PRADIS—An advanced programming system for 3-D-display

by J. ENCARNACAO

Heinrich-Hertz-Institute
Berlin/Germany

and

W. GILOI

University of Minnesota
Minneapolis, Minnesota

SUMMARY

The development of PRADIS was started in early 1968 as a vehicle to implement and evaluate new schemes of man-machine-interaction, data base organization, and hidden-line detection which were being developed in our laboratory. With time it evolved into a rather elaborate programming system for generation, construction, manipulation, and perspective display of arbitrary three-dimensional objects. All relevant object types such as polyhedrons or groups of polyhedrons with convex or concave structure, objects that are described by analytical expression, as well as the most general case of objects that are bounded by arbitrarily curved surfaces can be handled. Different modes of defining and inputting objects and different hidden-line suppression algorithms have been individually developed and implemented for these various types. Man-machine-communication has been facilitated by defining a simple command language for the user. The execution of any command can be initiated by pointing to the respective instruction in a command menu. Additionally, a keyboard (or the console typewriter) can be used for specification of parameters or the description of objects by analytical equations. Subroutines for scaling, translation, and rotation are part of the system as well as program modules which enable the user to construct three-dimensional objects by drawing two-dimensional views of them.

PRADIS is a modular system designed for implementation on a small to medium scale computer (our system has only 16 K of 24-bit words). Each module of the programming package is a functional entity, linked to other modules by an overlay technique. In the paper, the construction of the programming system and its modules, the different input modes, and the various hidden-line detection procedures are described. Representative pictures illustrate the performance of the system and its intrinsic procedures.

INTRODUCTION

PRADIS is a programming package written in FORTRAN II for the SDS 930 computer. As our configuration has only 16 K of core memory, the whole system has been divided into 16 segments (links). Each segment is a self-sustained entity that takes care of a certain complex of the total set of functions. The various subsystems exchange data via a shared common area.

The first module is a control module that interprets the command language by which a user defines what class of objects he is going to deal with and what input mode he will use. The control module then calls the respective link which provides a data input in the required mode, generates objects, and executes instructions in an interactive mode. By setting certain sense switches, the user finally terminates the job or calls the control module again for further execution of other jobs. Figure 1 gives a simplified block diagram of the entire system which we will discuss more in detail in the following sections.
Figure 1a—Block diagram of the system PRADIS
CLASSES OF OBJECTS

The classes of objects that can be selected are:

- Groups of polyhedrons of convex structure (KOVSTR)
- Groups of polyhedrons of concave structure (KOKSTR)
- Groups of objects that are defined by functions of two variables which have to be given in analytical form (FUNGLE)
- Groups of objects described by functions of two variables which are "primitive functions" (i.e., they are stored under a function name in the library) (FUNAME)
- Groups of objects that are neither polyhedrons nor defined by analytical functions but that are composed by "surface patches" as defined by COONS' theory (FLADAR)
(In the present version of PRADIS ‘FUNGLE’ and ‘FUNAME’ are intrinsically the same module. However, in an extended version they have to be distinguished.)

Figure 2 shows the “menu” or “light-button field” which enables the user to select the required module by pointing first to the PROTYP command and then to a selection instruction. The selected problem type (and the respective module) is indicated by an arrow. An erroneous selection can be corrected by a delete instruction (LOESCH). Otherwise, the next instruction menu is called, depending on the object class the user wants to deal with.

MODES OF INPUT

Each module provides several input modes for the generation and manipulation of objects as shown in Table 1.

Some comments shall be given. ZIDDI and BAUKAS are able to call each other to facilitate the interactive procedure of constructing objects of convex structure. ANKOR, so to speak, is a sort of preprocessor which calls automatically LIDREI after the analysis of the inputted objects has been performed (of course, this step can be skipped by entering LIDREI directly). LIDREI includes as well as BASTRU the 4×4-matrix operation that takes care of scaling, perspective display, and rotation of objects or groups of objects. In either case the hidden lines are evaluated and suppressed. Naturally, the employed hidden-line detection strategies have to be different.

