Contributed Articles

The Interart Graphics System

Sanford Ressler

The Interart graphics package was designed for use by artists and other non-technical people. Ease of use was the major design consideration. The package is implemented in APL and is therefore inherently interactive. Full three-dimensional representations are supported, and objects are defined using Cartesian xyz-coordinates. There is a controllable "eye" which can be moved around to view the object from anywhere in space.

From the outset, Interart was designed to be used by artists. The system was written by an artist and a computer scientist working together. Their goal was to provide naive users with complex graphical capabilities without sacrificing usability. A great deal of the usability of Interart is due to the environment that is provided by APL; it is highly interactive and well suited for easy graphic manipulations which the user can immediately test and play with. The ability to see some graphical ideas realized instantly is vital in getting non-computer people involved with a computer system. Given relatively few basic graphical functions, one can combine the commands in a fairly intuitive manner to achieve a complicated graphical operation.

In general use, an object is defined using a cross-hair cursor, or by combining previously existing objects which are then modified via repositioning, scaling, rotation, or any number of functions. The data are represented as an (N,4)-matrix, the first column of which is an operation code (0 for repositioning without drawing; 1 for drawing a line) and the remaining three columns are coordinates. An object can always be defined explicitly by entering the coordinates of each of its points.

The space is defined as a Cartesian coordinate system. In the default eye position, x increases to the right, y increases up, and z increases away from the user. The present graphics screen (a Tektronix 4013) measures 1024 units in the x-direction and 780 in the y-direction.

Most Interart commands are of the form:

```
NOBJ ← ARGS CMD OBJ
```

where:

OBJ is a graphic object to be operated on

CMD is an Interart command

ARGS contains the arguments to the command

NOBJ is the graphic object resulting from the command.

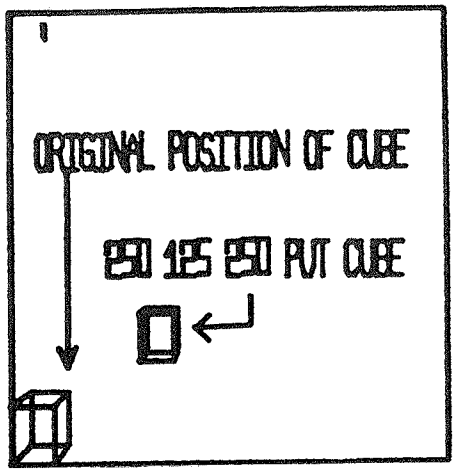
Constructing and Positioning

Because the results of an action may in turn be acted upon, commands can be composed to form powerful command lines. Following is a description of the major functions in Interart with examples of their uses:

PUT creates an object like its right argument with its lower left forward point positioned at the point specified absolutely (in screen coordinates) by its left argument:

```
R ← XYZ PUT OBJ
```

The result may be saved in a variable (R) and/or drawn right away, e.g. DRAW 100 100 200 PUT CUBE (see Illustration 1).



MOVE is a relative positioning function:

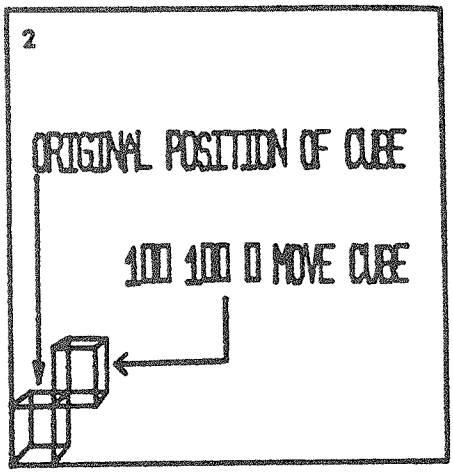
```
R ← XYZ MOVE OBJ
```

duplicates an object at specified distance XYZ from the original (see Illustration 2).

