALD.HOOTI.BUILDING]DRAW.DAT;1

```
 $ MACRO-OPERATION SELECT
 $ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 25JAN82
 $
 dr bu; end op;
```

_DNA1:[DONALD.HOOTI.BUILDING]ENO.DAT;1

```
 $ MACRO-OPERATION ENO
 $ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 11DEC81
 $
 DO DR; O I F:'ENO.BAS'; MA FO:7, S H: S H*1.5;
 END OP;
```

_DNA1:[DONALD.HOOTI.BUILDING]HEADER.DAT;1

Examples of BUILDER macro=operations.

```
$ MACRO-OPERATION HEADER
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 25JAN82
$
SHO BU HEAD;
END OP;
```

_DNA1:[DONALD.HOOTI.BUILDING]LAND.DAT;8

```
$ MACRO-OPERATION LAND
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 20JAN82
$
do dr; o i f:'popko.bas';
ma fo:2, s p:(24,184,200), c p:(754,579,6);
define dif: s p - c p;
ma s p: c p + dif * 3;
define dif: s p - c p;
define inc: dif/numinc;
DEFINE ZINC: STA H/NUMINC;
$ numinc = number of views for landing
dr per all;
$ LOOP:
make sta p: sta p - inc;
dr all;
IF: STA H- ZINC > 0. GOTO LINE :'LOOP';
 Fin;
  .      End op;
```

_DNA1:[DONALD.HOOTI.BUILDING]PARA.DAT;4

```
$ MAKE PARABOLA. NEED
$
DEFINE XX: XMIN;
DEFINE DELTA: (XMAX - XMIN) /NUMPTS;
$ LOOP:
DEFINE NUMPT: NUMPT +1;
DEFINE T : XX - XOFF;
DEFINE YY: (T*T) * F - YOFF;
MAKE BUILDING COORD: NUMPT VALUE: (XX,YY);
DEFINE XX: XX + DELTA;
IF: XX <= XMAX GOTO LINE: 'LOOP';
FIN;
END OP;
```

_DNA1:[DONALD.HOOTI.BUILDING]PLOT.DAT;3

```
$ MACRO-OPERATION PLOT
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 24NOV81
$
create plot;
draw all;
end plot;
end op;
```

_DNA1:[DONALD.HOOTI.BUILDING]PUB.DAT;4

```
$ MACRO-OPERATION PUB
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 14JAN82
$
do dr;
dr no c;
o i f:'[donald.hooti.building]rick.bas';
ma c p:(485,486,102);
ma s p:(2000,453,380);
end op;
```

_DNA1:[DONALD.HOOTI.BUILDING]RENUMBER.DAT;1

```
$ MACRO-OPERATION RENUMBER
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 18DEC81
$
DEF N: N + 1;
MA BU NUM:N;
COPY BU;
END OP;
```

_DNA1:[DONALD.HOOTI.BUILDING]RICK.DAT;1

```
$ MACRO-OPERATION RICK
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED  9DEC81
$
do dr;
```

_DNA1:[DONALD.HOOTI.BUILDING]SCALE.DAT;14

```
$ MACRO-OPERATION SCALE
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 23NOV81
$
define numpt:1;
$ START:
       Make Point Coord: numpt;
       Make Building Coord Val:
              (Tx*Xvalue - Ox, Ty*Yvalue - Oy);
       Define Numpt: Numpt + 1;
       If: Numpt <= Building Length Goto Line:'START';
       Fin;
End Op;
```

_DNA1:[DONALD.HOOTI.BUILDING]SELECT.DAT;2

```
$ MACRO-OPERATION SELECT
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 25JAN82
$
PER OP: OP; END OP;
$ PERFORM THE OP FOR DEFINED VAR "OP"
```

_DNA1:[DONALD.HOOTI.BUILDING]SET.DAT;1

Examples of BUILDER macro=operations.

```
 $ MACRO-OPERATION SET
 $ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED  1DEC81
 $
 make c p:(202,626,289), s p:(560,626,289), fo:1.6;
 end op;
```

_DNA1:[DONALD.HOOTI.BUILDING]SETR.DAT;3

```
 $ MACRO-OPERATION SETR
 $ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED  1DEC81
 $
 open in fi:'dn0:[studio.rick]THEATRE.bas';
 ma fo:1.6, s p:(207,750,111), c p:(139,58,81);
 end op;
```

_DNA1:[DONALD.HOOTI.BUILDING]SHRINK.DAT;1

```
 $ MACRO-OPERATION SHRINK
 $ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 18DEC81
 $
MA BU BA: BU BA+ 1, BU TOP: BU TOP -1;
COPY BU; END OP;
```

_DNA1:[DONALD.HOOTI.BUILDING]STACK.DAT;3

```
 $ MACRO-OPERATION STACK
 $ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 16DEC81
 $
 $ CREATE "N" CUBES OF HEIGHT "CUBE".
 $ TO USE THIS: DEFINE N AND CUBE, THEN
 $ CREATE AN OUTPUT FILE. THEN SAY "PERFORM OPERATION :'STACK';"
 $
 DEFINE MAX: N*CUBE;
 DEF X1: 0;
 $ XLOOP:
  DEF X1: X1 + CUBE;
 DEF Y1: 0;
 $ YLOOP:
  DEF Y1:Y1 + CUBE;
 DEF Z1: 0;
 $ ZLOOP:
  DEF Z1:Z1 + CUBE;
    PERFORM OPERATION : 'CUBE';
    COPY BUILDING;

    IF : (Z1 < MAX) GOTO LINE: 'ZLOOP';
    FIN;
    IF : (Y1 < MAX) GOTO LINE :'YLOOP';
    FIN;
    IF : (X1 < MAX) GOTO LINE: "XLOOP";
    FIN;
 END OP;
```

_DNA1:[DONALD.HOOTI.BUILDING]SWAP.DAT;6

```
$ MACRO-OPERATION SWAP
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 22NOV81
$
DEF NUMPT:1;
$ A:
MA PO CO: NUMPT;
MA BU CO VAL: (YVAL, XVAL);
DEF NUMPT : NUMPT + 1;
IF : NUMPT <= BUILDING LENGTH GOTO LI:'A';
FIN;
END OP;
```

_DNA1:[DONALD.HOOTI.BUILDING]SWITCH.DAT;1

```
$ MACRO-OPERATION BUILD:SWITCH
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 28JAN82
$
define here: stat poi;
make stat poi: cent poi;
make cent poi: here;
delete here;
end op;
```

_DNA1:[DONALD.HOOTI.BUILDING]TEMPLATE.DAT;1

```
$ MACRO-OPERATION TEMPLATE
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 18DEC81
$
DO DR; COPY BU; END FI;
O I F: O F, TEM FI: O F;
DO DIG; END OP;
```

_DNA1:[DONALD.HOOTI.BUILDING]WEED.DAT;1

```
$ MACRO-OPERATION WEED
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 14JAN82
$
if: building length>2 ;
       Copy bu;
       Fin;
End op;
```

_DNA1:[DONALD.HOOTI.BUILDING]WHOLE10.DAT;1

```
$ MACRO-OPERATION WHOLE10
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 20DEC81
$
DO DR;
MA HID TOL:(0,0,1);
OP IN FI:'WHOLE10.BAS';
MA S P:(301,775,1025), C P:(302,194,6), FO:2;
END OP;
```

Examples of BUILDER macro=operations.

_DNO:[STUDIO.STEVE]BOTTOM.DAT;1

```
$ MACRO-OPERATION BOTTOM
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED  2FEB82
$
ma bu base: 103 , top: 103 + c * 35;
end op;
```

_DNO:[STUDIO.STEVE]DEPTH.DAT;3

```
$ MACRO-OPERATION DEPTH
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 26JAN82
$
ma bu ba:near, top:far; cop bu; end op;
```

_DNO:[STUDIO.STEVE]FULL.DAT;1

```
$ MACRO-OPERATION FULL
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED  2FEB82
$
ma bu n:-bu n;
ma bu ba:103 , top: 103 + c * 69;
end op;
```

_DNO:[STUDIO.STEVE]INSTANCE.DAT;1

```
$ MACRO-OPERATION INSTANCE
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 26JAN82
$
make bu ba:39*c, top:51*c;
copy bu;
ma bu ba:147*c, top:159*c;
make build num: build num+100;
copy bu;
end op;
```

_DNO:[STUDIO.STEVE]MERGE.DAT;1

```
$ MACRO-OPERATION MERGE
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 26JAN82
$
$ merge the defined name "which" onto the output
$
open in fi: which;
dr sec next;
copy rest;
end op;
```

_DNO:[STUDIO.STEVE]MIRROR.DAT;1

```
$ MACRO-OPERATION MIRROR
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 26JAN82
$
make build ba: max - build ba;
```

```
ma build top: max - build top;
copy bu;
end op;
```

_DNO:[STUDIO.STEVE]MOVE.DAT;1

```
$ MACRO-OPERATION MOVE
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 26JAN82
$
MOVE BU; DR BU; END OP;
```

_DNO:[STUDIO.STEVE]OFFSET.DAT;1

```
$ MACRO-OPERATION OFFSET
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 26JAN82
$
MAKE INPUT OFF:(688,0); END OP;
$ TO RECTIFY THE REFLECTION...
```

_DNO:[STUDIO.STEVE]REFLECT.DAT;2

```
$ MACRO-OPERATION REFLECT
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 26JAN82
$
def numpt:1;
$ A:
make point co: numpt;
ma bu co val:(-x,y);
def numpt: numpt + 1;
if : numpt <= building length goto line:'A';
 fin;
end op;
```

_DNO:[STUDIO.STEVE]SECTION.DAT;1

```
$ MACRO-OPERATION SECTION
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 26JAN82
$
do dr; ma in ty pol;
ma in off:(-521,0), sc:40.5;
o i f:'section.ldg';
end op;
```

_DNO:[STUDIO.STEVE]SREFLECT.DAT;1

```
$ MACRO-OPERATION SREFLECT
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 26JAN82
$
PER OP:'REFLECT' ONCE;
COPY BU;
END OP;
```

_DNO:[STUDIO.STEVE]SWOPY.DAT;1

```
 $ MACRO-OPERATION SWOPY
 $ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 26JAN82
 $
 $ swap and copy...
 per op:'build:swap' once;
 copy bu;
 end op;

_DNO:[STUDIO.STEVE]TOP.DAT;1

 $ MACRO-OPERATION TOP
 $ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED  2FEB82
 $
 make bu n: - bu n;
 make bu base: 103 + c * 45, top: 103 + c* 69;
 end op;
```

7.0 LISTS OF BUILDER COMMANDS AND LEXEMES

In this section, all user input is in lower case, and all BUILDER echos and answers is in upper case.

```
$ Builder
--- WELCOME TO BUILDER!
$ List the valid verbs in intial mode (after DO)
do ?
===  DRAWING
===  DIGITIZING
dr;
.-. DO DRAWING
$ list the valid verbs in DRAWING mode
?
===  DO
===  DRAW
===  DIGITIZE
===  DEFINE
===  DELETE
===  OPEN
===  MAKE
===  MOVE
===  CREATE
===  COPY
===  SHOW
===  END
===  ECHO
===  ELSE
===  QUIT
===  RESTORE
```

```
===  REWIND
===  ROTATE
===  LIST
===  TRANSFORM
===  KILL
===  INSERT
===  IF
===  FLIP
===  FIN
===  HIDE
===  USE
===  BEGIN
===  PERFORM
===  PLOT
$ list the things you can make in DRAWING mode
make ?
===  OUTPUT
===  CENTEROFVISION
===  STATION
===  TRANSFORMATION
===  TERMINALSPEED
===  INPUT
===  FOCALRATIO
===  HIDDEN
===  HITHERCLIPPINGDISTANCE
===  HEIGHT
===  BUILDING
===  BASEDRAWING
===  POINT
===  YONCLIPPINGDISTANCE
===  WIDTH
 $ List the BUILDING attributes you can make
building ?
===  MODELFACE
===  COORDINATE
===  EXTRUDEDFACE
===  LENGTH
===  TOPELEVATION
===  BASEELEVATION
===  PLANE
===  NUMBER
do dig;
$ list the valid verbs in DIGITIZING mode
.-. DO DIGITIZING
?
===  DO
===  DRAW
===  DIGITIZE
===  DEFINE
===  DELETE
===  OPEN
===  MAKE
===  MOVE
===  CREATE
```

```
===  SHOW
===  END
===  ECHO
===  ELSE
===  QUIT
===  RESTORE
===  ROTATE
===  LIST
===  TRANSFORM
===  KILL
===  INSERT
===  IF
===  FLIP
===  FIN
===  HIDE
===  USE
===  BEGIN
===  PERFORM
===  PLOT
$ list all the lexemes (words) in BUILDER
define ?
===  < NEW WORD NOT IN FOLLOWING LIST >
===  DO
===  DRAW
===  DRAWING
===  DRAWINGMODE
===  DRAWINGPLANE
===  DIGITIZE
===  DIGITIZING
===  DEFINE
===  DELETE
===  OPEN
===  OPERATION
===  ON
===  ONCE
===  OFF
===  OFFSET
===  OUTPUT
===  OUTPUTFILE
===  MAKE
===  MOVE
===  MODELFACE
===  CREATE
===  COPY
===  COORDINATES
===  COORDINATE
===  CONNECTINGENDSEGMENTS
===  CENTEROFVISION
===  SHOW
===  STATION
===  SCREENSIZE
===  SCALEFACTOR
===  SEGMENT
===  SECTION
```

=== SET
=== SELECTIONMODE
=== SPACING
=== END
=== ECHO
=== ECHOLEVEL
=== ELSE
=== ELEVATION
=== EXTRUDEDFACE
=== QUIT
=== RESTORE
=== REST
=== REWIND
=== ROTATE
=== RANGE
=== LIST
=== LINE
=== LENGTH
=== LABELS
=== TRANSFORM
=== TRANSFORMATION
=== TRUE
=== TERMINALSPEED
=== TEMPLATEFILE
=== TEMPLATE
=== TOPELEVATION
=== TOLERANCE
=== TYPE
=== T4027
=== TILT
=== KILL
=== INSERT
=== INPUT
=== INPUTFILE
=== IF
=== IMAGE
=== IMAGEFILE
=== FLIP
=== FIN
=== FILENAME
=== FILE
=== FILESTATUS
=== FOCALRATIO
=== FOR
=== HIDE
=== HIDDEN
=== HITHERCLIPPINGDISTANCE
=== HEADER
=== HEIGHT
=== HORIZONTAL
=== USE
=== UNTIL
=== BEGIN
=== BUILDING

=== BUILDINGPLANE
=== BUILDERFILE
=== BASEELEVATION
=== BASES
=== BASEDRAWING
=== BY
=== PERFORM
=== PERSPECTIVEVIEW
=== PLOT
=== PLANE
=== PLAN
=== POWER
=== POINT
=== POLYGONFILE
=== GOTO
=== GRID
=== ACTION
=== ALL
=== AWAYFACES
=== ANNOTATION
=== AED512
=== NOTHING
=== NO
=== NUMBER
=== NEXT
=== VALUE
=== VERTICAL
=== XVALUE
=== XTILT
=== YVALUE
=== YTILT
=== YONCLIPPINGDISTANCE
=== WIDTH

8.0 COMMENTS, CREDITS, AND REFERENCES

BUILDER was written as a research tool towards geometric modeling and as a design tool for previsualizing interior space. It was designed and implemented by Bruce R. Donald at the Laboratory for Computer Graphics and Spatial Analysis of the Harvard Graduate School of Design.

