A SYSTEM FOR SCULPTING 3-D DATA

Richard E. Parent Computer Graphics Research **Group*** The Ohio State University Research Center 1314 Kinnear Road Columbus, Ohio 43212

A major research area in 3-D computer graphics is the inputting of complex descriptions. This paper describes our current attempt at solving that problem: creation of a sculptor's studio-line environment in which the user is provided with various tools to shape, cut and join objects. The emphasis of the implementation has been on naturalness and habitability. The issues involved in designing such a system, especially in a minicomputer-based color raster-scan animation environment, are discussed. The basic algorithms are described in some detail and a fast efficient implementation of a hidden-line algorithm is explained.

1. INTRODUCTION

A major problem in three-dimensional computer graphics is that of making available to the computer descriptions (or "models") of complex objects in a form suitable for various graphics manipulations. This paper represents an updated report of the research presented in (1). Some of the issues involved in the design of an interactive minicomputer-based 3-D data generation system are discussed, as is our current attempt at the creation of a sculptor's studio-like environment, in which the "sculptor" can create complex 3-D objects in the computer, as if moulding a piece of clay. The data generated is used by the ANIMA II system to create animation sequences which can be played in real time to a color video monitor.

There are two typical approaches to the 3-D data generation problem. In one, the objective is to recreate an existing object by constructing a definition of it in the graphics system by some means. This requires digitizing the surface of

This work was supported by the National Science Foundation under grant number DCR 74-00768.

Permission to make digital or hard copies of part or all of this work or personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. the object and inputting the coordinates of each point desired. In the other approach, the object definition is directly "created" or synthesized in the machine. In this case, the resultant object definition need not correspond to any physical counterpart, but rather, is specified for its visual attributes only. Of course, in practice one might adopt stances intermediate to these polarities such as reading in some "primitive" objects and then synthesizing more complex objects out of the primitive ones.

Our work in data generation is heavily influenced by the fact that, historically, the Computer Graphics Research Group (OGRG) at The Ohio State University has had as one of its aims for its graphics systems ease of use by artists and other non-programmers. Thus, the graphics language, Graphics Symbiosis System or GRASS (2), could be learned by persons with little computer background in a short time to create two-dimensional figures of nontrivial complexity. The WHATSISFACE system (3), developed at the **CGRG** enabled nonartists and nonprogrammers to draw on the CRT a facial image with remarkable likeness to a target photograph. The ANIMA II language currently in use at CCRG allows the nonprogrammer to easily specify complex time-dependent motions in 3-D for producing color animation sequences. In this historical context, we were naturally led to the notion of opening up the possibilities of the computer as a medium of expression for sculptors and animators of threedimensional objects.

2. APPROACHE TO 3-D DATA GENERATION

Elsewhere (1), we have discussed in detail various attempts at solving the data generation problem. Suffice to say, there can be no one solution to the problem of generating 3-D data. Inputting the description of a particular object one has at hand requires different techniques from synthesizing an object in the machine. Objects occurring in

138

engineering and architectural applications could be described by simpler techniques than free-form objects occurring in art contexts. These techniques necessarily place more of a burden on the user (4,5,6) usually in the form of preparing special 2-D projections of the data to be input, or on special hardware (7,8) such as scanning devices. Systems which allow the user to completely specify the object desired (2,9,10,11) have typically been severely restrictive in the class of objects describable or difficult to use.

Any approach to data generationmust also take into account the internal representation of objects and the class of objects to be handled. Baumgart (8) presents an excellent survey of the different representations that have been proposed and their relative advantages and disadvantages. We have chosen the planar polyhedral representation because it lends itself to hidden-line removal, visible surface shading and other graphics processing. However, since our aim is to "sculpt" interactively even objects which are only moderately complex conceptually can take on a quite complex polyhedral representation (e.g. a sphere). This fact places a great demand on the efficiency of all of the underlying routines. For instance, the standard implementations of hidden-line elimination routines become quite inefficient. As a result, we have written a powerful, fast, extremely efficient hidden-line elimination routine which is briefly described in Section 7.