In the case of objects defined by equations the input procedure consists in typing in either the defining equation (FUNGLE) or the functions name (FUNAME).

For the fourth class of objects (i.e., general three-dimensional objects composed by surface patches) we have two optional modes of object definition and input: FLEIN 1 and FLEIN 2. FLEIN 1 provides the definition of general three-dimensional objects following COONS’ method; i.e., by putting in the characteristic nodes and slope vectors.

FLEIN 2 is based on the input of nine points for a surface patch definition, providing a much easier way of defining objects. This technique will be described in a later section.

CONSTRUCTION AND DRAWING OF THREE-DIMENSIONAL OBJECTS

To our knowledge, the first program which served that purpose was developed by T. JOHNSON2 of M.I.T. in connection with the SKETCHPAD project. In this program, an object is represented by three views

<table>
<thead>
<tr>
<th>object type</th>
<th>input commands</th>
<th>visibility procedure</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>KOVSTR</td>
<td>ZIDDI</td>
<td>FLAVER</td>
<td>Display of objects bounded by plane surfaces with convex structure</td>
</tr>
<tr>
<td></td>
<td>BAUKAS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BASTRU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KOKSTR</td>
<td>ANKOR</td>
<td>FLAKA</td>
<td>Display of objects bounded by plane surfaces with concave structure</td>
</tr>
<tr>
<td></td>
<td>DRELIS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIDREI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FUNGLE</td>
<td>—</td>
<td>FLAVI</td>
<td>Display of objects defined by functions of z variables or by library-functions</td>
</tr>
<tr>
<td>FUNAME</td>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLADAR</td>
<td>FLEIN 1</td>
<td>FLEIN 2</td>
<td>Display of objects that are composed by ‘surface patches’ (Coons)</td>
</tr>
<tr>
<td></td>
<td>PURAKU</td>
<td>TANECK</td>
<td></td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
and one section which have to be drawn by light-pen. Recently, two more programs have been reported, but they do not offer the generality and flexibility of ZIDDI and BAUKAS.

ZIDDI takes advantage of the man-machine-
BAUKAS (from German "Baukasten" = building block) provides the possibility to call primitives which are stored in a font and use them as components of a more complex object. The primitives have been constructed at an earlier time by the aid of ZIDDI. Figure 5 illustrates the way BAUKAS works. A table is going to be constructed using the primitives and the coordinate transform procedures provided by BAUKAS.

Communication by which a display system with light-pen facilitates the task of constructing three-dimensional objects by drawing two-dimensional views. Additionally, the user has a set of powerful commands at hand for executing coordinate transforms and other operations. Using the light-pen, the user marks dots which are subsequently connected by lines. These lines may constitute surfaces which, again, bound bodies. Dots and lines can be arbitrarily added or erased. All coordinate transform procedures (scaling, translation, rotation) can be called by pointing to the respective command in the command menu, and they may be arbitrarily used to generate and modify figures and objects. In space, a point is defined by three Cartesian coordinates $x$, $y$, $z$. However, on the two-dimensional screen of the scope we have only two coordinates. This discrepancy is overcome by defining two areas on the screen: the $yz$-plane and the $xz$-plane. Both planes are automatically connected by linking related coordinates (for better accuracy the user can ask the system to display only one of the two planes in double size and to draw an auxiliary grid).

Concave structures are not always uniquely defined by two views, but by rotating them into an appropriate position a unique object definition can be obtained. Some simpler examples of objects that have been constructed with the aid of ZIDDI are shown in Figures 3 and 4.

Figure 5—Illustration of BAUKAS

Figure 5 (cont'd)
to be constructed by using cubes as primitives. By deforming the cubes in one way, we get the table-top; by deforming it in a different way, we get the legs. On top of the table we put two other primitives: the pyramids and the cube on top of them. The flexibility of BAUKAS is illustrated by some examples given in Figure 6.