There are two important references for BUILDER users, both by Donald:

"Computing the Topology of 3-Dimensional Forms" (80 pp).

"A Survey of Internal and Input Representations for Geometric Modeling" (15 pp.)

Changes and Updates:

A list of new and changed BUILDER commands.

1. ASSIGN PROPERTY : "NAME" STRING "COLUMN";
   ASSIGN PROPERTY: 'SHEAR RESISTANCE' SCALAR: 44.5/ C;
   ASSIGN PROPERTY: 'CONFIGURATION SPACE REFERENCE' VECTOR: (X,Y,Z,THETA);

2. GET PROPERTY:'NAME';
   Get the specified property. If it does not exist, set EXISTS to false. When the property does exist, set EXISTS to true, and place the value of the property, according to type, in the evaluable BUILDER variables SCALAR, STRING, or VECTOR, depending on their type. These variables may be used in expressions and if statements.

3. IF: (EXISTS) & STRING = "DOOR";
   (EXISTS is an evaluable lexeme set to true if the property exists).

4. SHOW FRAGMENT PROPERTIES;

5. MAKE DIE DIMENSIONALITY: 1 or 2;

6. STORE MEMORY: n
   RECALL MEMORY: n; Where n is an integer 1-10 (like a pocket calculator).

7. MAKE PLOTTERWIDTH: <inches> ;
   specify the width of the hi plotter.

8. DRAW [NO] BOX;

9. DRAW [NO] CLEARSCREEN;

10. HIDE QUICKLY;
    (Poor man's hidden surface algorithm on the raster devices).
    HIDE; should , with the HIDDEN TOLERANCE set reasonably, and the HITHER and YONCLIPPINGDISTANCE chosen correctly, produce a quality (correct) hidden surface view on the raster devices.

11. SHOW DIE COORDINATES;

12. FRAGMENT can be used instead of, or interchangably with, BUILDING.

13. Once you define (DEFINE) a variable, you have declared its type and given it a value. Subsequently, you should MAKE it, unless you want to redefine its type or length. In other words, MAKE "binds", or assigns value. DEFINE declares type and length, and then does a MAKE for the value you give.

Advanced Commands:

For Shaded views on the AED 512 terminal:

MAKE SHADES: 50;

Controls the number of grey-tone shades the AED terminal will use for value delineation (shading) of surfaces.

MAKE SUNPOSITION: (2000, 500,3000);

Specifies a new position for the light source which illuminates the surfaces.

To position fragments at arbitrary angles and positions:

ASSIGN PROPERTY:'ORIENTATION' VECTOR: (20,-10, 45);

BUILDER computes the center (or centroid) of the fragment. It will rotate the object successively about 3 axes, the x,y, and z starting at the fragment's centroid.

The above orientation will tilt the fragment 20 degrees about the x axis, -10 about the y, and 45 about the z.

This is really pretty intuitive when you use it, and lets you position the fragments in any position whatsoever.

Manipulating Properties in the BUILDER Data Base

Internal Memo (Not for Distribution)

Bruce Donald
Lab for Computer Graphics
GSD
February 18, 1982

Draft - Comments to author

Any "fragment", or entity in the BUILDER data base, can be assigned general "properties" or attributes. These attributes consist of a property name, which is a text string, and a property "value", which can be a scalar, vector, or another text string. The ASSIGN PROPERTY command is used:

```
ASSIGN PROPERTY: 'NAME'  STRING: "COLUMN";

assign property:'thickness' scalar: 45;

assign property:'origin' vector: (xvalue, yvalue);
```

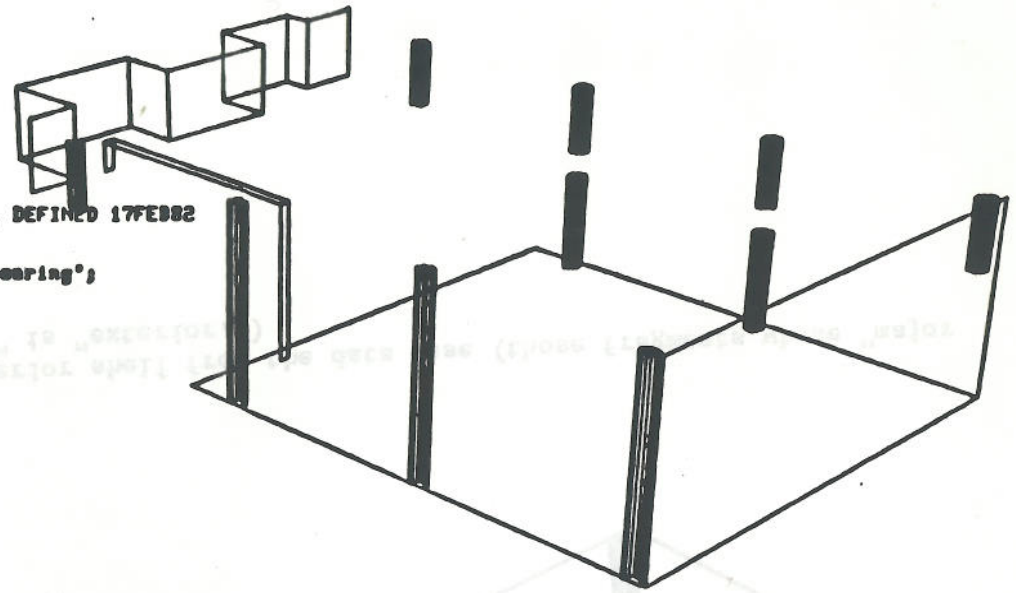
Once a data base with properties has been obtained or created, you can "get", or look at these properties. They are evaluable (in other words, they can be used in expressions) to create or select subsets of the data base.

You "get" (retrieve) a property by its name. The GET command will set the BUILDER variable EXISTS to "true" ("NOW SET") if the property exists. It will also retrieve the value of the property, and place it in the appropriate variable, STRING, VECTOR or SCALAR, depending on its type.

```
GET PROPERTY:'NAME';
 IF: Exists & String = "column" ;
   ...

GET PROPERTY:'span';
 If: Exists & scalar = 25;
  ...
   Get Property:'thickness';
     If:Exists & scalar = 50;
        ....
```

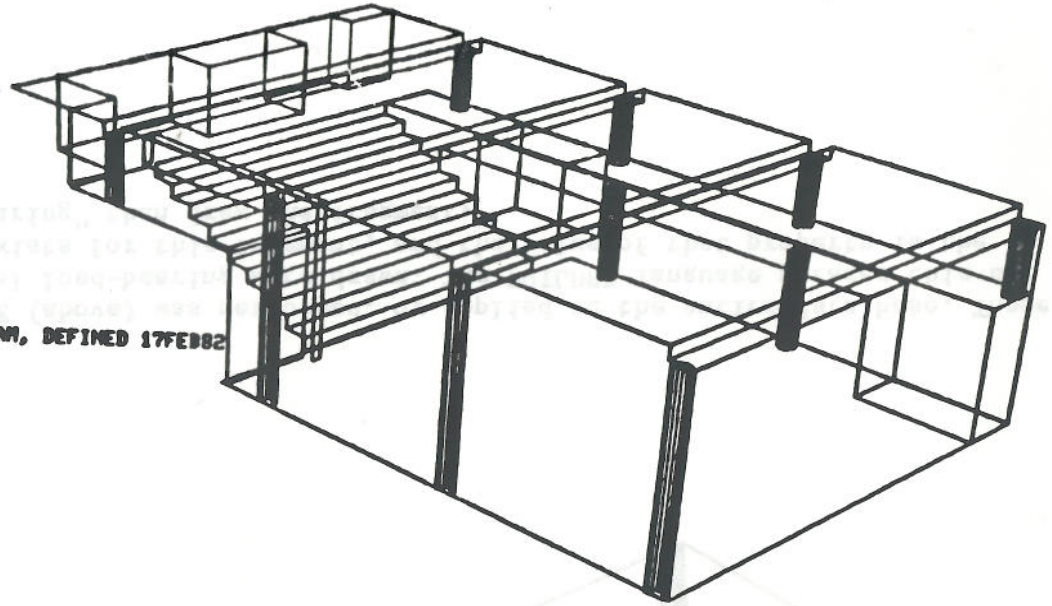
You can look at all the properties for a fragment with the SHOW FRAGMENT PROPERTIES; command.

```
brfore op;

?

list op;
$ MACRO-OPERATION RETRIEVE
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 17FEB82
$
get property:'bearing';
if: exists & string = 'vertical load-bearing';
      draw fragment;
 fin;
end op;
?
```



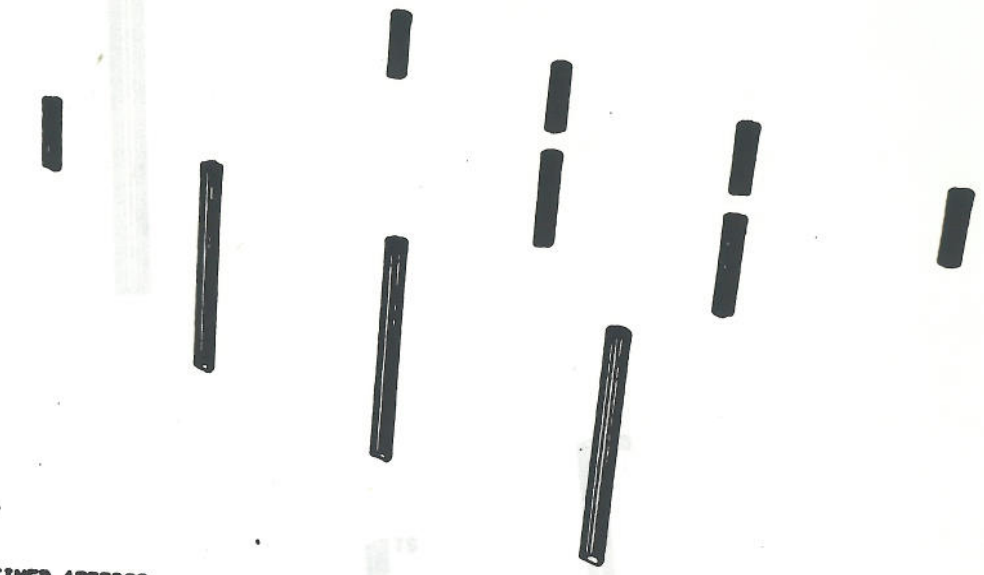
The macro-operation RETRIEVE (above) was performed, or applied to the entire data base. Those fragments which were vertical load-bearing were drawn. The BUILDER language phrases this as:
"If the property BEARING exists for this fragment, and the value of that property is the string "Vertical load-bearing" then draw the fragment."

```
p;
?
?
list operation;
$ MACRO-OPERATION RETRIEVE
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 17FEB82
$
get property:'major connectivity';
if: exists & string = 'exterior';
      draw fragment;
 fin;
end op;
?
```



Draw the exterior shell from the data base (those fragments whose "major connectivity" is "exterior.")

```
rfore op;

?

?

list op;
$ MACRO-OPERATION RETRIEVE
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 17FEB82
$
get property:'name';
if exists & string = 'column';
        draw fragment;
 fin;
end op;
?
```



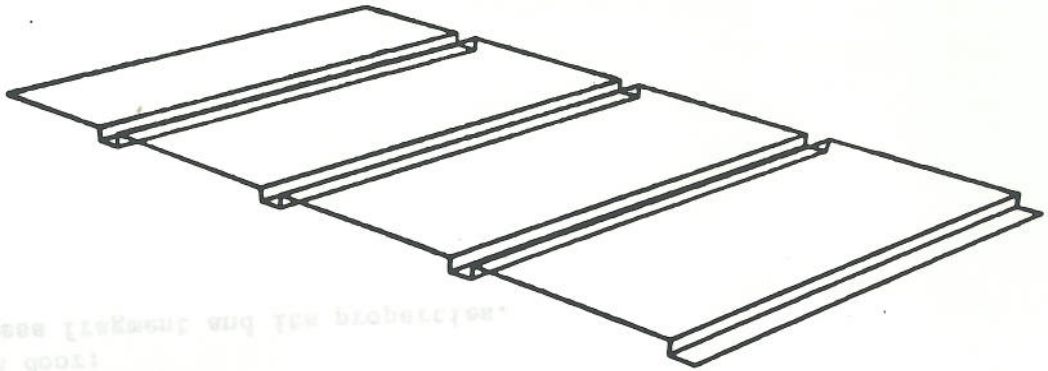
Macro-operation to draw all the columns in the data base.

```
>p;
```


```
?
dr sec fragment:11E;
?
sho fragment properties;
DIE ORIGIN :          (610,400)
SPAN :          -200
NAME      :  COLUMN
BEARING   :  VERTICAL LOAD-BEARING
FUNCTION      :  VERTICAL LOAD-BEARING
MAJOR CONNECTIVITY    :  EXTERIOR
?
```

The properties of a column.

```
  fr:113;
.-. DRAW FRAGMENT : 113


?

sho fr pr;
.-. SHOW FRAGMENT PROPERTIES
DIE ORIGIN :        (910.0001,304)
SPAN :        -529
NAME      :  ROOF
MAJOR CONNECTIVITY    :  EXTERIOR
BEARING   :  HORIZONTAL LOAD-BEARING
?
```