3. COMPUTE GRAPHICS SCULPTOR'S STUDIO

Hardware

The CPU is a **PDP-11/45** with **96K** 16-bit words of memory, out of which 64K is magnetic core and 32K is MOS. In addition, an 88 megabyte disk as well as several disks of 2 megabytes each are available. The display is a Vector General with 4096 x 4096 addressable points. The peripherals include a joystick, 16 function buttons, sonic pen and 10 dials, all of which are used to interact with the system.

Command Language

When he is sculpting, a sculptor's natural mode of thinking is in terms of transforming a lump of clay in front of him by operating on it by means of various tools. It is possible to design a command language in which the graphics sculptor can specify what transformations he wants done on the object on the screen. We believe that for our aims, vis., to give maximum opportunities for the sculptor to be creative by letting him operate in a mode most compatible to his thinking, this kind of command language is unnatural. We should not force him to think in terms of numerical coordinates etc., but rather to feel as if he is operating on the object as directly as possible. For this reason, we make extensive use of the dials and function buttons. Scaling, rotating, translating, joining, intersecting, choice of "tools" etc., will be done by the sculptor by means of these analog devices or function buttons. The system still provides the sculptor with a "language" to communicate with it, but this is not a conventional command language.

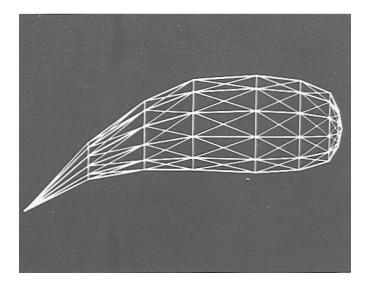
Primitive Object Input

The system provides the user with certain primitive objects, the sculptor might desire to generate his own primitive object. This can be done by selecting the sonic pen input routine in which the sculptor draws an orthogonal projection of the object desired. The system then forms the cylindrical extension of the projection forming a 3-D object. This object can then be used as is, or another projection of the object may be drawn with resulting cylindrical extension, and the two 3-D objects intersected at appropriate orientation forming an object with orthogonal views consistent with the two drawn.

If the user consistently uses certain higher-level primitives, i.e., objects which he has sculpted using the system's primitives, it is a simple matter to save these and then substitute them for the system's primitive objects during subsequent data generation sessions. Related to this is the ability by which a user may work on a "sculpture" but not complete it, even save it and then retrieve it the following day or the following week and finish his creation.

<u>The Scenario</u>

The scenario envisioned can be typified as follows. The sculptor either starts with a polyhedral object provided to him by the system or a primitive object created by him by the methods described in the previous section. In this case the sculptor is making a whale and thus selects a polyhedral approximation to a sphere. By means of dials the user can rotate, position and then scale the sphere along any axis to the appropriate size for the head and body. With group warp, the user can form the tail section by positioning and then attaching a cursor to a surface point and, in this case, pulling the point out (Fig. 1). An interpolation routine pulls neighboring points proportionally. How many points to warp in the group and the weighting function are selected on dials. The eyes are added simply by joining a cylinder (polyhedralapproximation, of course) which has been scaled, rotated and translated appropriately. The mouth is formed by using a wedge as a cutting tool. This can now be temporarily stored away so that the sculptor can create the fins. In this case the tail fin is a lowresolution sphere which is scaled and then cut with a wedge (Fig. 2). The side fins are half of the tail fin which have been scaled down more. The resulting object appears in Figure 3, which has been made from the primitive objects shown in Figure 4. The sculptor, however, is also an animator who wishes to use the whale in a sequence swimming. For this he must now make the whale in the extreme positions of swimming so that the inbetween positions can be interpolated by the animation language. The user does this by bending the tail section and the side fins. He can bend the tail by drawing a "skeleton" through the whale. The system maps the surface points of the whale onto the line segments of the skeleton. After the control points of the skeleton are repositioned by the animator/sculptor, the system remaps the surface points onto the skeleton maintaining their relative positions with respect to the skeletal line segments (Figs. 5 & 6).