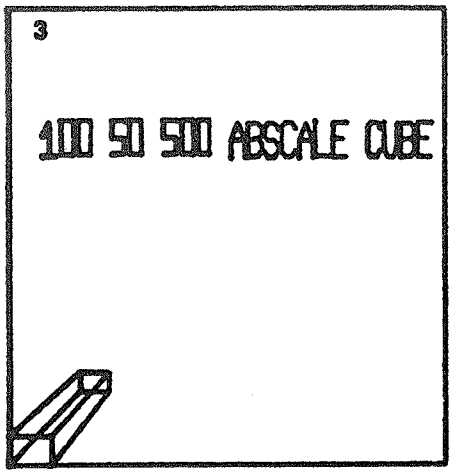
ABSCALE duplicates an object resized to the exact widths specified in XYZ:

```
R ← XYZ ABSCALE OBJ
```

100 50 500 ABSCALE CUBE will create a cube



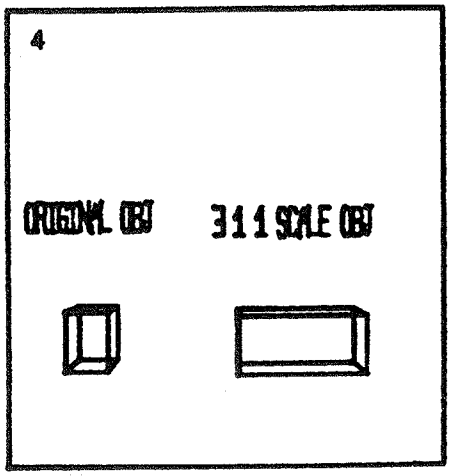
100 units wide, 50 tall, and 500 deep (see Illustration 3).



SCALE will duplicate an object changed in size by the relative factors specified in XYZ:

```
R ← XYZ SCALE OBJ
```

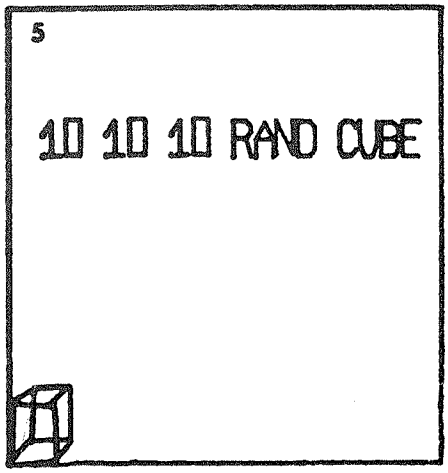
Thus 2 1 4 SCALE CUBE will double the width, leave the height the same, and increase the depth four times (see Illustration 4).



RAND duplicates an object with random perturbations of its coordinates, within the range specified by XYZ:

```
R ← XYZ RAND OBJ
```

(See Illustration 5).



CAT duplicates two objects, concatenating them into a single object:

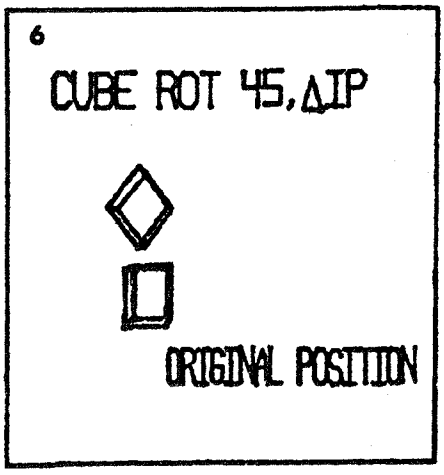
```
R ← OBJ1 CAT OBJ2
```

The two data matrices are joined into one with no visual change to the position of the objects. There will also be a joining line when the second object begins with operation code set to "draw".