Display the properties of the roof fragment.

```
dr frag;
.-. DRAW FRAGMENT
```

```
?

sho frag pr;
.-. SHOW FRAGMENT PROPERTIES
DIE ORIGIN :          (713,103)
SPAN :          -5
NAME       :  DOOR
FUNCTION       :  EGRESS
?
```

A door:
An egress fragment and its properties.

```
^R
perform op for fragment ra!(0,170);
.-. PERFORM OPERATION FOR
.-. FRAGMENT
.-. RANGE : (0,170)
```

```
?

list op;
$ MACRO-OPERATION RETRIEVE
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 17FEB82
$
get prop!'function';
if: exists & string = 'egress';
        draw fragment;
fin;
end op;
?
```

Draw all the fragments whose function is "egress."

```
tw;
.-. PEWIND
?

sho p.v; per op;
.-. SHOW P : "FUNCTION"
.-. SHOW V: "EGRESS"
```

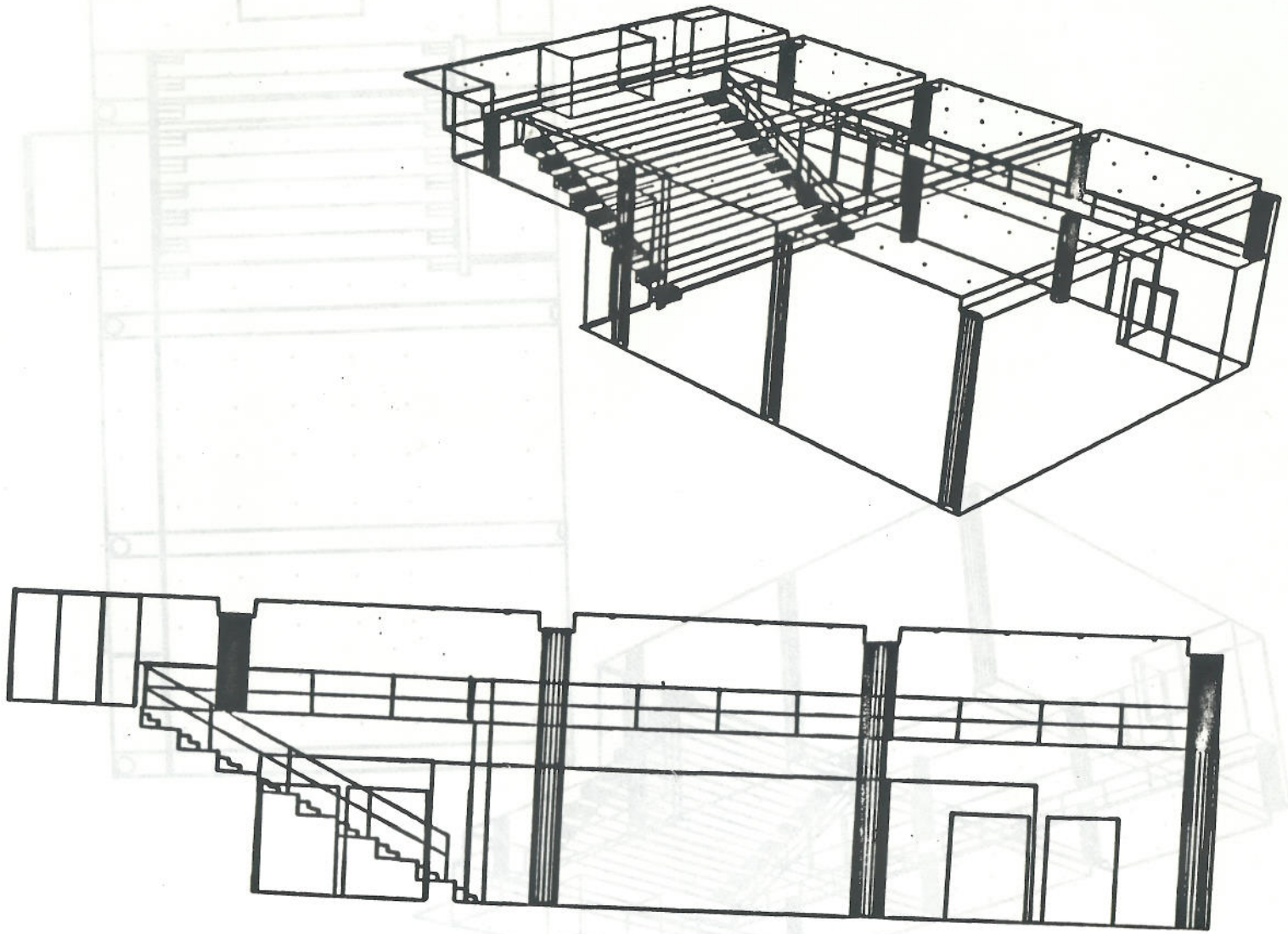

```
?
```


```
?

list op;
$ MACRO-OPERATION GENERAL
$ FOR THE BUILDER 3-D DISPLAY PROGRAM, DEFINED 18FEB82
$
get prop'p;
if:exists & string = v;
        draw fragment;
 fin;
end op;
?

draw plan fragment:140;
```

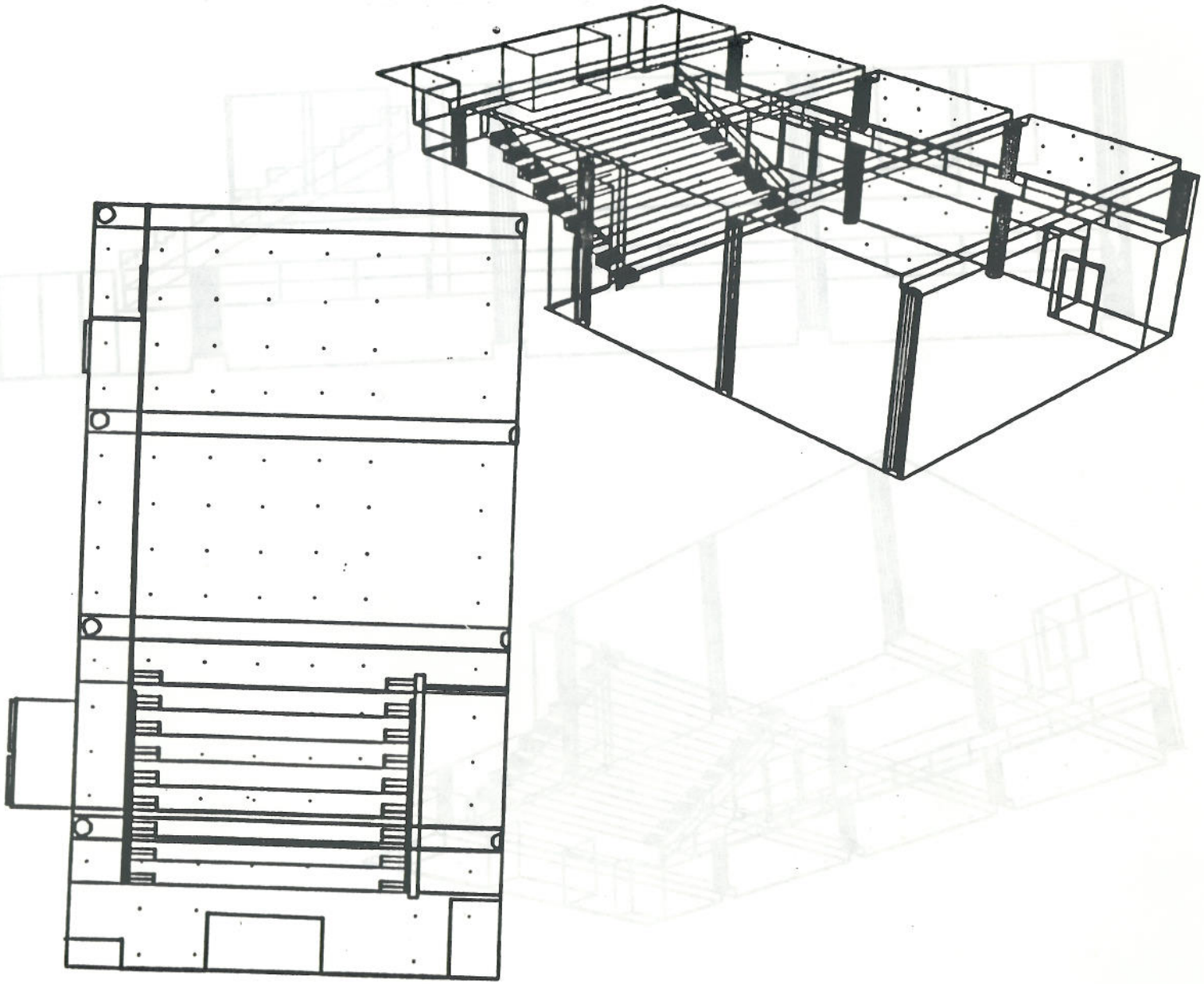

```
?

sho fragment properties;
DIE ORIGIN :          (194,101)
SPAN :      -6
NAME     :  DOOR
FUNCTION     :  EGRESS
?
```

A general macro-operation that tests for the existence of property "P" and draws fragments for which property "P" has string value"V".

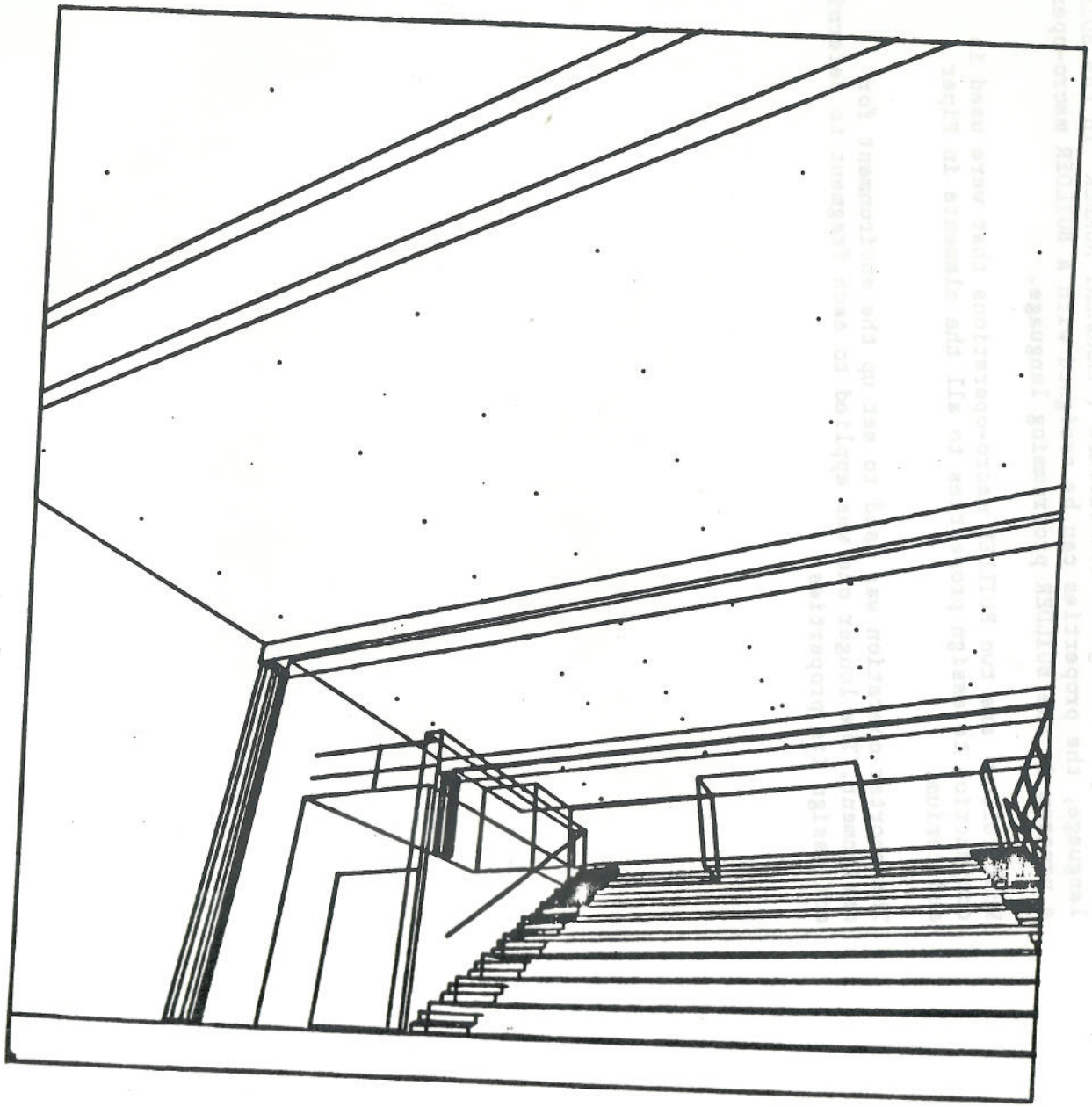
Below, one fragment is drawn and its properties displayed.



A section and near-axonometric of the entire data base.



A plan and near-axonometric of the entire data base.



An interior perspective view of the data base.

