Discussed are two computer programs for generating and realistically plotting any view of a three-dimensional object from the same object description, thereby simulating the viewpoints of a person moving around the object. Although the programs have been implemented—on an experimental basis—for digital plotting, the use of the underlying concepts for graphic display is contemplated.

Involved in the SIGHT *program are approaches to some of the most difficult problems in three-dimensional graphics—the hidden-line problem, approximating curved solids by polyhedra, and simulating degrees of surface transparency.*

The description of the program LEGER *emphasizes the design of data storage for the object description. This scheme allows the use of the same data for generating all views of the object. The data structure can be modified to adjust the dimensions of the scene and the relative orientations of the component parts.*

INTERACTIVE GRAPHICS IN DATA PROCESSING

Modeling in three dimensions

by A. Appel

Designing a computer graphics system for producing views of objects in three dimensions involves four problem areas: describing the object, storing the description, producing views that are recognizable as the object, and using the computer to generate geometric descriptions based upon nongeometric descriptions. When the object description and storage are based upon the two-dimensional picture plane, the recorded description typifies the actions of a draftsman in generating orthographic, perspective, or oblique views. Lines or shading of the picture are treated as the equivalent of the material object.

Modeling in three dimensions, the subject of this paper, has proved to be a difficult task for man. By three-dimensional modeling it is meant that the model description is invarient. Therefore, a given description (data) can be used for plotting any or all views of the object. Discussed first is an early three-dimensional graphics program, written in FORTRAN, known as SIGHT.[1] This program has advantages over previous attempts as well as shortcomings. Attempts to overcome the limitations of SIGHT led to the development of the program package LEGER, a description of which is presented next. LEGER is primarily a programming strategy and a group of FORTRAN subroutines that make possible three-dimensional per-

spective computer drawings of polyhedra with shadow and color. Both SIGHT and LEGER have been programmed to produce outputs on a digitally controlled plotter; however, interactive console output is contemplated.

Some advantages of three-dimensional modeling are: changes in the object description can be based upon functional requirements, and the views can be adjusted automatically; assembly drawings can be generated simply by bringing together component descriptions, as shown in Figure 1; and stored descriptions can be used for nongraphic purposes such as the generation of instructions for machine-tool control and for the calculation of weights, moments, and volumes. The computer can also be used to generate pictures having a greater range of graphic elements—tone and color—than is commonly used by draftsmen. Thus, the output graphics quality is significantly more realistic than in conventional drafting.

There are also disadvantages in working with a three-dimensional model. Except for principal views in orthographic projection, there is no simple, direct correspondence between the object description and the display. Traditional languages, graphic or verbal, do not enable economic and precise description of three-dimensional solids, and there is almost no precedent mathematics.

The SIGHT program

Examples of graphic output generated by the early system, SIGHT,[3] are shown in Figures 1 and 2. The SIGHT system was designed to draw free-standing polygons in space. A solid is described by a series of polygons completely enclosing a volume. The input (object) description is a list of cartesian coordinates that specify the vertex points of the polygons.