ROT duplicates an object rotated a specified number of degrees (D) around an arbitrary line:

```
R ← OBJ ROT D, XYZXYZ
```

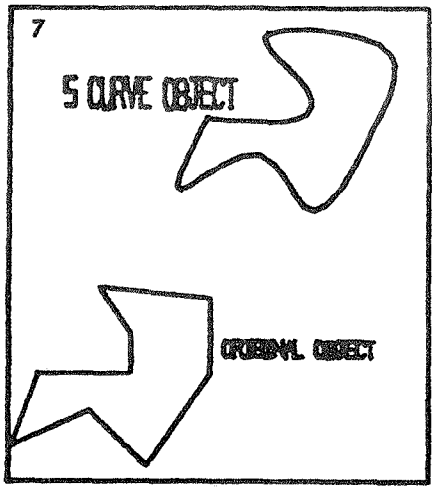
Since a line is defined by two sets of coordinates, six numbers are required. There are several predefined lines: LS, LD, RS, RD, HS, HD, VS, VD. In these names, L (or R) means that the top point of a diagonal line is on the left (or right) side of the screen. H (or V) means horizontal (or vertical). A line whose name ends in S is on the surface, a D in the name means the line is pushed back in space, while IP is in the plane. All lines are centered. (See Illustration 6).



CURVE does curve fitting for objects:

R ← N CURVE OBJ

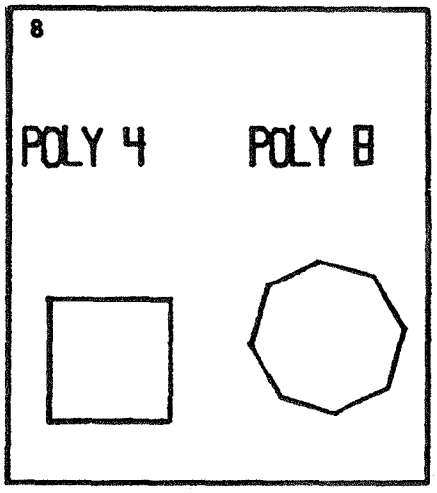
5 CURVE OBJ will place five points between successive pairs of points of OBJ, and the curve generated passes through successive pairs of points of OBJ (see Illustration 7).



POLY generates regular polygons of N sides:

R ← POLY N

POLY 4 generates a square (see Illustration 8).

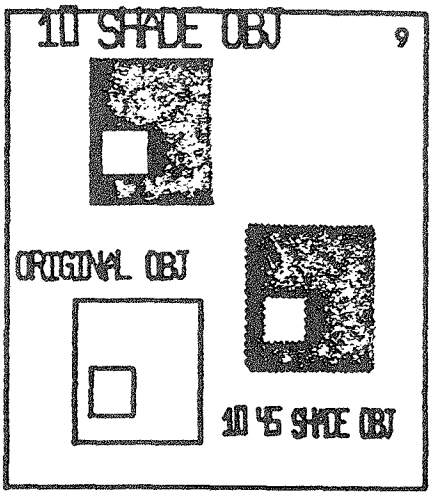


SHADE will generate a hatching and/or cross-hatching effect within an area on the surface:

R ← ID SHADE OBJ

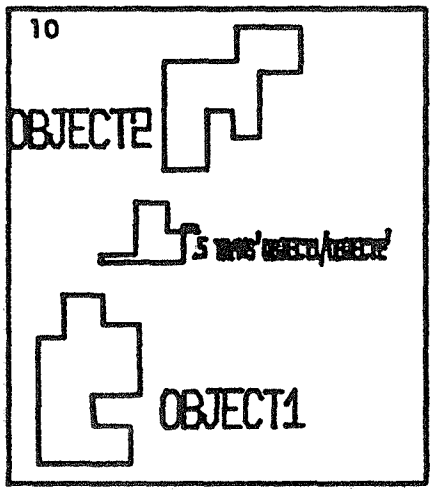
The object must be two-dimensional (i.e. all values z are zero). ID is the number of units between pairs of parallel lines (spacing) and the angle of the lines, in degrees. If the angle is omitted, the default, vertical lines, is used (see Illustration 9).

TRANS produces transformations from one object to another:



R ← N TRANS 'OBJ1/OBJ2'

N, a number between 0 and 1, is the degree to which a linear interpolation from OBJ1 into OBJ2 should be carried. Thus, 0.5 TRANS 'OBJ1/OBJ2' would produce an object halfway between the two. (See Illustration 10).