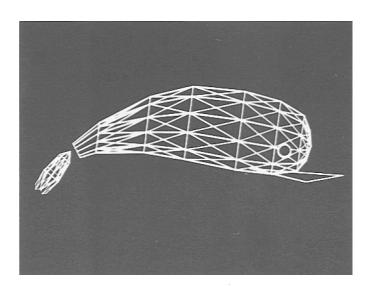


Figure 2

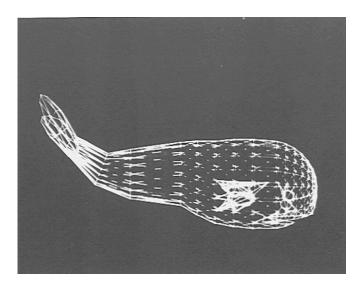
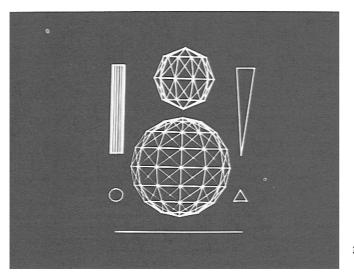


Figure 3





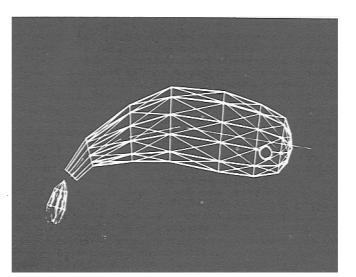


Figure 5

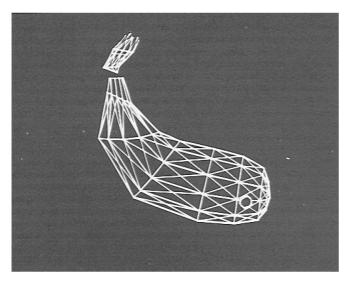


Figure 6

The sculptor can also, if necessary, fashion his own tools to do specific kinds of cutting, because after all, all the tools are "objects" and all operations of one object on another can be executed by the intersection routine.

It should be noted that as these operations are being done, the objects appear in the wire-frame mode (i.e., with hidden lines not deleted). This is due to the fact that the data structures for the input to the hidden-line processing routines are generated by a preprocessing routine which is not resident in memory except when specially called for. In order to increase the interaction, we have opted to do the sculpting operations in the wireframe mode, since the basic algorithms do not need this preprocessing. However, at any time, the sculptor can have the hidden-line eliminated version on the screen to increase his perceptual comfort before deciding on where to cut or how to shape the object. Another aspect of the system is that the sculptor, after a sequence of shapings, can go back to an earlier part of the sequence that he has saved on a LIFO stack, if he happens not to be satisfied with the current state of his sculpture.

We next give brief descriptions of the basic algorithms. The intersection, warping and bending algorithms are explained in Sections $\mathbf{4}$, 5 and 6, respectively. Hidden-line processing in a minicomputer environment has been generally slow, however, fast efficient hidden-line implementations are essential for many applications. Our hidden-line processing routine is described in Section 7.

4. THE INTERSECTION ALGORITHM

The intersection algorithm, as noted earlier, is an important part of the sculptor's studio. It operates on two overlapping closed polyhedra, say $\mathbf{0}_1$ and 02, and can calculate any of the four resulting polyhedra: the object defined by the intersection of 01 and 02, that defined by their union and either of the two objects formed when using one polyhedron to cut into the other.

Although intersectionalgorithms are not new, the one presented here is an advancement in generality over others found in the literature. Braid (9) describes two types of "addition" for two objects. Besides planar polyhedra, Braid also handles cylindrical surfaces when certain conditions are met (e.g. two curved surfaces cannot intersect). The first type of addition is the fusing of two objects when they meet at juxtaposed flat faces. The second type of addition is the actual intersection of two objects as long as one object is a (possibly transformed) cube or cylinder. Cutting or joining is treated as a property of an object in that a "negative" object cuts and a "positive" object joins to another positive object.