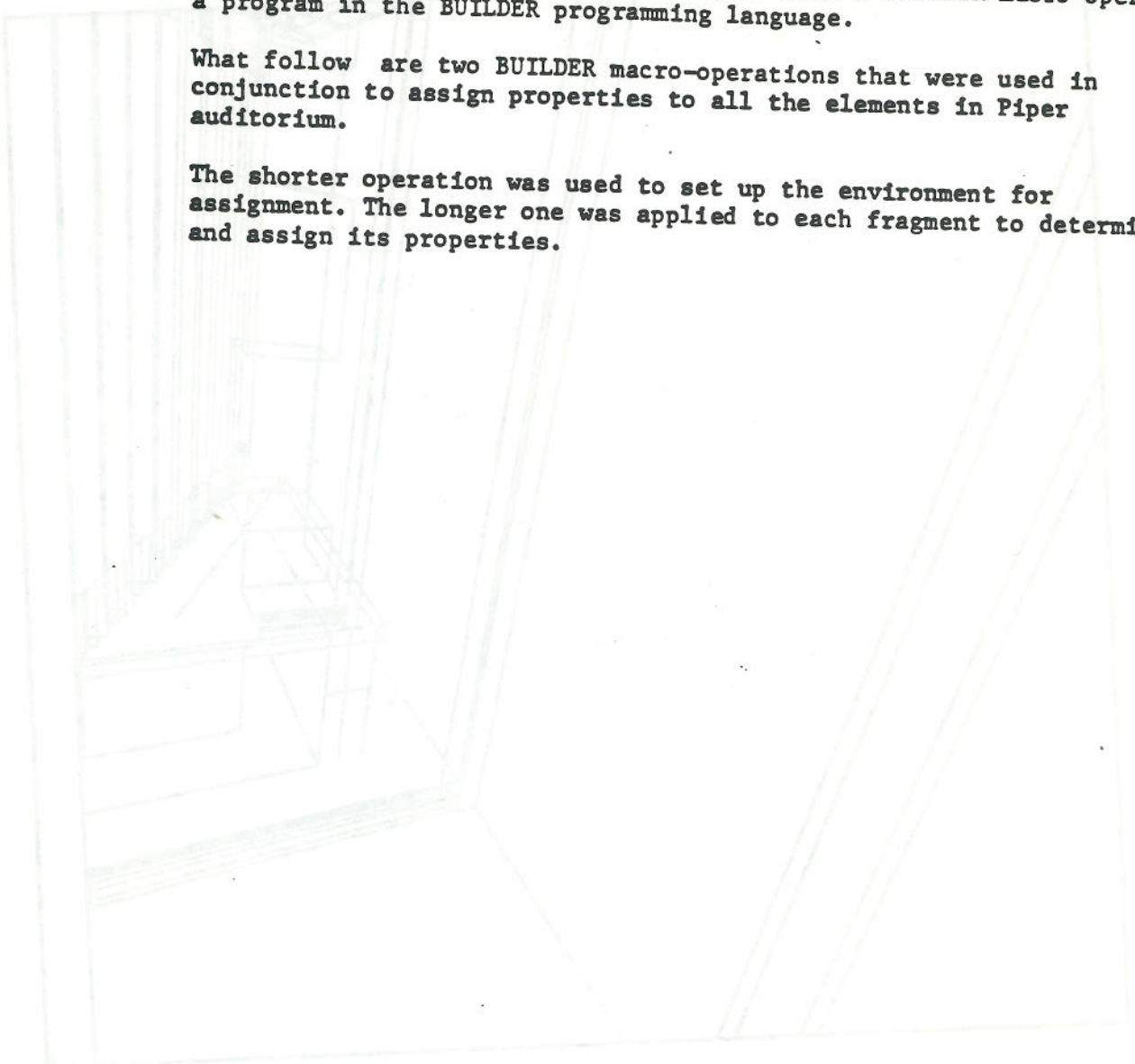
Appendix I

No fragment has properties unless they are explicitly specified, or unless an external program computes and assigns them in the BUILDER data base.

However, given the metric information that every fragment has, and by knowing the fragment identifiers ("FRAGMENT NUMBER" in the BUILDER language, the properties can be assigned with a BUILDER macro-operation--a program in the BUILDER programming language.

What follow are two BUILDER macro-operations that were used in conjunction to assign properties to all the elements in Piper auditorium.

The shorter operation was used to set up the environment for assignment. The longer one was applied to each fragment to determine and assign its properties.



```
$ assign properties to piper
$
def n: fragment number;
define light: n<99;
define door: n=99 % n=100 % n=140 % n=141 % n=166 % n=165;
define pipe: n=101 % n=118 % n=119 % (n>141 & n<165);
define column: n>102 & n<113;
define step: n>120 & n<138;
make point coordinate:1;
assign prop:'die origin' vector: value;
make span: fragment top - fragment base;
 if: span < 0;
        make span: -span;
  fin;
assign prop: 'span' scalar: -span;
$
define other: #(light % door % pipe % column % step);
if:light;
        assign prop:'name' str:'light';
        ass prop:'Major Connectivity' str:'ceiling';
 else;
 if:door;
        assign prop:'name' str:'door';
        assign prop:'function' str:'egress';
 else;
 if: (pipe) ;
        assign prop:'name' str: 'pipe railing';
        make fx: xvalue;
        make fy: yvalue;
        make point coord:fragment len;
        make dx: fx - xval; make  dy: fy-yval;
        make len: (dx*dx + dy*dy) ^ (.5);
        assign prop:'length' scalar:len;
 else;
 if:column;
        assign prop:'name' str:'column';
        assign prop:'bearing' str:'vertical load-bearing';
        assign prop:'function' str:'vertical load-bearing';
        ass prop:'Major Connectivity' str:'exterior';
 else;
 if: step;
        ass prop:'name' str:'step (small)';
        ass prop:'Major Connectivity' str:'large step contours';
$
fin; fin; fin; fin; fin;
$ special cases (one of a kind)
if: other;
if: n = 139;
        ass prop:'name' string: 'entrance';
        ass prop:'function' str:'egress';
        ass prop:'major connectivity' str:'exterior';
 else;
if: n = 102;
        ass prop:'name' str:'back wall';
        ass prop:'bearing' str:'vertical load-bearing';
        ass prop:'Major Connectivity' str:'exterior';
else;
if: n = 113;
        ass prop:'name' str:'roof';
        ass prop:'Major Connectivity' str:'exterior';
        ass prop:'bearing' str:'horizontal load-bearing';
else;
if: n = 115;
        ass prop:'name' str:'interior volume';
        ass prop:'Major Connectivity' str:'exterior';
```

```
        ass prop:'location' str:'side/front';
        ass prop:'bearing' str:'vertical load-bearing?';
 else;
fin; fin; fin; fin; fin;
$
define done: n=139 % n = 102 % n = 113 % n = 115;
if: other & #done;
 if: n = 117;
        ass prop:'name' str:'interior volume';
        ass prop:'location' str:'rear';
        ass prop:'Major Connectivity' str:'exterior';
 else;
 if: n = 120;
        ass prop:'name' str:'large step contours';
        ass prop:'orientation' str:'diagonal';
        ass prop:'Major Connectivity' str:'exterior';
 else;
 if: n = 114;
        ass prop:'name' str:'Floor/Front wall';
        ass prop:'bearing' str: 'vertical load-bearing';
        ass prop:'Major Connectivity' str:'exterior';
else;
 if: n = 116;
        ass prop:'name' str:'step wall';
        ass prop:'bearing' str:'vertical load-bearing';
        ass prop:'Major Connectivity' str:'exterior';
fin; fin; fin; fin; fin;
$
show fragment header, properties;
copy fragment;
end op;
```

```
do dr;
$ declare variables to give properties to piper

open in fi:'dn0:[studio.steve]piper3.bas';
define which: (1,170);
define fx:0,fy:0,dx:0, dy:0,span:0,len:0;
 create out fi:'piper.bap';
 per op:'giveprop' for fragment range: which;
 end file;
 quit;
 end op;
```

Appendix II

A tabular listing of the properties of all fragments in the data base--something like a "parts list," is easy to produce.

What follows is a selection from the property list of the entire data base.

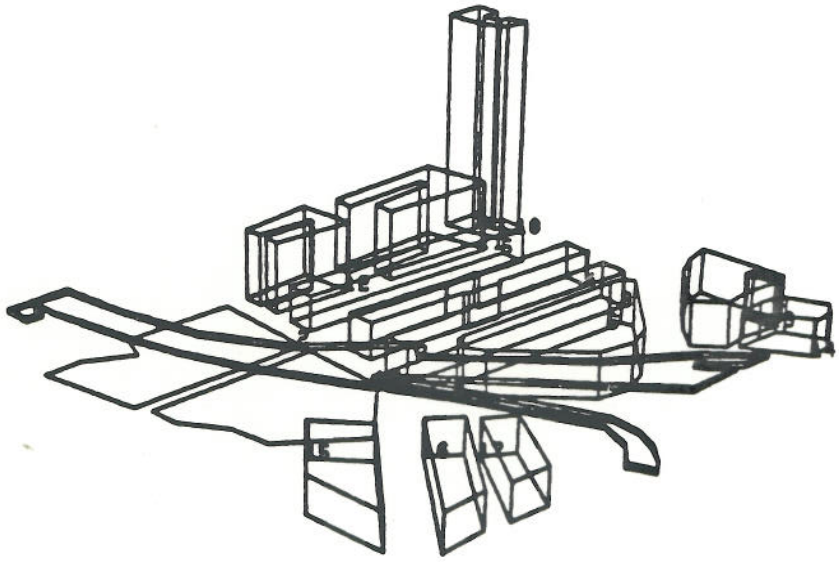
```
SPAN :        -200
NAME      :  COLUMN
BEARING   :  VERTICAL LOAD-BEARING
FUNCTION      :  VERTICAL LOAD-BEARING
MAJOR CONNECTIVITY   :  EXTERIOR
Fragment          109  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       110      X-Y               10           234          303
DIE ORIGIN :          (608,157)
SPAN :        -69
NAME      :  COLUMN
BEARING   :  VERTICAL LOAD-BEARING
FUNCTION      :  VERTICAL LOAD-BEARING
MAJOR CONNECTIVITY   :  EXTERIOR
Fragment          110  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       111      X-Y               10           103          303
DIE ORIGIN :          (610,400)
SPAN :        -200
NAME      :  COLUMN
BEARING   :  VERTICAL LOAD-BEARING
FUNCTION      :  VERTICAL LOAD-BEARING
MAJOR CONNECTIVITY   :  EXTERIOR
Fragment          111  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       112      X-Y               10           103          303
DIE ORIGIN :          (612,642)
SPAN :        -200
NAME      :  COLUMN
BEARING   :  VERTICAL LOAD-BEARING
FUNCTION      :  VERTICAL LOAD-BEARING
MAJOR CONNECTIVITY   :  EXTERIOR
Fragment          112  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       113      Y-Z               16           610           81
DIE ORIGIN :          (910.0001,304)
SPAN :        -529
NAME      :  ROOF
MAJOR CONNECTIVITY   :  EXTERIOR
BEARING   :  HORIZONTAL LOAD-BEARING
Fragment          113  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       114      Y-Z                3           610           81
DIE ORIGIN :          (344,101)
SPAN :        -529
NAME      :  FLOOR/FRONT WALL
BEARING   :  VERTICAL LOAD-BEARING
MAJOR CONNECTIVITY   :  EXTERIOR
Fragment          114  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       115      Y-Z                7           148           81
DIE ORIGIN :          (191,189)
SPAN :        -67
NAME      :  INTERIOR VOLUME
MAJOR CONNECTIVITY   :  EXTERIOR
LOCATION      :  SIDE/FRONT
BEARING   :  VERTICAL LOAD-BEARING?
Fragment          115  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       116      Y-Z                4           508          497
DIE ORIGIN :          (365,102)
SPAN :        -11
NAME      :  STEP WALL
BEARING   :  VERTICAL LOAD-BEARING
MAJOR CONNECTIVITY   :  EXTERIOR
Fragment          116  WRITTEN TO OUTPUT FILE.
```

```
Fragment    PLANE        # POINTS    BASE ELEV    TOP ELEV
    117       Y-Z            3           610          508
DIE ORIGIN :        (352,101)
SPAN :        -102
NAME       : INTERIOR VOLUME
LOCATION   : REAR
MAJOR CONNECTIVITY  : EXTERIOR
Fragment        117  WRITTEN TO OUTPUT FILE.
Fragment    PLANE        # POINTS    BASE ELEV    TOP ELEV
    118       Y-Z            2           610          508
DIE ORIGIN :        (349,264)
SPAN :        -102
NAME       : PIPE RAILING
LENGTH :        1
Fragment        118  WRITTEN TO OUTPUT FILE.
Fragment    PLANE        # POINTS    BASE ELEV    TOP ELEV
    119       Y-Z            2           610          508
DIE ORIGIN :        (349,248)
SPAN :        -102
NAME       : PIPE RAILING
LENGTH :        0
Fragment        119  WRITTEN TO OUTPUT FILE.
Fragment    PLANE        # POINTS    BASE ELEV    TOP ELEV
    120       Y-Z           18           497          148
DIE ORIGIN :        (101,233)
SPAN :        -349
NAME       : LARGE STEP CONTOURS
ORIENTATION : DIAGONAL
MAJOR CONNECTIVITY  : EXTERIOR
Fragment        120  WRITTEN TO OUTPUT FILE.
Fragment    PLANE        # POINTS    BASE ELEV    TOP ELEV
    121       Y-Z            5           497          462
DIE ORIGIN :        (101,228)
SPAN :        -35
NAME       : STEP (SMALL)
MAJOR CONNECTIVITY  : LARGE STEP CONTOURS
Fragment        121  WRITTEN TO OUTPUT FILE.
Fragment    PLANE        # POINTS    BASE ELEV    TOP ELEV
    122       Y-Z            5           183          148
DIE ORIGIN :        (101,228)
SPAN :        -35
NAME       : STEP (SMALL)
MAJOR CONNECTIVITY  : LARGE STEP CONTOURS
Fragment        122  WRITTEN TO OUTPUT FILE.
Fragment    PLANE        # POINTS    BASE ELEV    TOP ELEV
    123       Y-Z            5           497          462
DIE ORIGIN :        (130,214)
SPAN :        -35
NAME       : STEP (SMALL)
MAJOR CONNECTIVITY  : LARGE STEP CONTOURS
Fragment        123  WRITTEN TO OUTPUT FILE.
Fragment    PLANE        # POINTS    BASE ELEV    TOP ELEV
    124       Y-Z            5           183          148
DIE ORIGIN :        (130,214)
SPAN :        -35
NAME       : STEP (SMALL)
MAJOR CONNECTIVITY  : LARGE STEP CONTOURS
Fragment        124  WRITTEN TO OUTPUT FILE.
Fragment    PLANE        # POINTS    BASE ELEV    TOP ELEV
    125       Y-Z            5           497          462
DIE ORIGIN :        (161,200)
SPAN :        -35
NAME       : STEP (SMALL)
MAJOR CONNECTIVITY  : LARGE STEP CONTOURS
Fragment        125  WRITTEN TO OUTPUT FILE.
Fragment    PLANE        # POINTS    BASE ELEV    TOP ELEV
```

```
NAME      :  STEP (SMALL)
MAJOR CONNECTIVITY    :  LARGE STEP CONTOURS
Fragment          135  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       136      Y-Z                5           183           148
DIE ORIGIN :          (315,126)
SPAN :         -35
NAME      :  STEP (SMALL)
MAJOR CONNECTIVITY    :  LARGE STEP CONTOURS
Fragment          136  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       137      Y-Z                5           497           462
DIE ORIGIN :          (344,111)
SPAN :         -35
NAME      :  STEP (SMALL)
MAJOR CONNECTIVITY    :  LARGE STEP CONTOURS
Fragment          137  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       138      Y-Z                5           183           148
DIE ORIGIN :          (344,111)
SPAN :         -35
Fragment          138  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       139      Y-Z                5            81             6
DIE ORIGIN :          (322,102)
SPAN :         -75
NAME      :  ENTRANCE
FUNCTION      :  EGRESS
MAJOR CONNECTIVITY    :  EXTERIOR
Fragment          139  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       140      Y-Z                4             6             0
DIE ORIGIN :          (194,101)
SPAN :         -6
NAME      :  DOOR
FUNCTION      :  EGRESS
Fragment          140  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       141      Y-Z                4             6             0
DIE ORIGIN :          (259,102)
SPAN :         -6
NAME      :  DOOR
FUNCTION      :  EGRESS
Fragment          141  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       142      Y-Z                2           494           493
DIE ORIGIN :          (102,263)
SPAN :         -1
NAME      :  PIPE RAILING
LENGTH :          259.8558
Fragment          142  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       143      Y-Z                2           151           152
DIE ORIGIN :          (102,263)
SPAN :         -1
NAME      :  PIPE RAILING
LENGTH :          259.8558
Fragment          143  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
       144      Y-Z                2           151           152
DIE ORIGIN :          (103,247)
SPAN :         -1
NAME      :  PIPE RAILING
LENGTH :          258.5209
Fragment          144  WRITTEN TO OUTPUT FILE.
 Fragment     PLANE          # POINTS     BASE ELEV     TOP ELEV
```