Figure 1 Bringing together component parts by SIGHT

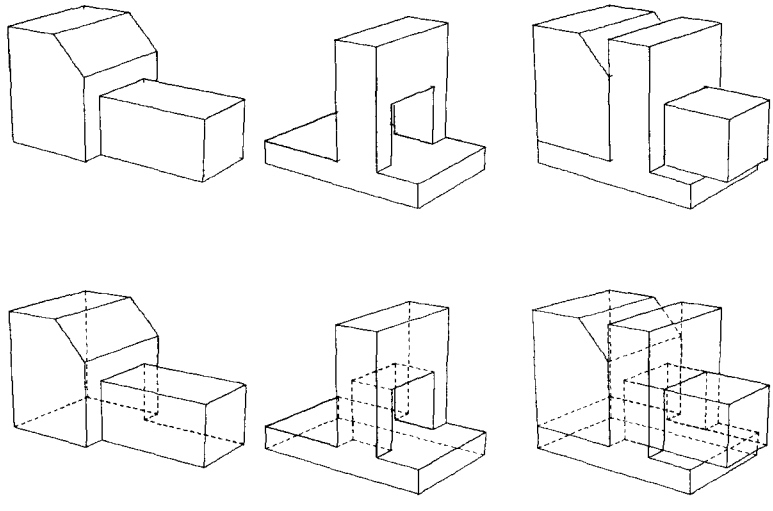
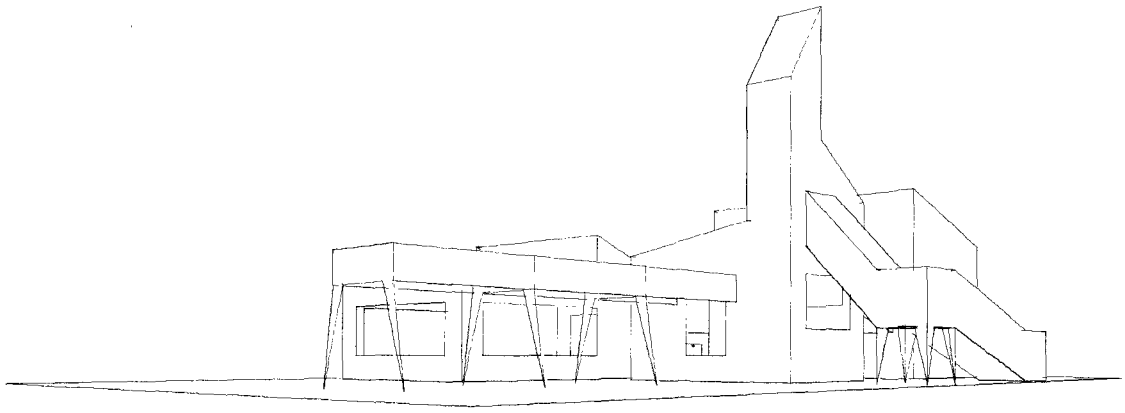


Figure 2 A graphic output of SIGHT



With this list, the bounding polygons can also be designated transparent, translucent, or opaque. In order to represent boundary planes as possessing some material substance, lines hidden from the observer are eliminated or dashed. All lines to be drawn are divided into short segments, and the line of sight to a point on each segment is tested to determine if the line of sight pierces an opaque surface. If the line of sight pierces a bounded surface, the segment is not drawn. This technique has been used in other drawing schemes for quadric surfaces.[4,5] Translucent surfaces hide every other line segment, while transparent surfaces do not hide segments.

The SIGHT system stores surface descriptions as rings of vertex coordinates and the equations of lines connecting the vertices. Simple to describe, SIGHT is somewhat cumbersome and time-consuming to use. However, it was and still is a useful medium for investigating computational graphic effects. For example, computer graphic techniques for shading line drawings were investigated using SIGHT as a framework. These graphic techniques were later introduced into the more efficient, but more complex, program LEGER.

modeling criteria

The SIGHT system has several limitations that are now discussed in order to gain some insight into criteria for evaluating computer-graphic modeling schemes. In SIGHT, restrictions on surfaces are that they may be bounded by one external polygon and by one internal coplanar polygon (a hole). In representing solids by a configuration of planes in space, this is a severe restriction on the descriptive vocabulary.

There is no attempt in SIGHT to form a hierarchical order for associating points with surfaces, lines, and objects or for associating surfaces with lines, points, and objects, etc. Such ordering is of great value in making the model description more specific and in reducing graphic output calculation time.

SIGHT allocates identical amounts of main storage to each sur-

face, i.e., the amount that is adequate to describe a surface externally bounded by twenty lines and internally bounded by ten lines. That these arbitrary dimensions hopefully would approximate the most complex surfaces proved false; there should be no such restrictions. Because the basic scheme is essentially a point-by-point one, calculation time increases with resolution or with the minimum size of line segment. The ideal graphic output scheme should enable perfect resolution for any size output.