Baumgart (8) describes an algorithm which seems to be based on geometric principles similar to ours, but the data structures and actual algorithm are different. Further, his algorithm imposes the restriction that the objects intersected have convex polygons for faces. This not only makes the objects more visually complex but since the basic computation in calculating the intersection is comparing each edge against each face for intersection, the operations required are increased. Our algorithm imposes no such restriction and is quite general.

The data structure which is common to all of the routines is organized as a list of faces. Organization by faces facilitates the recording of properties of faces such as color, planar equation, the object the face belongs to, the subpart of the object the face belongs to, etc. Each face definition consists of a face header (FH), followed by a list of pointers to the vertices' coordinates (VP) as they appear clockwise when viewing the face from the "front" with the first vertex pointer repeating at the end of the list. Between each two vertex pointers in the face definition there is an edge header (EH). Thus, a triangle's definition takes the form: FH, **V**P₁, EH, VP2, EH, VP₃, **EH**, VP₁. In the implementationeach item is one word long. Notice that vertice pointers and headers alternate throughout the entire data structure since the face header for the next face follows immediately after the last vertex pointer of a preceeding face. The high order bit of a header indicates whether it is a face header or an edge header. The face header contains color information, object membership specification and an offset into a work area where the planar equation and pointers to linked lists for intersections and new edges are kept during the intersection calculation. The edge header contains a bit indicating whether it is the first or second occurrence of that edge. If it is the second occurrence, the remaining bits are a pointer back to the edge header of the first occurrence. The remaining bits of the header of the first occurrence of an edge are a pointer to a linked list of the intersections already found for that edge during the intersection calculation. The vertex pointers merely point into a list of coordinate triples.

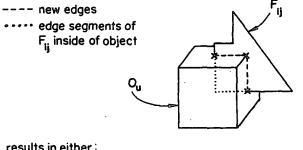
The algorithm uses the fact that the resulting object will have faces which are either portions of faces or entire faces of either **01** or 02. Each face of both objects must be compared against the other object (i.e., the object that the face is not a part of) to determine exactly what portions of the face lie inside of the other object. Once this is done for each face, the type of operation desired dictates which portions (those inside versus those outside of the other object) will be included in the final object and they completely define the final object. For example, if 01 cuts 02 the resultant object consists of those portions of faces of **O1** which are inside of O2 together with those portions of faces of 02 which are outside of 01.

As is readily apparent, the major issue is the organization of the edge-face intersection information during the actual intersection calculation. The data structure described allows for efficient organization of partial results obtained during the calculation. Faces and edges can be searched for efficiently and simply yet the data is compact which is a major consideration when working in a **32K** partition.

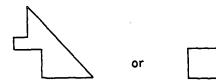
In order to simplify the description of the algorithm, let us define by $F_{1,j}$, the j-th face of object O_1 , and by $e_{1,j}^k$, the k-th edge of face $F_{1,j}$. The algorithm can now be informally described as follows:

MAIN: For i = 1, 2 and all j, give subroutine Al the face $F_{i,i}$ and information about the type of operation desired. At the conclusion of MAIN, all the faces and portions of faces belonging to the resulting object will have been generated.

Al: Given face F_{1j} , for F_{1j} and F_{uv} , $u \neq i$ and all v, give subroutine A2 the faces F_{1j} and F_{uv} . A2 compares the two faces to find edges of intersection and to find the intersection points along the edges $\mathbf{E}_{ij}^{\mathbf{k}}$ while noting whether the edge is "going into or "coming out of" the face \mathbf{F}_{uv} at that point. When face F_{ij} has been compared against all faces F_{uv} (i.e., all possible values of v) those portions of face Fij lying inside (or outside, depending on the operation) can be defined in terms of old edge segments of the e_{1j}^k and the new edges generated by the face-face intersections (see Diagram 1).



results in either:



depending on operation.

Diagram 1. Face-Object Intersection Calculation.