B U I L D E R
A Graphics System for Spatial Analysis

Problem Set 4



Heuristics in Urban Design
Edward Popko Lecturer
Bruce Donald Research Associate
Urban Design Program
Graduate School of Design
Harvard University
Cambridge, Ma. 02139

April 16, 1982

Acknowledgements:

This problem set could not have been compiled without the support of the staff of the Laboratory for Computer Graphics and Spatial Analysis. I would like to thank Bruce Donald and Hugh Keegan in particular for their suggestions and technical assistance in preparing data and computer accounts for everyone.

Background

Boston is undergoing unprecedented center-city redevelopment. In the last 5 years alone, more than 4 square miles of land have been redeveloped. The city's Public Facilities Group scehdules all long term capital improvement projects and often uses architectural competitions as a way of obtaining quality designs for schools, parks, community centers, and parking structures.

Problem Statement

You have been commissioned by the city to develop a building envelope for a new parking structure to be located east of Quincy Market. Your envelope defines the outer-most limits of any design built on the site and it will be be used as the basis for evaluating submissions for city-wide urban design competition; submissions must fit within your envelope and respect the circulation points that you indicate.

Figure 1 shows the general site of the proposed structure. The site is approximately 2 acres with potential access from Commercial Street, State Street, and the South-East express service drive. Your envelope must meet the following specifications:

1) four (4) feet setback from the lot boundaries for sidewalks.

2) maximum height of 50 feet. Note that the portion of the site under the expressway can not exceed 38 feet.

3) two points of ingress/egress (four total) for automobile circulation.

4) four points for pedestrian access to building.

Use the BUILDER program to create one or more buildings (or fragments) that meet the above requirements. You may need to stack several fragements to achieve the desired envelope. Figure 2 shows an aerial perspective of the entire Quincy Market redevelopment project. The proposed parking structure is site 2. Figure 2 also lists the base and height of each of the buildings in the area. [1]

Document your building envelopes with one plan view perspective of the entire site and five time-series perspectives using the view-points and station-point

---

[1] The basic site plan is contained in file "quincy.bld".

indicated in Figure 3. [2] For extra credit, create animations of the site using BUILDER options for color, hidden-line, or focal ratio options.

---

[2] Five perspectives are required, one from station points 27, 28, 29, 30, and 31 looking at center of vision point 26.

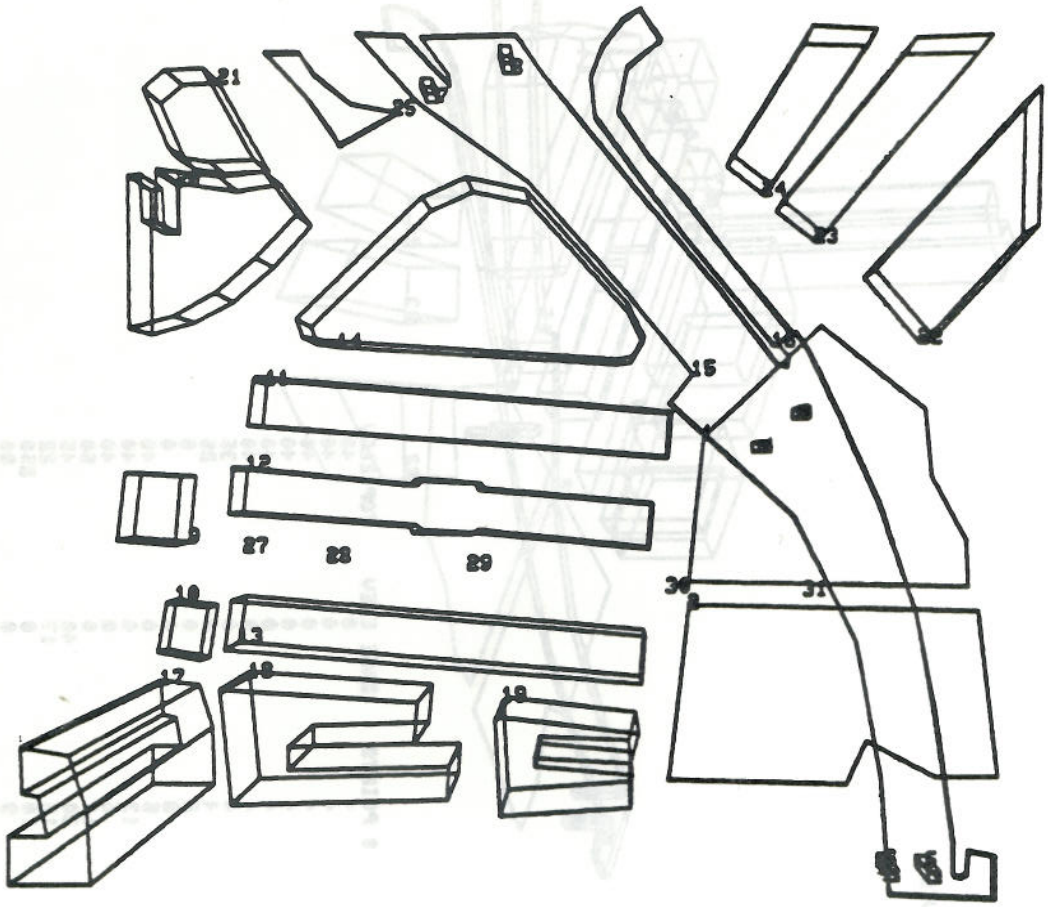


Figure 1 - Quincy Market Redevelopment Area

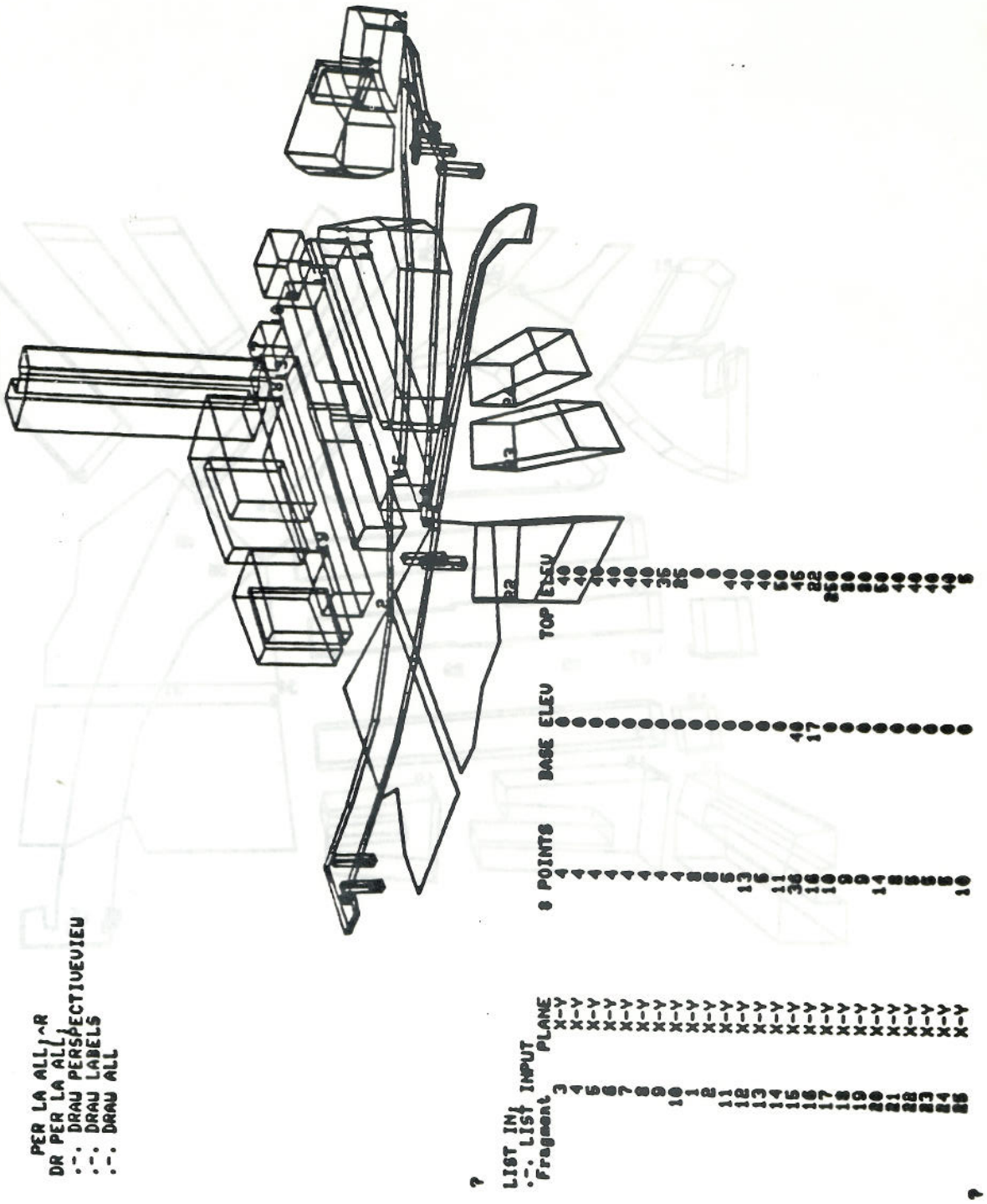


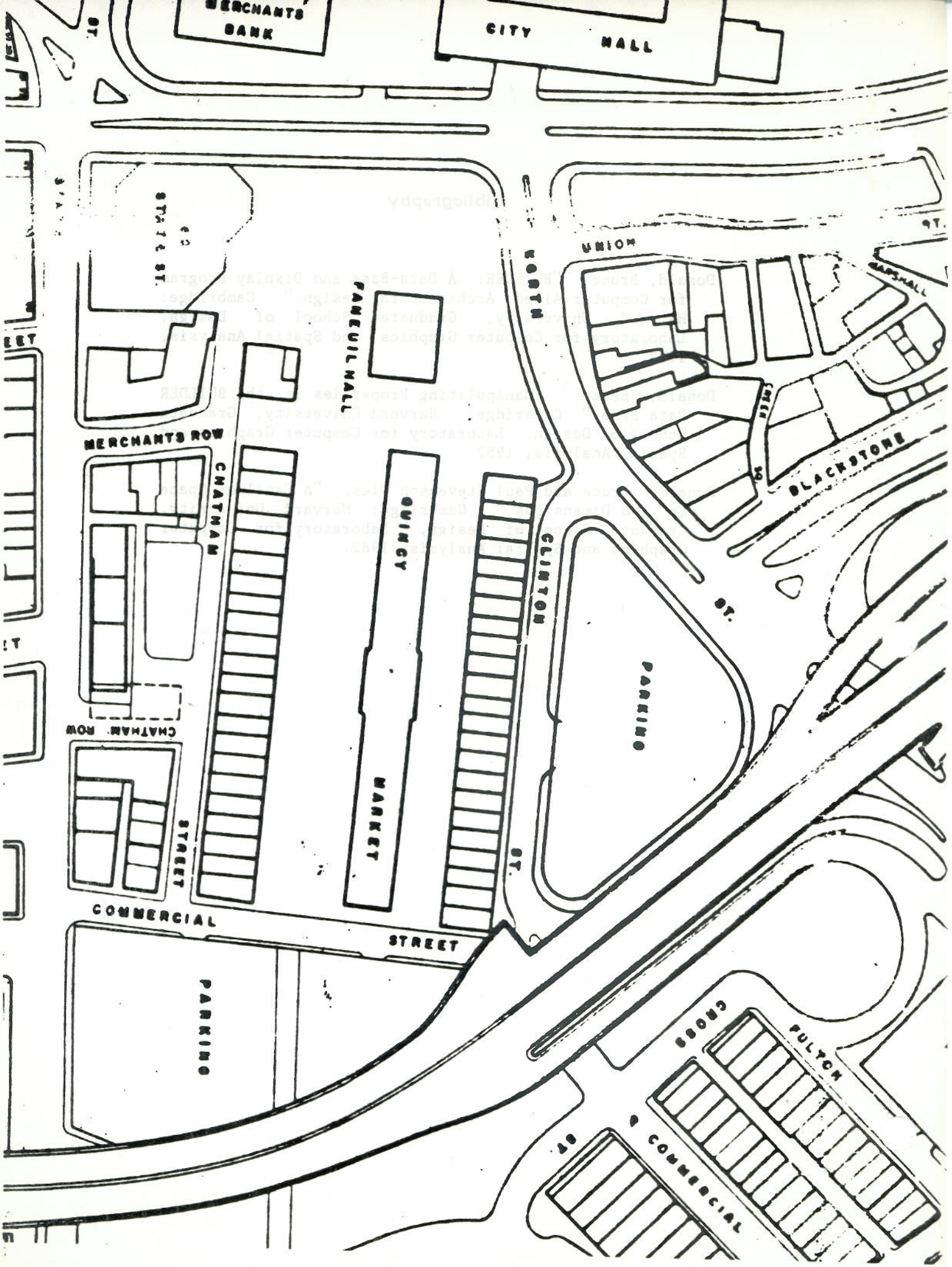
Figure 2 - Perspective View of Redevelopment Site