ANALYSIS OF POLYHEDRONS WITH CONCAVE STRUCTURE

Concave objects are those which are bounded by one or more concave surfaces or which have cavities. A group of disconnected objects (with empty space in between) is—as a group—concave, too. It is evident, that in contrast to convex structures some difficulties may arise with respect to establishing unique relations between all points, lines, surfaces, and objects of a structure. However, these difficulties are overcome by the following procedure. Prerequisite for it is, that a list is available which gives the three spatial coordinates of all points, and which marks all connections between them. Such a list may have been put in by the user or it will be generated by the program.

The most general case is that the user has three drawings which show three different (two-dimensional) views of the object. In that case the user has only to provide for each drawing a list of the two coordinate values of all relevant points, together with a second list of all visible connections. The order in which the points are numbered is arbitrary. From these lists the program generates a list of all points in space in the following way: the y-coordinate of a point in the xy-list, for example, is taken, and the yz-list is searched for the same y-coordinate. If the search is successful, the xz-list is searched for a point, whose x- and z-coordinates correspond respectively with the x-coordinate of the point taken from the xy-list and the z-coordinate of the point taken from the yz-list. Triples of x-, y-, and z-values obtained in such a way constitute the coordinates of a (spatial) point of the three-dimensional object. Eventually, the list of all spatial points is sorted with respect to increasing values of x and a running index is assigned to each triple.

The second task is to generate a list of all lines of the three-dimensional objects from the input of their two-dimensional views. A similar search procedure as in the case of the generation of the lists of spatial points is used, i.e., the three lists which belong to the two-dimensional views are searched for certain equivalences. The difference is that now an equivalence is not given by the coordinate values but by their running index. The point is that first of all the arbitrary numbering in the list of all points of the two-dimensional views has to be replaced by the corresponding running index taken from the ordered list of spatial points. In order to make that feasible, the lists of the points of the two-dimen-

Figure 6—Examples of objects constructed with the aid of BAUKAS
DEFINITION OF OBJECTS BY ANALYTICAL FUNCTIONS

There are three different ways of defining three-dimensional objects by analytical expressions:

1. the implicit form \( F(x, y, z) = 0 \)
2. the explicit form \( z = f(x, y) \)
3. the parametric form \( x = f_1(u, v), y = f_2(u, v), z = f_3(u, v) \).

Though the implicit form is the most simple one, it does not lead to an efficient computer algorithm. The explicit form, on the other hand, does not provide a unique description (e.g., for \( z = 0 \) we have \( f(x, y) = 0 \) which does not only define all the points of a plane but all the points on a cylinder). So we have to rely on the parametric form.

In order to facilitate the input procedure, the system enables the user to key-in his equations on the display keyboard (or console typewriter, respectively). After having the equations analyzed and transformed, the system calls an appropriate algorithm for function evaluation. Furthermore, the user will be asked to specify all necessary reference coordinates, scaling factors, and other parameters. Syntactical errors will be detected, and error messages will be output. The respective language which we use is simple; its syntax has been described elsewhere.\(^7,\(^8\)

The typed-in equations are processed in the following steps:

1. loading into a buffer storage area
2. equation analysis and coding
3. assignment of values to the parameteric variables \((u, v)\)
4. assignment of values to the coordinate variables \((x, y, z)\)
5. calculation of the right-hand expressions and storage of the obtained results.

In the second step the equations are converted into polish notation, and a push-down stack is set up. An operation matrix defines whether or not an actual operation can be immediately performed on two subsequent variables. The first row denotes the actual operators and the first column their predecessors. The meaning of the numbers in the matrix is

1: Operation cannot be executed. Bring operator into the operator stack.
2: Both operators have same priority. Execute operations.
3: Actual operator has higher priority than predecessor. Execute operation.
4: Erase parentheses (or brackets). If the expression between parentheses (brackets) is a subroutine for function evaluation, it is executed.
5: Recognition of a terminal operator.
6: Invalid combination of operators. Output error message.