A2: Given faces F_{ij} and F_{uv} for e_{ij}^k and F_{uv} for all k, and for e_{uv}^m and F_{ij} for all m, this calculates the coordinates of the point of intersection between the edge and the face if such a point exists and sets up an information structure relating the edge, the intersection point and an indication of which vertex of the edge is in "front" of the face. The planar equation of the face is calculated so that it reflects the fact that faces are defined by a clockwise list of vertices if one looks at the face from the "front." All the intersection points generated, say p_1 , . . , p_n are sorted. They will be collinear and n even (see Diagram 2). Pairs (p_i, p_{i+1}) , i odd, define edges that belong to the new object. At the conclusion

of A2 the algorithm will have available new edges as well as information that will be used by Al to generate segments of edges of the original objects which are to be included in the resultant object.

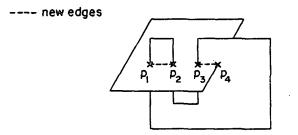


Diagram 2. Face-Face Intersection Calculation.

Figure 7a and b show two primitive objects: a sphere and a cube. Figures 8 through 11 show the four results obtainable from the intersection routine: the joining of the two (Figure 8), the intersecting of the two (Figure 9), the sphere cutting the cube (Figure 10) and the cube cutting the sphere (Figure 11). Figures 12 and 13 show example objects created using the system.

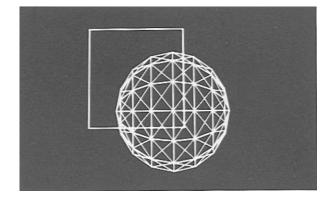


Figure 7a.

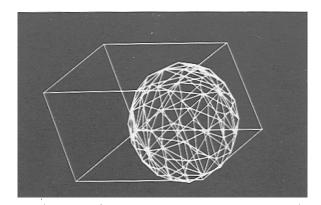


Figure 7b.

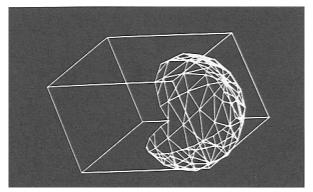


Figure 8.

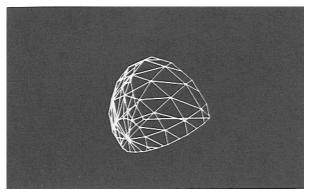


Figure 9.

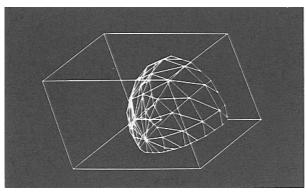


Figure 10.

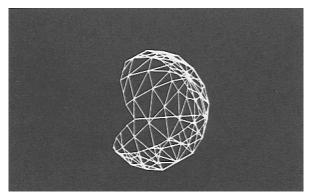


Figure 11.

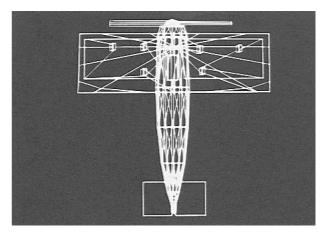


Figure 12.

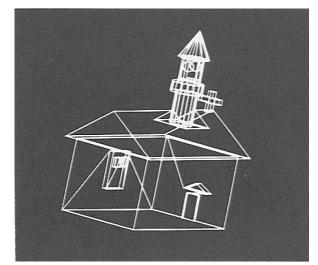


Figure 13.

5. WARPING ALGORITHM

The ability to warp an object is the ability to reposition a single point, or a group of points, independent of the rest of the object. As provided here, the user positions a cursor near the point to be warped, and then at the press of a button, can pick up the point and move it as he moves the cursor. The user can also specify, by means of a dial how many adjacent points to warp along with the initial point. The adjacency of a particular point is specified by the fewest number of edges which must be traversed from the initial point to reach the particular point. Also specified on a dial is the selection of a weighting function to use on the adjacent points being warped, giving the illusion of elasticity. The weighting functions possesses the attribute of being both easy to implement and visually what one would like to see.