Because vertex points are not tagged in a logical order, it is difficult to add subroutines that automatically generate surface descriptions. The fundamental assumption of SIGHT, namely, that most objects can be approximated by polyhedra, is questionable, but it has yet to be demonstrated that higher-order surfaces are more effective.[6]

The LEGER program

In designing LEGER, the main goal was to improve the strategy of input, storage, graphic output, and automatic description generation and manipulation. Improvements in capability of LEGER over SIGHT are that there are no restrictions on the polyhedra faces that LEGER can store, storage efficiency is improved, and the calculation time for line drawings is reduced by two orders of magnitude.

microdescription

There are two modes of object description in LEGER, microdescription and macrodescription. A microdescription of a picture gives all vertices and all lines that join them to form polygon-bounded planes. Those planes must enclose specified volumes. A LEGER microdescription of a picture is given by preparing two input card decks, printouts of which are shown by example in Tables 1 and 2. Table 1 is a series of tagged vertices and their spatial coordinates. The tags need not be in order, but the series must end with a 1000 tag. Essentially, a vertex list determines the dimensions of the picture; if the coordinates are changed, only the dimensions of the picture change, but not the geometric organization.

Table 2 gives the organization of the tagged vertices into lines, surfaces, and objects. The first column in Table 2 describes the origin and termination of geometric elements, where 999 signifies the end of a list of surface boundaries, 888 signifies the end of an object, and 777 indicates the end of all input cards. Other numbers, such as 1 in the first column, are used by the programmer to mark the object to which the card refers. The second column indicates the surface to which a card refers, but these numbers are not used for processing. The third column gives the card number, which is also not used for processing. The fourth column gives the tags of vertices in space and implies that a vertex so tagged is connected to the succeeding tagged vertex by a line. Zero tags in column four are used to break the vertex strings into boundary loops, where the vertex preceding the zero joins the first vertex following a preceding zero.

Table 1 A vertex list for LEGER

Vertex tag	*Coordinates*		
	x	*y*	*z*
1	0.0	0.0	0.0
2	1.0	0.0	0.0
3	1.0	0.0	1.0
4	0.0	0.0	1.0
5	0.0	0.2	1.0
6	1.0	0.2	1.0
7	1.0	0.2	0.2
8	0.8	0.2	0.2
9	0.8	0.2	0.8
10	0.2	0.2	0.8
11	0.2	0.2	0.2
12	0.0	0.2	0.2
13	0.2	0.0	0.2
14	0.2	0.0	0.8
15	0.8	0.0	0.8
16	0.8	0.0	0.2
17	1.0	1.0	0.0
18	0.0	1.0	0.0
19	0.0	1.0	0.2
20	1.0	1.0	0.2
21	0.8	0.8	0.0
22	0.2	0.8	0.0
47	0.6	0.4	0.4
48	0.4	0.4	0.4
49	0.6	0.0	0.4
50	0.4	0.0	0.4
51	0.4	0.0	0.6
52	0.6	0.0	0.6
53	0.6	0.4	0.6
54	0.4	0.4	0.6
1000			

In order to describe a picture, the designer should make at least one sketch of the component parts, such as shown in Figure 3. All vertices and surfaces are labeled in order that the sketch may help in debugging input cards. A sketch may also be used during program debugging to correct and improve calculations. The vertex list is made from such a sketch. No vertex is to be labeled zero. While a logical order is not necessary, some organization aids in checking coordinates. The list ends with a 1000 tag, as shown in Table 1. During execution of the program, the vertex list is read in from punched cards.

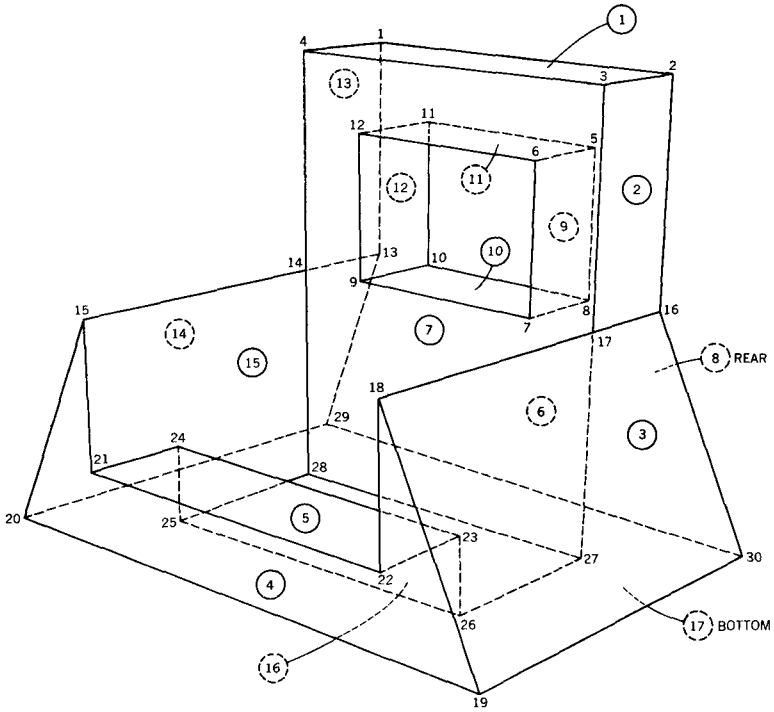
A geometry organization list such as shown in Table 2 is then prepared as follows:

1. For each external surface, taken one at a time, enter the external boundary points in the fourth column, using a particular sense of rotation around each surface. End each loop with a zero tag.

Table 2 A geometry organization list for LEGER

Picture element	*Picture organization*	*Surface*	*Card number*	*Vertex tag*
	1	1	1	3
	1	1	2	4
	1	1	3	5
	1	1	4	6
	1	1	5	0
End surface	999			
	1	2	6	5
	1	2	7	12
	1	2	8	11
	1	2	9	10
	1	2	10	9
	1	2	11	8
	1	2	12	7
	1	2	13	6
	1	2	14	0
End surface	999			
	1	3	15	3
	1	3	16	6
	1	3	17	7
	1	3	18	20
	1	3	20	2
	1	3	21	0
End surface	999			
	1	16	94	7
	1	16	95	8
	1	16	96	24
	1	16	97	23
	1	16	98	11
	1	16	99	12
	1	16	100	19
	1	16	101	20
	1	16	102	0
End surface	999			
End object	888			
	2	17	103	27
	2	17	104	28
	2	17	105	29
	2	17	106	30
	2	17	107	31
	2	17	107	32
	2	17	108	33
	2	17	109	34
	2	17	110	0
End surface	999			
	2	32	201	49
	2	32	202	50
	2	32	203	51
	2	32	204	52
	2	32	205	0
End surface	999			
End object	888			
End input cards	777			

Figure 3 Typical design sketch for LEGER



2. Enter all internal boundary vertices. For internal loops, the sense of rotation must be opposite that of the external sense of rotation. Again, end each loop with a zero tag.
3. When all coplanar loops for a particular surface have been entered, close the surface segment with a 999 tag.
4. When all surfaces that enclose a particular object have been entered, close the object segment with an 888 tag.
5. When the geometric representation is complete, close the organization list with a 777 tag.
6. As vertex points and segment markers are being recorded, additional card identifications can be made as suggested.

The organization list in Table 2 is used by the program to generate a storage list such as shown in Table 3. The structure of the storage list is augmented by the program with more information about the model, such as equations of planes and lines, surface orientation, internal and external corner identification, surface-to-line correspondence, object-to-surface correspondence, and the general sense of rotation of surface boundaries. From the storage list, the program then generates a string of element description blocks, as indicated in Figure 4. Such a data storage organization enables dense data packing, and construction and decomposition of the picture elements.

Table 3 A storage list

Surface count per object, indexed by object count $(1 \rightarrow IO)$	*Line count per surface, indexed by surface count* $(1 \rightarrow NS(IO) = IS)$	*Vertex point at start of line, indexed by line count* $(1 \rightarrow NV(NS(IO)) = IP)$
$NS(1)$	$NV(1)$	$K(1)$
		$K(2)$
		$K(3)$
		.
		.
		.
		$K(NV(1))$
	$NV(2)$	$K(NV(1)+1)$
	.	.
	.	.
	.	.
	$NV(NS(1))$	$K(NV(2))$
		.
		.
		.
$NS(2)$	$NV(NS(1)+1)$	
.	.	
.	.	
	.	
	$NV(NS(2))$	
	$NV(NS(2)+1)$	
	.	
	.	
	.	
$NS(IO)$	$NV(NS(IO-1)+1)$	
	.	
	.	
	.	
	$NV(NS(IO))$	$K(NV(IS-1)+1)$
		.
		.
		.
		$K(IP)$

IO = Number of objects IS = Number of surfaces IP = Number of lines

An advantage of decomposing a particular element is to be able to determine whether a point lies within a bounded surface. This requires that all lines in that surface that form the boundary loops be found. An advantage of constructing an element is to be able to determine the solid side of a surface that requires bringing together all the faces of the object associated with that surface. The parallel-string data organization is very efficient for both purposes. When LEGER was run on an IBM 7094, the object description of the picture shown in Figure 5 utilized main memory with an efficiency of about 85 percent of the FORTRAN-allocated storage locations.