ANIMA is a repetitive version of the TRANS function:

R ← I ANIMA 'OBJ1/OBJ2'

It produces all the objects between OBJ1 and OBJ2 in increments of I; e.g. 0.1 ANIMA 'OBJ1/OBJ2' will draw both objects and nine intermediate transformations (see Illustration 11).

CONNECT generates connecting lines between objects of the same size (same number of points):

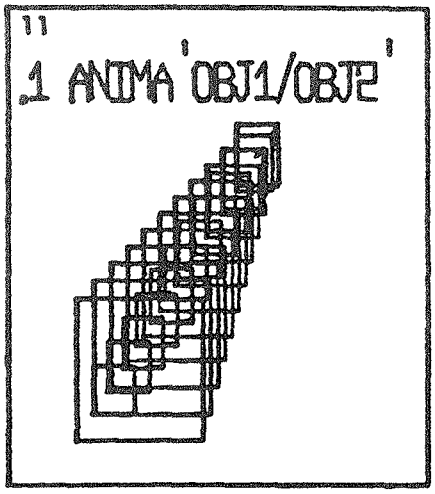
R ← OBJ1 CONNECT OBJ2

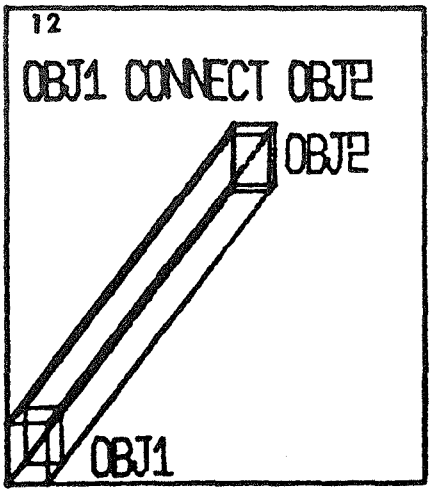
(See Illustration 12).

CONROT produces continuous rotations of an object:

CONROT

prompts for all input.





CONMOV causes an object to move continuously in specified directions:

```
CONMOV
```

prompts for all input.

CONPUT does a continuous (multiple) PUT of the left argument onto each point of the right argument:

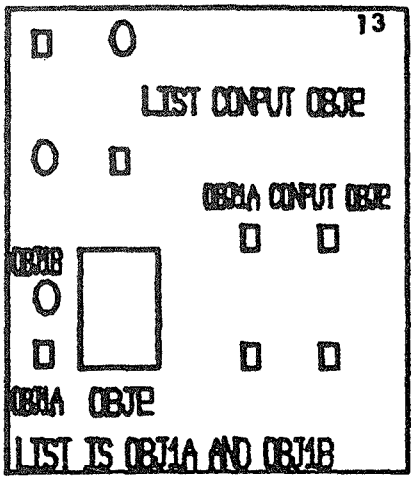
```
R ← OBJ1 CONPUT OBJ2
```

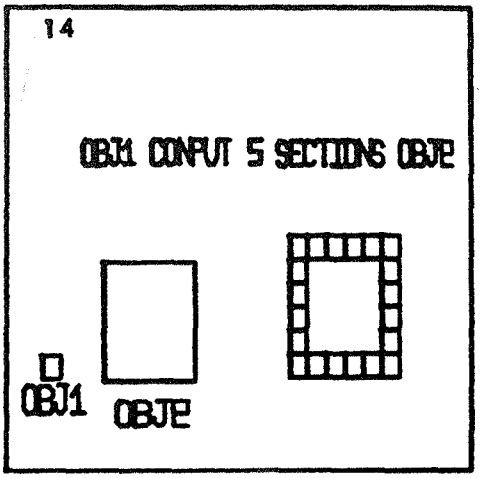
If the left argument is a square and the right argument is a circle, then SQUARE CONPUT CIRCLE would result in a circle, drawn with squares. Also, the left argument may be a character matrix containing the names of several pictures. These pictures will then be drawn sequentially onto the points of OBJ2. (See Illustration 13).