If N (positive) is the adjacency indicator, all points N or less edges away from the initial point will be warped along with the initial point. If d_x represents the delta movement in the X-axis direction of the initial point and k (-64 \leq k \leq 63, for example) is the weighting function selector, a point I edges away from the initial point will be warped d^{I} along the X-axis where:

$$d_{X}^{I} = - \left(\left(1 - \left(\frac{I}{N+1} \right)^{k} \right) \overset{\circ}{=} d_{X} \qquad k > 0 \\ \left(1 - \frac{I}{N+1} \right) \overset{\circ}{=} d_{X} \qquad k = 0 \\ \left(1 - \frac{I}{N+1} \right)^{-k} \overset{\circ}{=} d_{X} \qquad k < 0 \end{cases}$$

6. BENDING ALGORITHM

Bending, as implemented here, is similar to the idea discussed by Wein (15) in controlling motion by key frame animation. In both, the basic principle is to alter the shape of an already existing object. Wein's skeletal representation "provides a definition of some coordinate space within which the image, described in relative coordinates, is distributed." The relative coordinates are based on a polygonal mesh over which the object is defined. This requires either an irregular mesh to be defined over the object or the object to be distorted over a regular mesh.

The skeleton used in bending is, conceptually, merely a collection of (possibly connected) 2-D edge segments which a 3-D object is mapped onto. There is no interference caused by multiple unconnected skeletons. This skeletal bending also easily lends itself to implementation in threespace, if one **is** willing to program the mathematics. Not only is the skeletal bending powerful, but at the same time it is easy for the user to employ since all he must be concerned with is drawing and manipulating the skeleton itself as opposed to a coordinate grid. Because of the interactive nature of the implementation the user can repeatedly reposition points in the skeleton and remap the surface until the correct amount of alteration is attained. The time taken for the initial mapping is required only once.

Ease-of-use stems from the fact that the system automatically maps each surface point to a skeletal edge segment. This is accomplished by first calculating "dividing" lines along the skeleton. These are lines which bisect the angle made at each skeletal edge-edge junction (a perpendicular is used at the ends of the skeleton). For each surface point, it is first determined which skeletal edges are possibilities for mapping. The surface point must lie between the dividing lines of a skeletal edge (and the dividing lines have not crossed each other) for that edge to be considered a possibility for mapping. For all such possible skeletal edges, the perpendicular distance from the surface point to the infinite line containing the skeletal edge is calculated and the closest skeletal edge is used for the mapping. The actual mapping consists of an indication of which skeletal edge is being used, the perpendicular distance as explained above and shown as distance ${f d}_1$ in Diagram 3, the length of the line parallel to the skeletal edge which passes through the surface point and lies between the dividing lines (d in Diagram 3), and the position of the surface point on that line (d in Diagram 3). This completely

locates a point relative to a skeletal edge. Figures 5 and 6 show a simple example of skeletal bending.

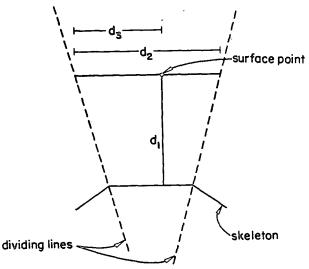


Diagram 3. Mapping of Surface Point to Skeletal Edge.

7. HIDDEN-LINE PROCESSING IN A MINICOMPUTER ENVIRONMENT

Sutherland et. al. (12) and Encarnacao (13) give surveys of available hidden-line elimination algorithms. Our hidden-line routine is based on Loutrel's algorithm (14) which employs what is called the path-of-edges technique. Though some changes have been made to the algorithm, mainly in the handling of special cases and the treatment of boundary vertices, we omit a detailed discussion of the algorithm in view of the easy availability of (14). Instead, we discuss the special considerations arising out of the fact that the implementation was to be close to real time (here meaning refresh of 30 frames per second and update less than one second per frame) in a minicomputer environment.