Figure 4 Data storage organization

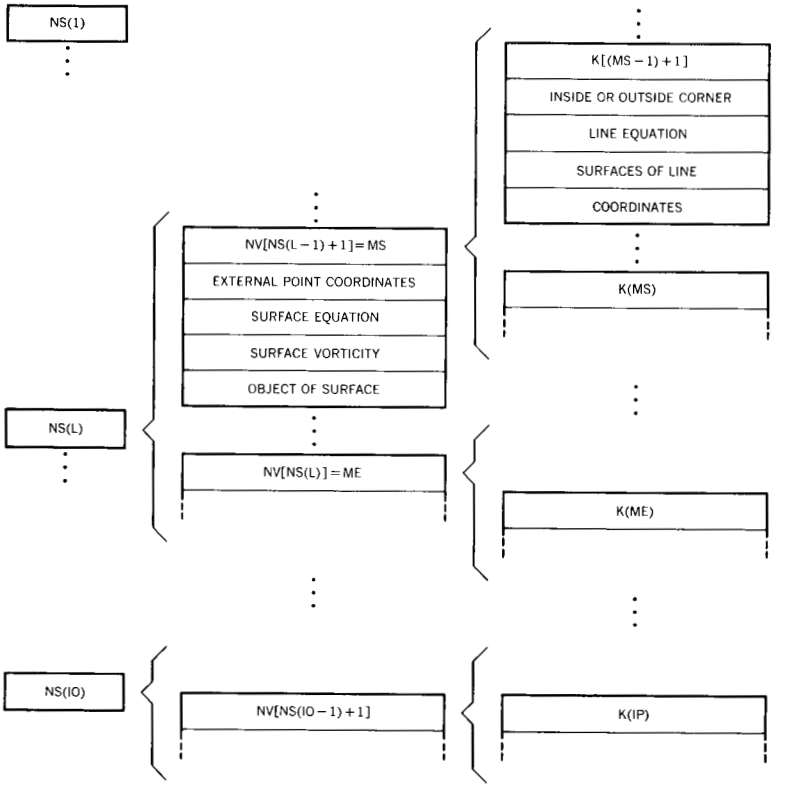


Figure 5 Use of microdescription and