Bibliography

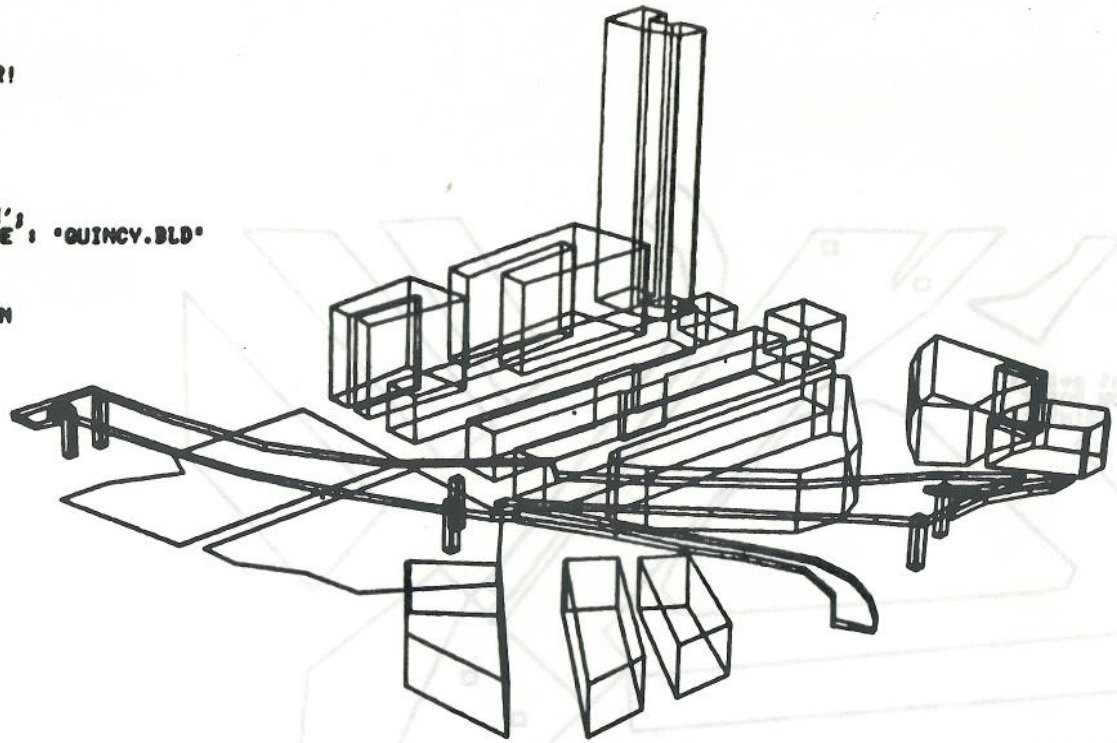
Donald, Bruce. "BUILDER: A Data-Base and Display Program for Computer-Aided Architectural Design." Cambridge: Harvard University, Graduate School of Design, Laboratory for Computer Graphics and Spatial Analysis, 1982.

Donald, Bruce. "Manipulating Properties in the BUILDER Data Base." Cambridge: Harvard University, Graduate School of Design, Laboratory for Computer Graphics and Spatial Analysis, 1982.

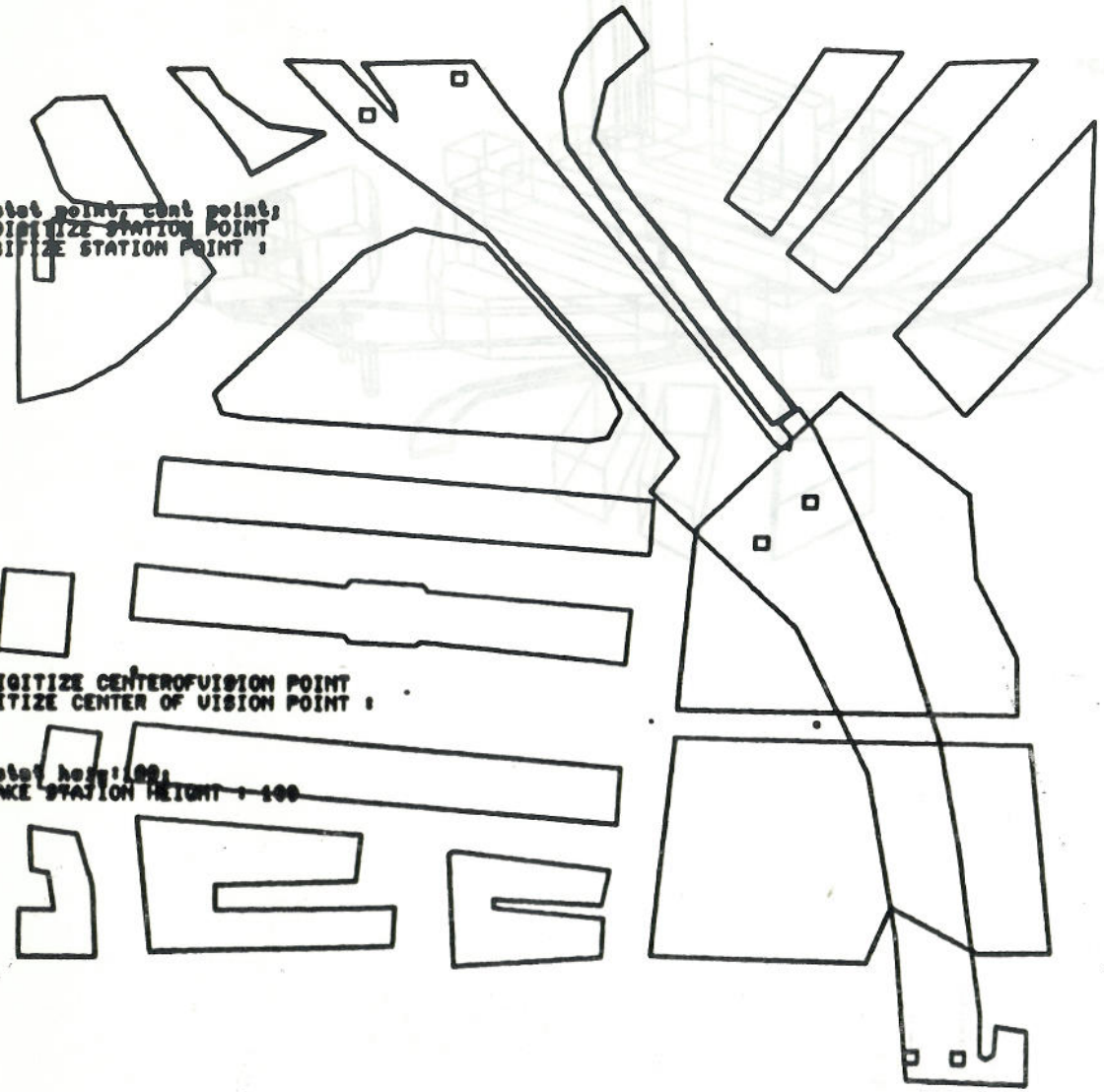
Donald, Bruce and Paul Stevenson Oles. "A Familiar Space in Two Dimensions." Cambridge: Harvard University, Graduate School of Design, Laboratory for Computer Graphics and Spatial Analysis, 1982.


MERCHANTS BANK
CITY HALL
STATE ST.
FANEUIL HALL
MERCHANTS ROW
CHATHAM
QUINCY
MARKET
CLINTON
ST.
UNION
MARSHALL
BLACKSTONE
PARKING
COMMERCIAL
STREET
PARKING
COMMERCIAL
FULTON

```
builder
--- WELCOME TO BUILDER!
?

do draw;
.-. DO DRAWING
?

op in file='quincy.bld';
.-. OPEN INPUT FILENAME : 'QUINCY.BLD'
?

draw no clear;
.-. DRAW NO CLEARSCREEN
?

draw all;
.-. DRAW ALL
```

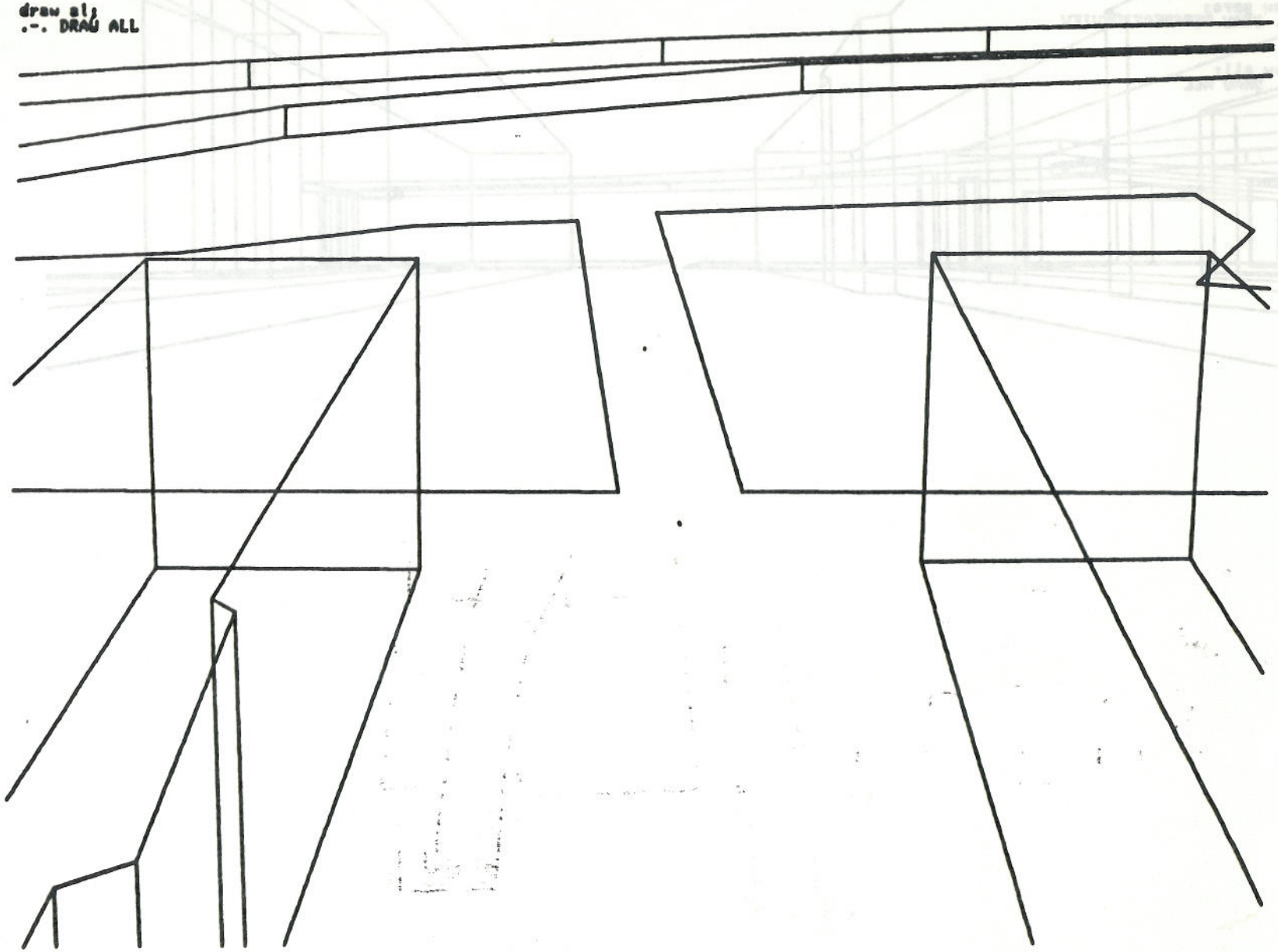


```
?

draw plan;
:-. DRAW PLAN
?

draw all;
:-. DRAW ALL
```

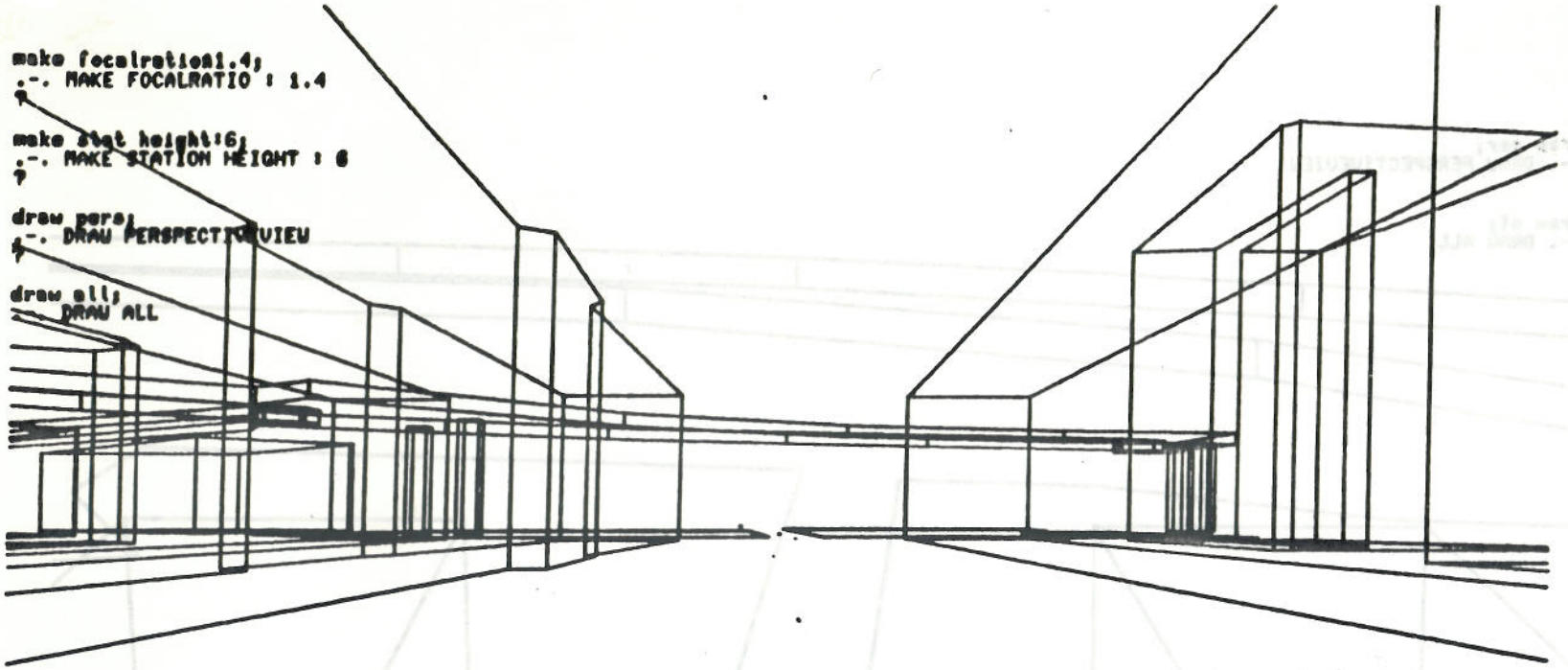


```
?

dig stat point, cent point;
:-. DIGITIZE STATION POINT
+ DIGITIZE STATION POINT :

:-. DIGITIZE CENTEROFVISION POINT
+ DIGITIZE CENTER OF VISION POINT :
?

make stat height;
:-. MAKE STATION HEIGHT + 400
?
```