In our system, clipping in \boldsymbol{X} and \boldsymbol{Y} directions is provided and if desired, the scene can be displayed in perspective. All transformations are done by software. In addition, the following features are noteworthy. First, it is interactive. The user is provided with a joystick, various dials and buttons for interacting with the program. Dials control rotation, placement of the picture plane (for the perspective calculation) and scaling. The joystick is used for three-dimensional translation. Function buttons provide for, among other things, transformation speed changes, temporary halt, specification of scene or object for transformation, and exit. The second feature is the speed of processing, obtained by programming in assembler and performing all computations in integer arithmetic. Third, is the ability to transform (rotate and translate) independently each object in the scene as well as transforming the entire scene.

The limitations are two-fold. First, because of the size of the data structures used, the routine is restricted to handling less than nine hundred edges. This still allows for reasonably complex models. Second, due to the use of integer arithmetic on a 16-bit word machine, overflow and underflow errors occur at times. These, however, are usually few and far between and appear only as occasional flashes in normal operation.

The routine resides in a 32K partition and consists of two parts. The first part is the preprocessor which builds the data structures needed by the second part to efficiently calculate the visible edge segments. These data structures are built separately for each object and an object list is maintained. The requirement of real-time or close to real-time operation when the objects are being manipulated imposes the requirement that the preprocessor output should be basically independent of the vertex position data. Thus, the data structures have information about faces, edges and objects with respect to one another, and not with respect to the user, and refer to the vertices, not by their coordinates, but by pointers to them. Further, in order to minimize cumulative inaccuracies because of integer arithmetic, the position data are recomputed for each dial setting. An arbitrarily chosen "initial" position and size are operated on by transformation matrices whose parameters are set by the dial information.

Figures 14 and 15 give an idea of the results obtainable by the program. Figure 14a represents a 615-edge scene with hidden lines drawn, while Figure 14b is the same scene with hidden lines removed. Processing time was approximately nine seconds. Figure 15 shows an 81-edge scene with four objects, the time taken for this being about 0.1 second. It should be mentioned that edges are counted only once, not once for each face they appear in. These timings compare favorably with the five seconds required by Loutrel's implementation on a CDC 6600 for a 200-edge object.

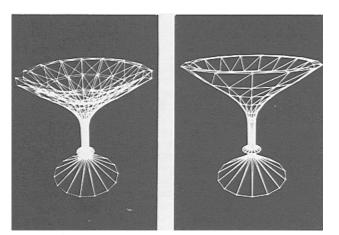
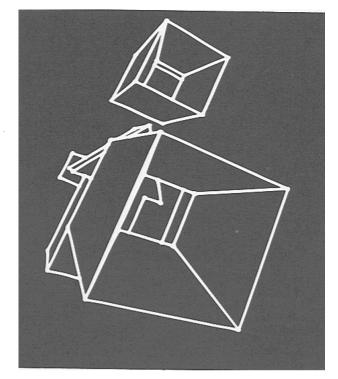


Figure 14a.

Figure 14b.





8. 3-D PAINT

Once our color video playback system became operational, it was obvious that it would be useful if the data generation system could make multi-colored objects. In order to keep the data generation interactive, however, it was necessary to retain the wire-frame representation of the objects. Thus, the user has no visual feedback of color information during the generation process, other than being able to selectively display only those faces of a certain color on a particular object.

Introducing color into the data generation system was facilitated by the fact that the data was already organized by faces. Three bits were set aside in the face header to specify color. When an object is input to the system, the user has the option of setting the color bits in each face header to a specific color, or he may leave them the way they are set. The intersection algorithm maintains the color information with those portions of the faces kept in the resulting object. Thus, a cutting operation leaves the color of the cutting object wherever that object cuts into the other object. This provides a very easy and natural means by which the user can build or sculpt a multi-colored object.

The second method of coloring an object to specifically pick out a face with a cursor and leave a particular color on that face. When the face is picked, the color bits in the face header are set according to three function buttons. Albeit this can be a bit tedious, it provides the significant ability of painting color designs on an object.

9. CONCLUDING REMARKS

It is hoped that the preceding discussion has given the reader a fair idea of the issues involved in the problem of generating 3-D data and of the merits of our attempt at a solution. The reader should especially note the kinds of human interactionpermitted. The system has been tailored for use in generating colored objects for animation and the operations available to the user reflect this.