macrodescription for relating two pictures

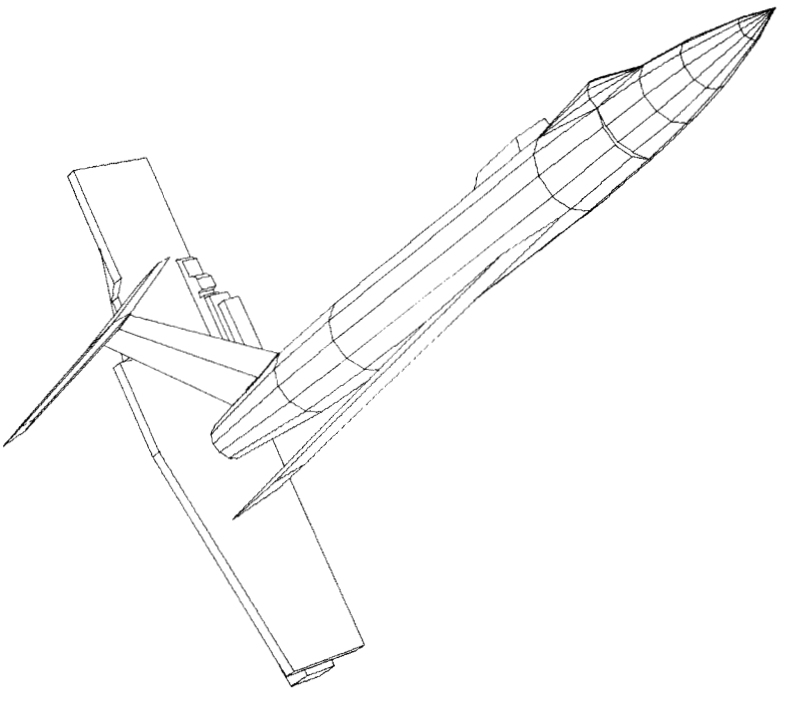
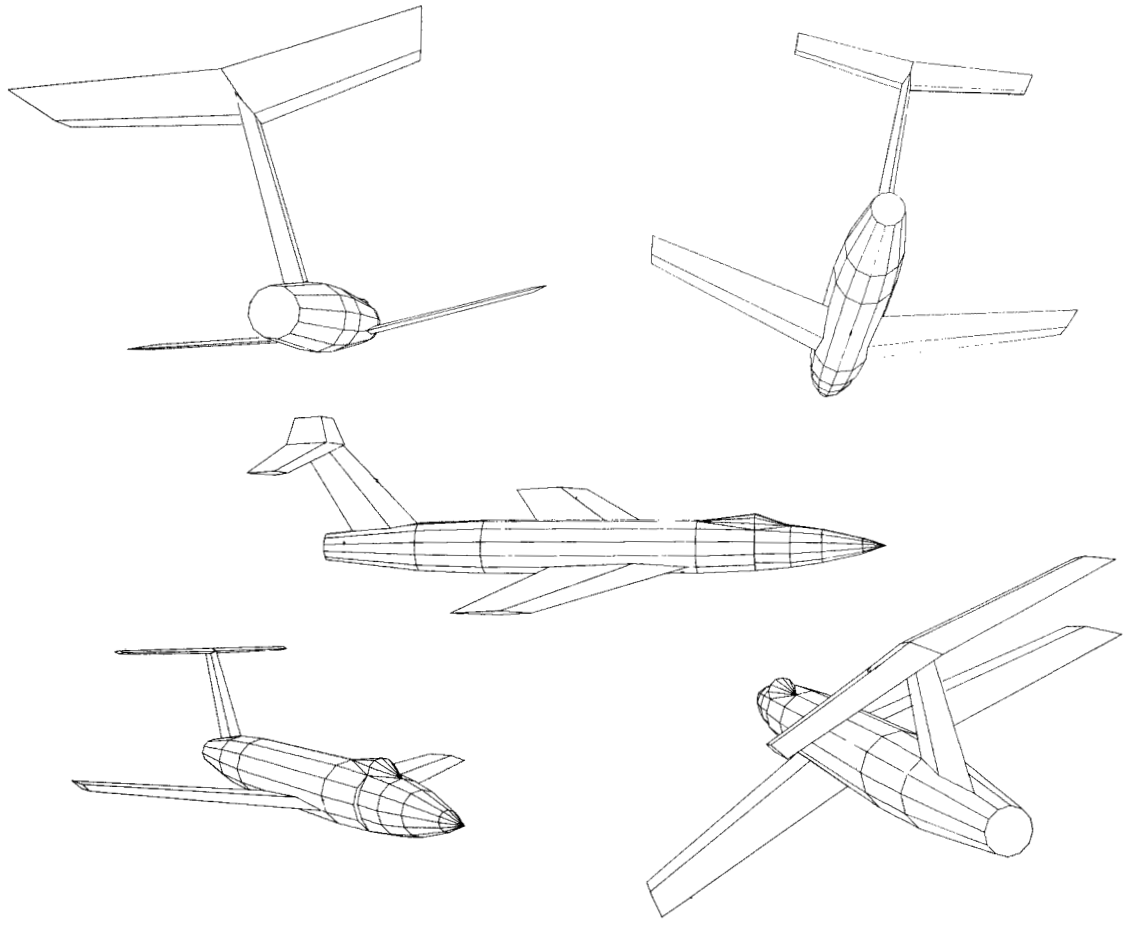