SECTIONS duplicates an object with each line broken into N segments:

```
R ← N SECTIONS OBJ
```

A useful sequence is to combine SECTIONS with CONPUT. Thus SQUARE CONPUT 5 SECTIONS SQUARE will draw a square with squares. Just saying SQUARE CONPUT SQUARE would put a square only at four corners. (See Illustration 14).





GRID will produce a layout of points on a grid:

```
R ← GRID RCD, RCDS
```

RCD contains three numbers specifying the number of rows, columns, and depth for the grid; RCDS has numbers specifying the row, column, and depth spacings. If only two numbers are specified, an evenly spaced, two-dimensional grid will be produced. The grid produced by this function is not meant to be drawn, but rather to be used in conjunction with CONPUT for placing other objects onto a grid.

C is a function which activates the cross hairs:

```
R ← C
```

After adjusting the x- and y-hairs, pushing the space bar gets you out of cross-hair mode and returns their xy-position.

Much of the usefulness of the Interart system derives from the capability of stringing together commands, allowing a large number of manipulations to be specified compactly. For example, a very useful operation is to use the cross hairs to reposition an object. By linking together the cross-hair input function C with the function PUT, one can manually relocate an object on the screen:

```
DRAW R ← C PUT OBJ
```

then rescale it twice as large:

```
DRAW 2 2 2 SCALE C PUT OBJ
```

and rotate it:

```
DRAW 2 2 2 SCALE C PUT OBJ ROT
  45, IP
```

Viewing Commands

DRAW is the major display function:

```
DRAW OBJ
```

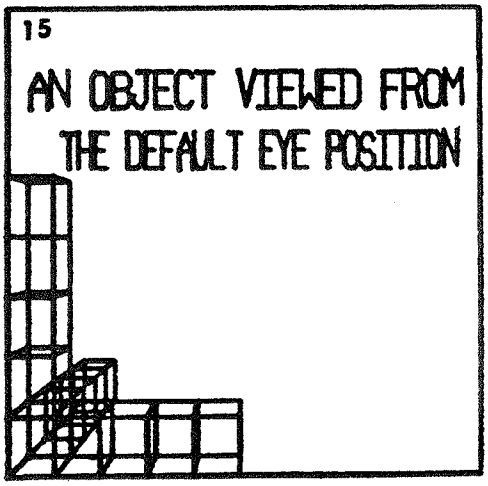
actually displays the data in OBJ, an (N,4)-matrix.

CDRAW creates the character vector which may be displayed later via quote-quad output:

```
R ← CDRAW OBJ
```

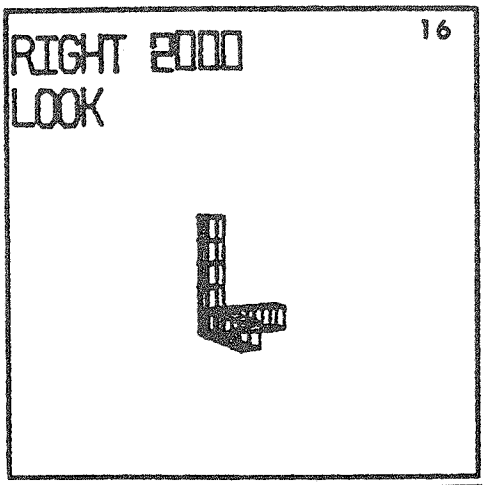
OBJ is graphical data, but the output of this function cannot conveniently be modified at a later time, so is generally used to store final products. This character vector takes up much less of the workspace than storage of the same picture as a numeric matrix. "Typing out" R will draw the picture created by CDRAW.