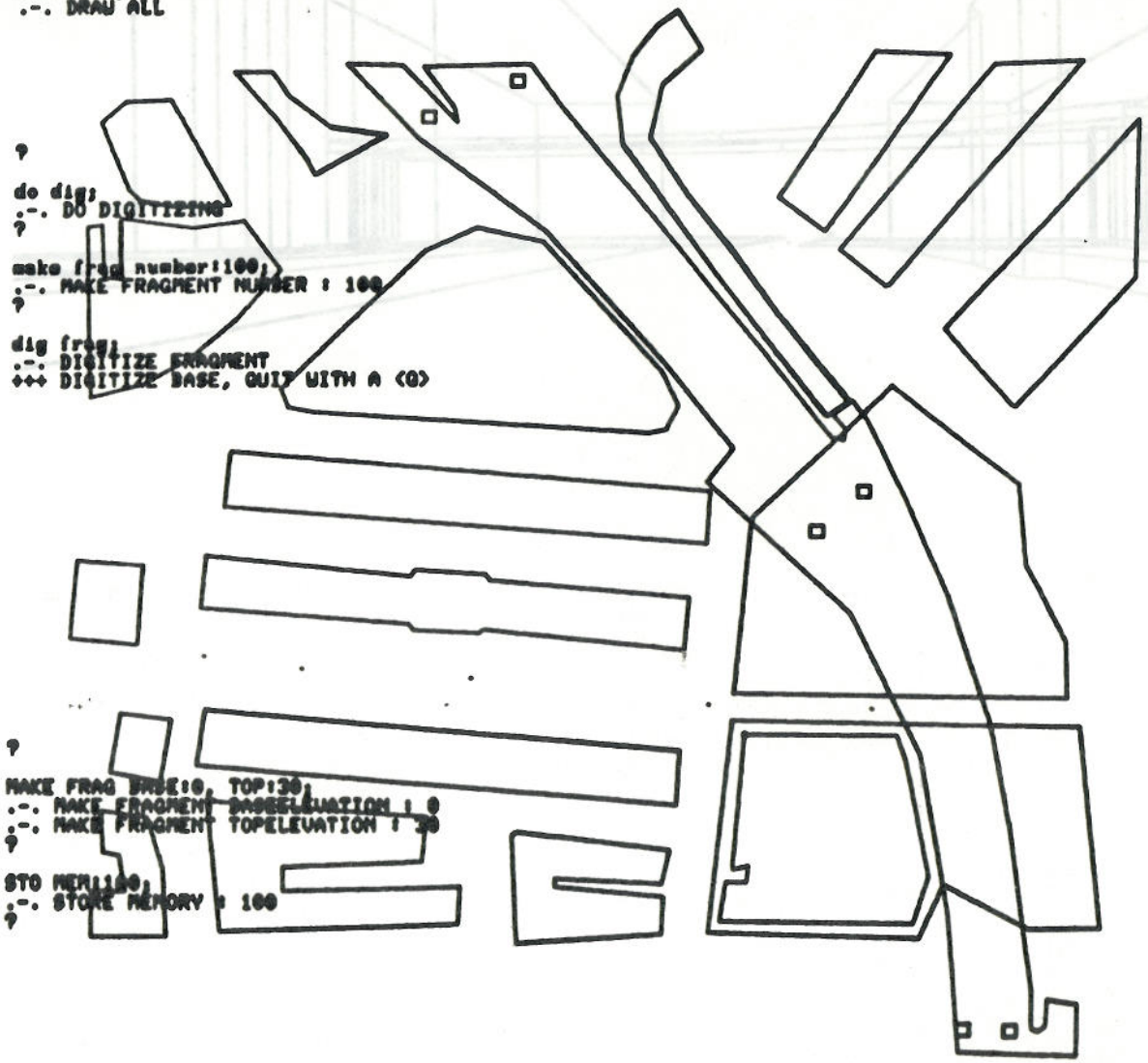
```
draw per;
:-. DRAW PERSPECTIVEVIEW
?

draw al;
.-. DRAW ALL
```



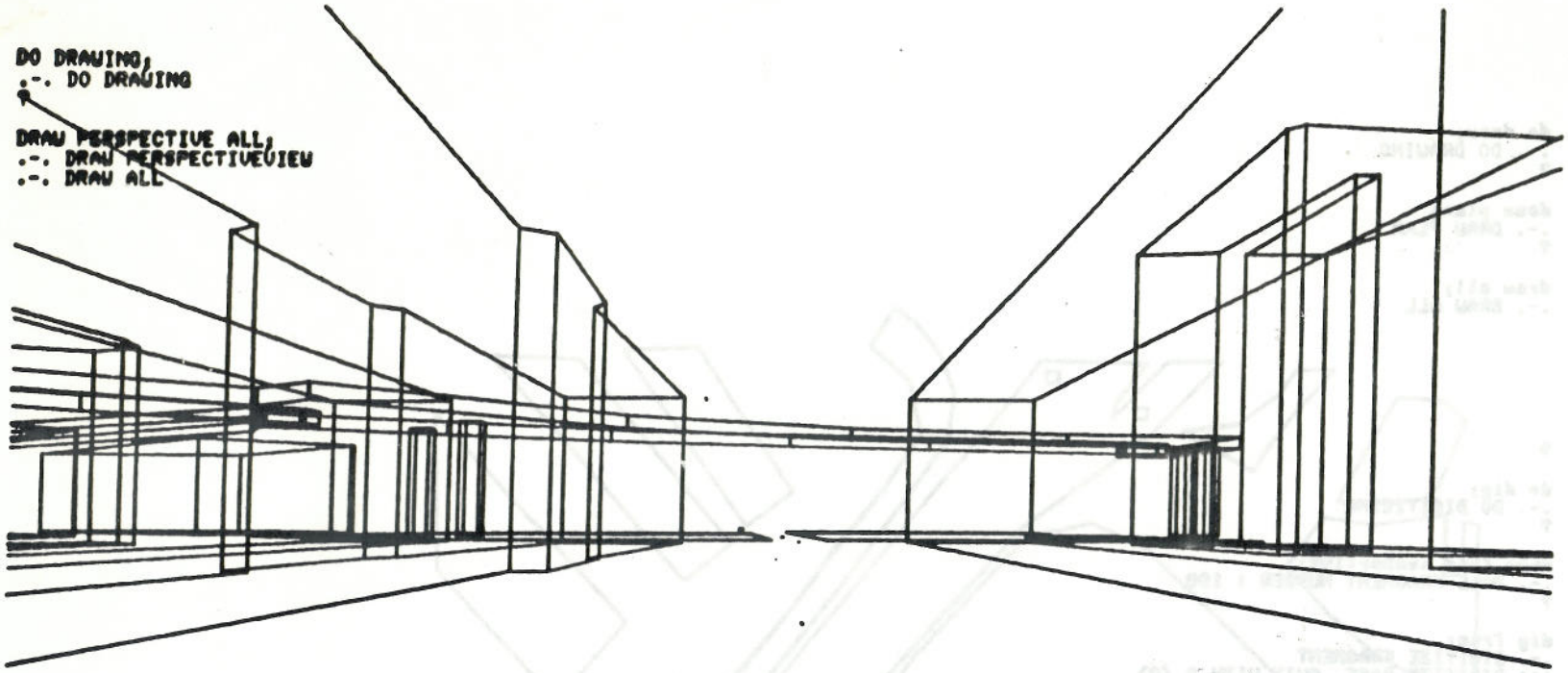
```
make focalration1.4;
.-. MAKE FOCALRATIO : 1.4
?

make stat height:6;
.-. MAKE STATION HEIGHT : 6
?

draw para;
.-. DRAW PERSPECTIVE VIEW
?

draw all;
.-. DRAW ALL
```



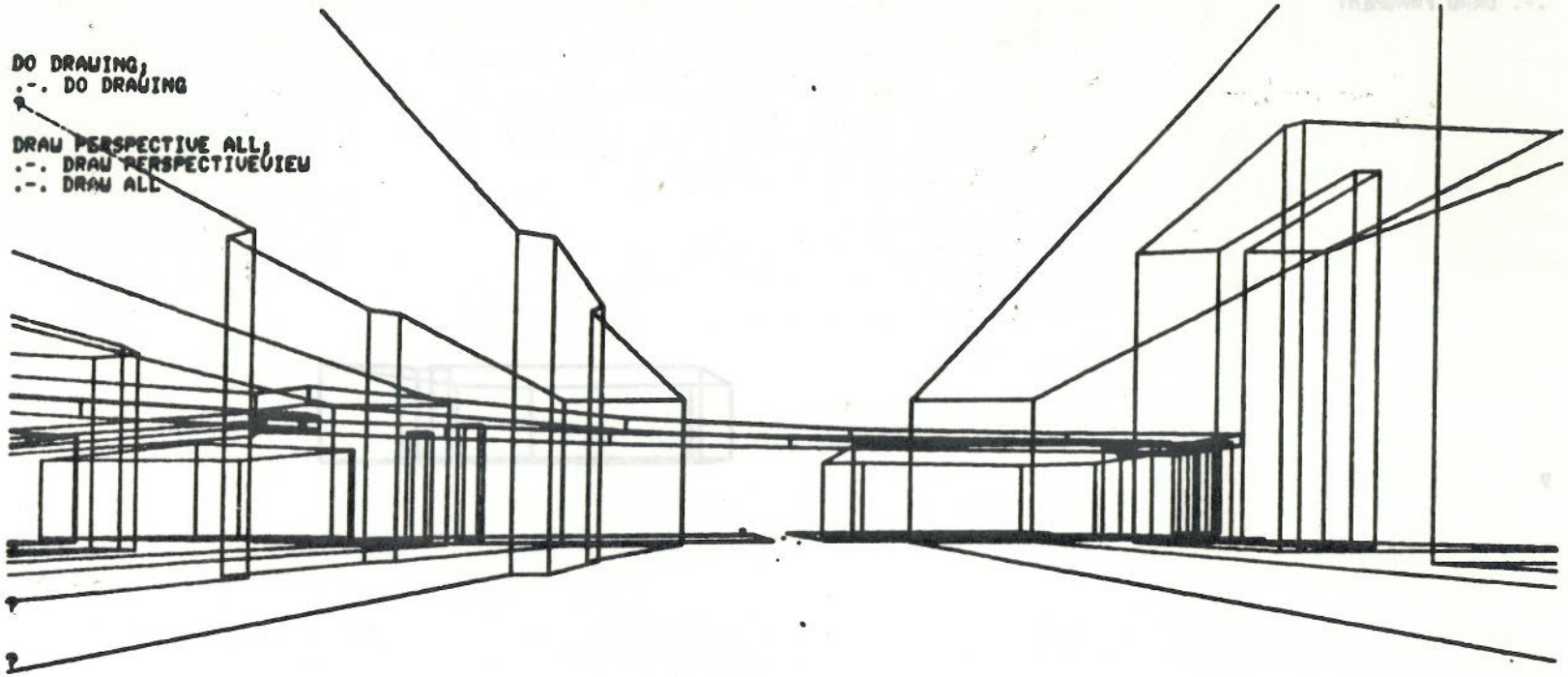
```
do draw;
.-. DO DRAWING
?

draw plan;
.-. DRAW PLAN
?

draw all;
.-. DRAW ALL
?

do dig;
.-. DO DIGITIZING
?

make frag number:100;
.-. MAKE FRAGMENT NUMBER : 100
?

dig frag;
.-. DIGITIZE FRAGMENT
+++ DIGITIZE BASE, QUIT WITH A <0>
?

MAKE FRAG BASE:0, TOP:30;
.-. MAKE FRAGMENT BASEELEVATION : 0
.-. MAKE FRAGMENT TOPELEVATION : 30
?

STO MEM:100;
.-. STORE MEMORY : 100
?
```



```
DO DRAWING;
.-. DO DRAWING
;

DRAW PERSPECTIVE ALL;
.-. DRAW PERSPECTIVEVIEW
.-. DRAW ALL
```

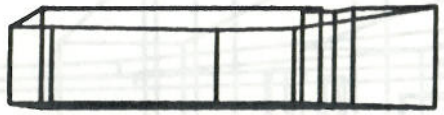


```
DO DRAWING;
.-. DO DRAWING
?

DRAW PERSPECTIVE ALL;
.-. DRAW PERSPECTIVEVIEW
.-. DRAW ALL
```