Figure 6 Views produced by the same object description



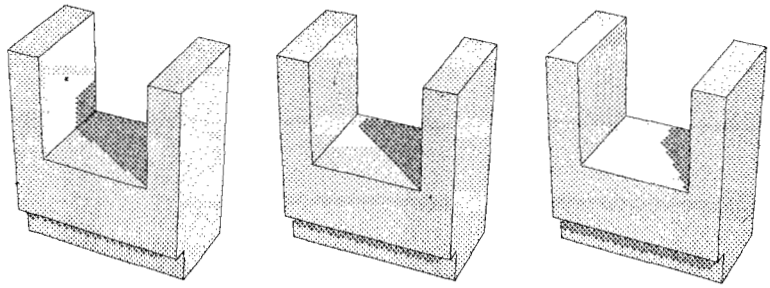
We now consider the second level of object description in LEGER, the macrodescription. This description consists of appended subroutines that can be called from the data input stream to generate a simulation of a microdescription or to manipulate a previously stored microdescription. Both uses of macrodescriptions were involved in processing Figure 5. A series of macrodescriptive subroutines generates the aircraft fuselage, and a rotation subroutine orients the aircraft carrier with respect to the airplane. This is necessary because of the complexity of the picture. Appended subroutines that create new objects must operate on both the vertex and the organization lists. Rotation and translation subroutines need only operate on the vertex list, since the geometric organization remains the same.

macrodescription

Concluding remarks

The overall rendering capability of LEGER is illustrated by Figures 5, 6, and 7. Specific techniques for generating these pictures are

Figure 7 One view with shading and moving light source



discussed elsewhere,[8,9] but it should be mentioned here that the major contribution of LEGER is in the design of data storage. The system has been expanded to include shadows, as shown in Figure 7, and to generate four-color separations for color printing. From the sample plotter outputs, it is seen that LEGER has the capability of controlling the plotter to produce drawings without gaps and runovers. LEGER also produces good tone control in shading and shadows.

It is desirable to improve the input-description method in LEGER. One possibility envisions developing methods of manipulating the vertex list, geometry organization list, and the data organization with more powerful statements. Typical problems that require solution are determining how two arbitrary planes intersect in space and how surface boundaries of polyhedra are affected as the polyhedra intersect. The use of LEGER in a real-time environment should also be investigated.

A computer graphics system must do useful work, but the act of designing the system is an educational process in itself. Such a system is also useful as a tool for the study of geometric properties of objects, which are independent of the sense of sight. Thus, the graphic-systems programmer becomes increasingly aware of the complexity and exactitude of the discipline of the geometry of perception.

CITED REFERENCES AND FOOTNOTES

1. Further information on SIGHT may be obtained from the author, IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.
2. Since LEGER represents polyhedra (and approximates nonplanar surfaces) by assemblies of polygons in space, it is appropriate to name the package after the pioneer cubist painter Fernand Léger (1881–1955). Further information about LEGER may be obtained from the author (Footnote 1).
3. A. Appel, *The Visibility Problem and Machine Rendering of Solids*, IBM Research Report RC 1618, IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598 (1966).
4. R. A. Weiss, "BE VISION, a package of IBM 7090 FORTRAN programs to draw orthographic views of combinations of plane and quadric surfaces," *Journal of the Association for Computing Machinery* **13**, No. 2, 194–204 (April 1966).
5. See the article in this issue by Y. Okaya.

6. R. Weiss, at the Bell Telephone Laboratories and members of the Mathematics Applications Group Incorporated, White Plains, New York, have developed systems based on quadric surfaces. S. A. Coons,[7] at the Massachusetts Institute of Technology, has worked on the description of free-form, mathematically complex surfaces. Coons generates contour mappings, which are most easily interpreted on a moving CRT display.
7. S. A. Coons, "Surfaces for computer-aided design of space forms," Technical Report MAC-TR-41, Massachusetts Institute of Technology, Cambridge, Massachusetts (1967).
8. A. Appel, "The notion of quantitative invisibility and the machine rendering of solids," *Proceedings of the 22nd National Conference of the Association for Computing Machinery* **P-67,** Thompson Book Company, Washington, D. C., 387–393 (1967).
9. A. Appel, "Some techniques for shading machine renderings of solids, "*AFIPS Conference Proceedings, Spring Joint Computer Conference* **32,** Thompson Book Company, Washington, D. C., 37–45 (1968).