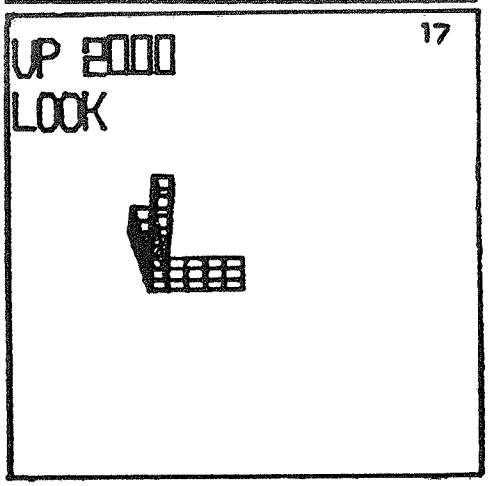
Another set of functions deals with the position of the "eye", or "viewpoint", for the display; its coordinates are stored in the variable VPOINT. The following functions deal with moving the eye (or camera) around and pointing it in different directions. (See Illustrations 15, 16, 17.)



REYESET resets the eye to default position. It sets EYEO to be an identity matrix. EYEO is a (3 3)-transformation matrix used for perspective calculations.

LOOK points the eye toward the point specified by ACENTER. ACENTER is a three-element vector. Its default value is 512 390 1000, approximately center screen.





OBSERVE OBJ points the eye toward the center of OBJ.

IN N moves the eye N units in the direction in which it is looking.

RIGHT N moves eye to the right of its present direction.

UP N moves eye above its present direction.

TILT N rotates eye N degrees to right around x-axis.

ROLL N rotates eye around y-axis.

PAN N rotates eye around z-axis.

Direct picture entry

The final component of the system is FOLLOW, the routine which allows one to use the cross hairs to draw pictures. The syntax is:

```
R ← FOLLOW
```

The use of FOLLOW is very similar to the use of an Etch-a-Sketch (TM) drawing board.

You move the cross hairs to a place on the screen and enter a command character, followed by a blank. Then move the cross hairs to a second location and enter another command. As this process continues, you construct a picture on the screen.

The FOLLOW sub-commands described below consist of one letter each. The commands are entered one at a time and are separated by a space.

M Moves the drawing point in pen-up state to the present cross-hair position. Generally the first command given.

D Draws a line from the last position entered to the present one. After pushing the space bar you may enter a number which will then be the z-value. Be careful to enter digits slowly, waiting for the cursor to space back after each digit.

L Fills in the triangular space between the last three points entered with intersecting "op-art" lines.

F Fits a curve to the last three points. The straight lines originally through those points are dropped from the data. Do a sub-command sequence Z V to see what has been produced.

C Determines what displayed point is closest to the cross hairs (for those with shaky hands).

U Undraws (erases) the last line (in case you make a mistake). If you type several U's, you will continue to delete preceding lines of the object.

Z Clears (zaps) the screen.

V Lets you view what you have so far. It is generally used after Z.

S Stops picture entry.

Conclusion

As part of an art course entitled Art and the Machine, taught at Rutgers College by Professor Philip Orenstein, students used the Interart system to produce a large variety of artworks. More importantly, art students who had never seen a computer before were getting involved with computing, and learning a great deal in the process. This system is most certainly not a state-of-the-art graphic system, or even extremely sophisticated in its internal operations. However, it has proved to be an effective instrument for involving novices in computing and for making these users more receptive to the possibilities of computers in the arts. It is the involvement of artists in computer graphics which brings to life the graphic systems evidenced every year by the "art" films shown at the SIGGRAPH conferences.

Acknowledgments

The Interart system was written by Dave Touretzky and Sanford Ressler at Rutgers University while both were undergraduates. The eye perspective and clipping routines were written by Josh Hall. The original system had assistance from Lyn Marantz.

The author would like to acknowledge the support of the Rutgers Computer Arts Center and, in particular, Philip Orenstein for his artistic guidance.

Sanford Ressler
Bell Laboratories
600 Mountain Avenue
Murray Hill, New Jersey
USA 07974

References

[1] Newman and Sproul. Principles of Interactive Computer Graphics, McGraw-Hill (1973).

[2] Gilman and Rose. APL: An Interactive Approach, Wiley (1976).