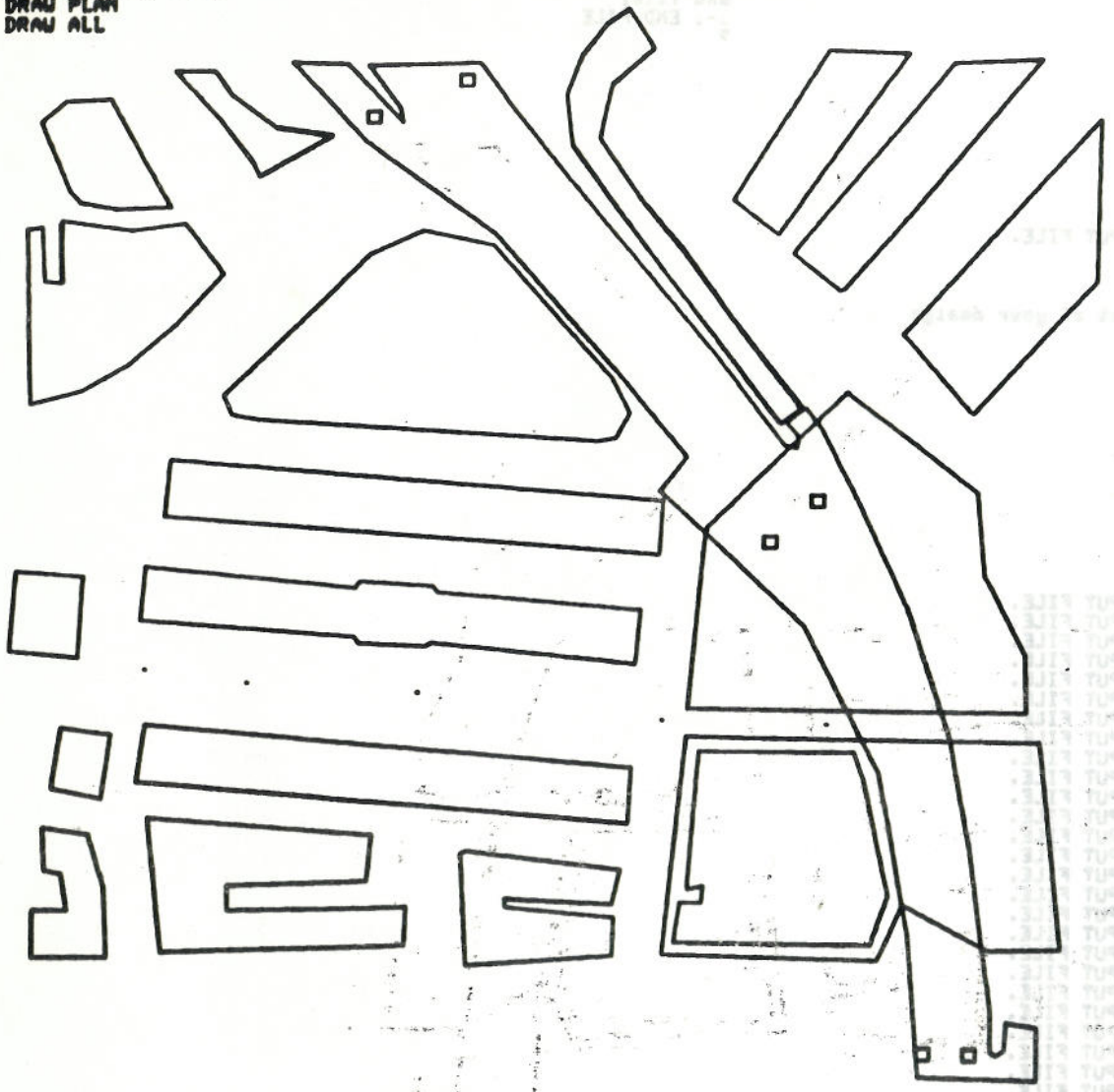
```
?

?

REC MEM:100;
.-. RECALL MEMORY : 100
?

DRAW FRAGMENT;
.-. DRAW FRAGMENT
```

```
DRAW FRAG,
.-. DRAW FRAGMENT
```

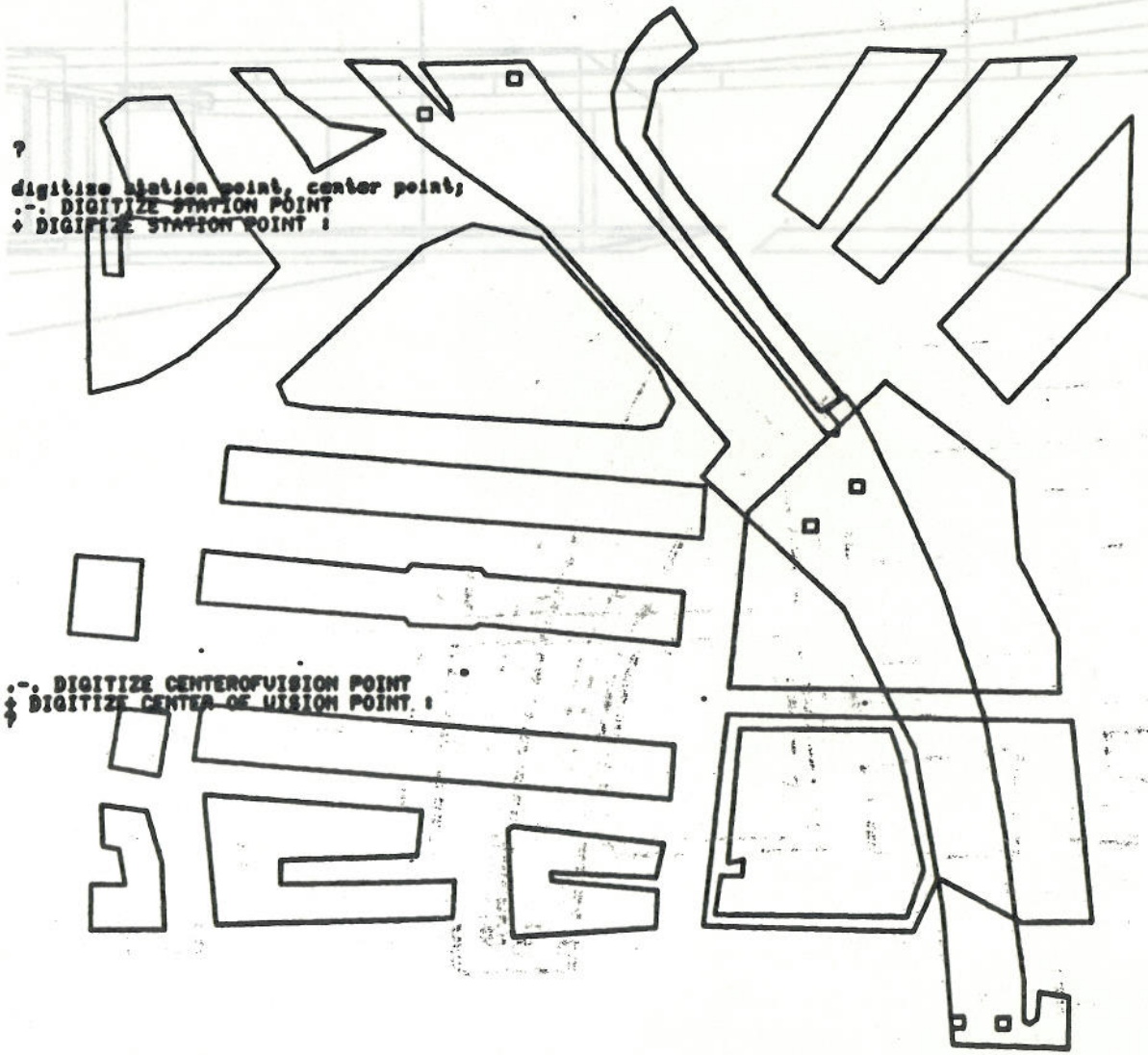


```
?

create output file:'XXX.bld';
.-. CREATE OUTPUT FILENAME : 'XXX.BLD'
?

do drawing;
.-. DO DRAWING
?

recall mem:100;
.-. RECALL MEMORY : 100
?

copy fragment;
.-. COPY FRAGMENT
Fragment        100  WRITTEN TO OUTPUT FILE.
?

$
$ Note: Recall and copy each fragment of your design
$       as above.
$
?

rewind;
.-. REWIND
?

draw next;
.-. DRAW NEXT
?

copy rest;
.-. COPY REST
Fragment         26  WRITTEN TO OUTPUT FILE.
Fragment         27  WRITTEN TO OUTPUT FILE.
Fragment         28  WRITTEN TO OUTPUT FILE.
Fragment         29  WRITTEN TO OUTPUT FILE.
Fragment         30  WRITTEN TO OUTPUT FILE.
Fragment         31  WRITTEN TO OUTPUT FILE.
Fragment          3  WRITTEN TO OUTPUT FILE.
Fragment          4  WRITTEN TO OUTPUT FILE.
Fragment          5  WRITTEN TO OUTPUT FILE.
Fragment          6  WRITTEN TO OUTPUT FILE.
Fragment          7  WRITTEN TO OUTPUT FILE.
Fragment          8  WRITTEN TO OUTPUT FILE.
Fragment          9  WRITTEN TO OUTPUT FILE.
Fragment         10  WRITTEN TO OUTPUT FILE.
Fragment          1  WRITTEN TO OUTPUT FILE.
Fragment          2  WRITTEN TO OUTPUT FILE.
Fragment         11  WRITTEN TO OUTPUT FILE.
Fragment         12  WRITTEN TO OUTPUT FILE.
Fragment         13  WRITTEN TO OUTPUT FILE.
Fragment         14  WRITTEN TO OUTPUT FILE.
Fragment         15  WRITTEN TO OUTPUT FILE.
Fragment         16  WRITTEN TO OUTPUT FILE.
Fragment         17  WRITTEN TO OUTPUT FILE.
Fragment         18  WRITTEN TO OUTPUT FILE.
Fragment         19  WRITTEN TO OUTPUT FILE.
Fragment         20  WRITTEN TO OUTPUT FILE.
Fragment         21  WRITTEN TO OUTPUT FILE.
Fragment         22  WRITTEN TO OUTPUT FILE.
Fragment         23  WRITTEN TO OUTPUT FILE.
Fragment         24  WRITTEN TO OUTPUT FILE.
Fragment         25  WRITTEN TO OUTPUT FILE.
Fragment         32  WRITTEN TO OUTPUT FILE.
?

end file;
.-. END FILE
?
```

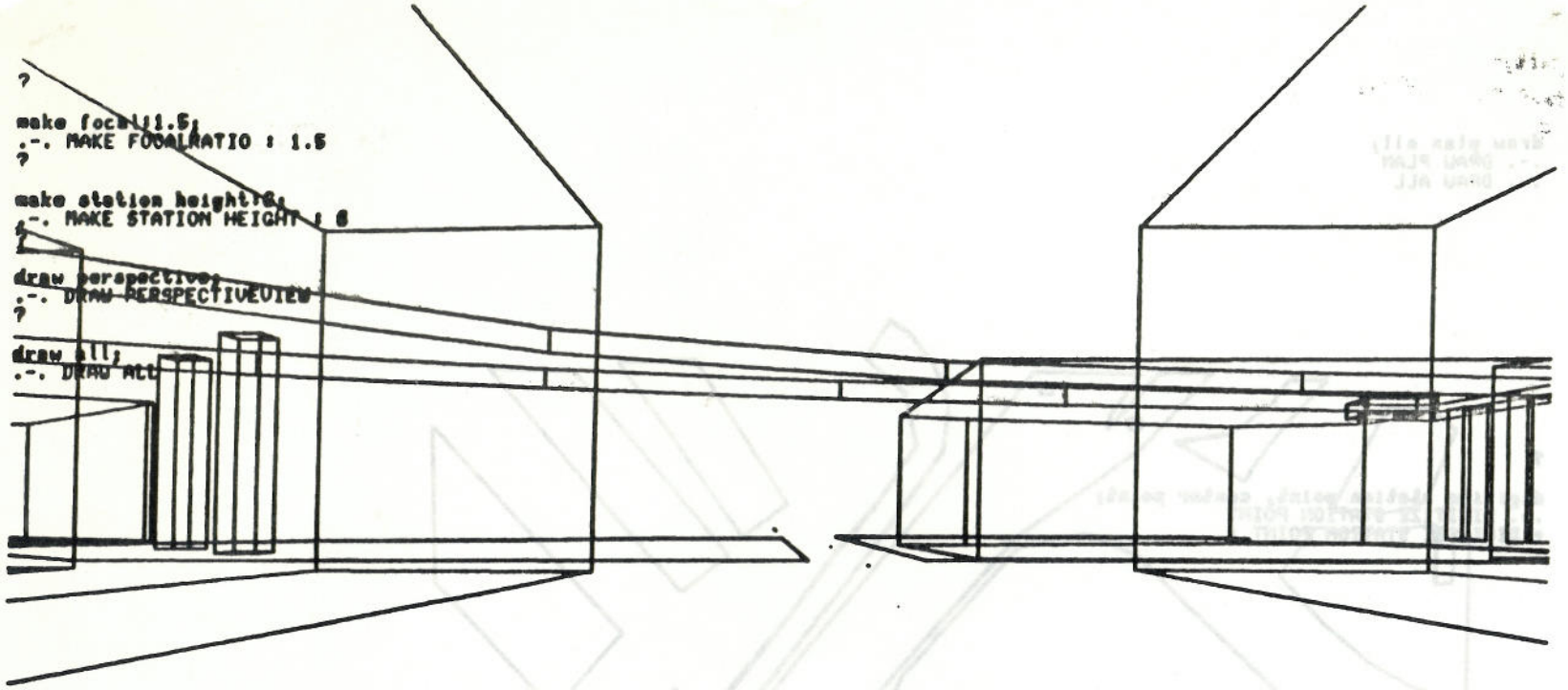
```
do drawing;
.-. DO DRAWING
?

open input file:'XXX.bld';
.-. OPEN INPUT FILENAME : 'XXX.BLD'
?

draw plan ; draw all;
.-. DRAW PLAN
.-. DRAW ALL
```



```
draw plan all;
.-. DRAW PLAN
.-. DRAW ALL
```



```
?
digitize station point, center point;
.-. DIGITIZE STATION POINT
: DIGITIZE STATION POINT :
.-. DIGITIZE CENTEROFVISION POINT
: DIGITIZE CENTER OF VISION POINT :
```


?
make focal1.5;
.-. MAKE FOCALRATIO : 1.5
?
make station height6;
.-. MAKE STATION HEIGHT : 6
?
draw perspective;
.-. DRAW PERSPECTIVEVIEW
?
draw all;
.-. DRAW ALL

```
?
quit;
SEE YOU LATER!!.
$
```