Almost all of the limitations of the system stem from the 16-bit word length of the machine: the limit on complexity (about 2600 edges), inaccuracy in numerical calculations (an occasional problem), necessity for overlay structure of the program. A 32-bit minicomputer would solve many problems. Once allowing for the handling of greater complexity, however, the interactiveness of the system, and therefore, the naturalness of use would deteriorate with the higher-complexityobjects.

ACKNOWLEDGEMENTS

The author wishes to thank Professor Charles Csuri and the Computer Graphics Research Group for providing the facilities and a stimulating environment. I would also like to personally thank NSF Grant DCR 74-00768. I am indebteded to Ron Hackathorn for some of the data generation and photography.

REFERENCES:

- Chandrasekaran, B.; Parent, R. "Moulding Computer Clay - Steps Toward a Computer Graphics Sculptors' Studio," <u>Pattern Recogni-</u> tion and Artificial Intelligence, Academic Press, 1976.
- Defanti, T. A. "The Graphics Symbiosis System

 An Interactive Minicomputer Graphics
 Language Designed for Habitability and Extensibility," Ph.D. Thesis, The Ohio State
 University, 1973. Also technical report,
 Computer Graphics Research Group.
- Gillenson, M. L.; Chandrasekaran, B. "A Heuristic Strategy for Developing Human Facial Images on a CRT," <u>Pattern Recognition</u>, Vol. 7, 1975, **pp.** 187-196.
- Sutherland, I. E. "Three-DimensionalData Input by Tablet," <u>Proc. of the IEEE</u>, Vol. 62, No. 4, April 1974.
- Thorton, R. W. "MODEL Interactive Modeling in Three Dimensions Through Two-Dimensional Windows," Master Thesis, Cornell University, 1976.
- Lafue, G. "Computer Recognition of Three-Dimensional Objects from Orthogonal Views," Carnegie-MellonUniversity, Institute of Physical Planning, Research Report No. 56, September 1975.
- Shirai, Y.; Tsuji, S. "Extraction of the Line Drawings of 3-Dimensional Objects by Sequential **Illumination** from Several Directions,"

Second InternationalJoint Conference on Artificial Intelligence,London, September 1971.

- Baumgart, B. G. "Geometric Modeling for Computer Vision," Stanford University, Department of Computer Science, available from NTIS as AD-A002, October 1974.
- Braid, I. C. "The Synthesis of Solids Bounded by Many Faces," CACM, Vol. 18, No. 4, April 1975.
- Yoshimura, S.; Tsuda, J.; Hirano, C. "A Computer Animation Technique of 3-D Objects with Curved Surfaces," <u>Proc. of the 10th Annual</u> <u>Meeting of UAIDE</u>, Los Angeles, October 1971.
- 11. Davis, J. R.; Nagel, R.; Guber, W. "A Model Making and Display Technique for 3-D Pictures, <u>Proceedings of the 7th Annual Meeting of UAIDE</u>, San Francisco, October 1968.
- 12. Sutherland, I. E.; Sproull, R. F.; Schumacker, R. A. "A Characterization of Ten Hidden-Surface Algorithms," <u>ACM</u> Computing Surveys, Vol. 6, No. 1, 1974, pp. 1-55.
- 13. Encarnacao, J. L. "A Survey of and New Solutions to the Hidden Line Problem," Proc. <u>Interactive Computer Graphics Conference</u>, Delft, Holland, October 1970.
- 14. Loutrel, P. "A Solution to the Hidden Line Problem for Computer-Drawn Polyhedra," IEEE Trans. Computers, March 1970.
- 15. Burtyk, N.; Wein, M. "Interactive Skeleton Techniques for EnhancingMotion Dynamics in Key Frame Animation," <u>Comm. Association for</u> <u>Computing Machinery</u>, Vol. 19, No. 10, October 1976.
- 16. Myers, A. J. "An Efficient Visible Surface Program," Technical Report, Computer Graphics Research Group, The Ohio State University, July 1975.