OPERATION MATRIX

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>-</th>
<th>*</th>
<th>/</th>
<th>(</th>
<th>)</th>
<th>A</th>
<th>Δ</th>
<th>†</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>2 2 1</td>
<td>1 1 1</td>
<td>1 3 3</td>
<td>1 3 1</td>
<td></td>
<td></td>
<td>A Δ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>2 2 1</td>
<td>1 1 1</td>
<td>3 3 1</td>
<td>1 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>3 3 2</td>
<td>2 1 1</td>
<td>3 3 1</td>
<td>3 1 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>3 3 2</td>
<td>2 1 1</td>
<td>3 3 1</td>
<td>3 1 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>4 4 1</td>
<td>6 1 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>)</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>4 4 1</td>
<td>6 1 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>6 6 6</td>
<td>6 1 1</td>
<td>6 6 6</td>
<td>6 6 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blank</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>6 6 1</td>
<td>5 1 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>†</td>
<td>3 3 3</td>
<td>3 1 1</td>
<td>3 3 1</td>
<td>3 1 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The above outlined analysis program has been written in FORTRAN II which, however, had to be supplemented by bit and character manipulation subroutines. The submodule FLAVI can be applied for hidden line detection and suppression to the thus calculated functions. Figure 10 shows some objects generated in that way.

GENERAL OBJECTS

According to COONS’ theory and the nine point modification we have three main modes of defining bounding surfaces of general three-dimensional objects:

1. The boundary contours have to be specified; blending functions (functions that define how the boundaries are connected by curved surfaces) are defined once and for all.

2. Four corner points and eight corresponding slope vectors have to be specified for each surface patch; fixed blending functions are given, as in the first method.

3. For each surface patch nine characteristic points have to be specified. No blending functions are required.

Generation of boundaries by polynomial approximation (FLEIN 1, PURAKU)

The first approach offers the advantage of dealing with larger entities than the “surface patches” in the second and third approach. Each boundary is approximated as an entity by a polynomial approximation. For each contour $n+1$ quintuples have to be specified; $n$ being the degree of the approximation polynomial.
The five values of each quintuple give the three coordinates of a point through which the polynomial passes as well as two parameters for the specification of the two degrees of freedom of the surface. Thus, the degree of partitioning the objects can be lower.

However, the method shows some deficiencies. (One serious deficiency is, for example, that it is not very well suited for an interactive use of the display system but requires rather elaborate preliminary work.) It is left to the user to define the degree of each approximation polynomial; if he is not very familiar with the performance of polynomial approximations, large errors may occur. Of course, the best performance is obtained if the boundaries are conic sections.

As mentioned before, for each construction element called “surface patch” of a curved surface we have to specify four corner points and eight corresponding slope vectors, two for each boundary contour. These vectors, of course, are defined in space by specifying their components. In FLEIN 1, the intrinsic blending functions are of third order. Figure 11 shows an example for an object defined in such a way. In order to obtain a smooth representation of an object, it is necessary that the user develops a good feeling of how to specify slope vectors in an optimum way. Hence, the application of this method requires a lot of experience and preliminary work with pencil and paper.

The nine point method (FLEIN 2)

Here, a surface patch is defined by specifying nine characteristic points: the four corner points, four points on the boundaries (one point on each one), and one point in the center of the surface. Figure 12 gives an example: the surface element is a quadrilateral, the center point is on the same level (has the same z-coordinate) as the four corner points, and the points on the boundaries are on a higher level. In Figure 13, in contrast, the center point has been raised to the level of the boundary points.

The nine point method works better the more information about the objects one has (points of inflexion, boundary, etc.). Boundaries are approximated in an optimum way if they are parabolic. Any time a break point or a point of inflexion occurs, a new surface patch has to be defined. Hence, the nine point method leads to a finer partitioning than COONS’ method. Another disadvantage is that it does not guarantee...
a continuous transition between surface patches. A big advantage, on the other hand, is given by the fact that no blending functions have to be defined and no spatial vectors have to be specified, both requirements expecting very much from the user with respect to his power of imagination.

Additionally, the nine point method requires much less time for preparation and execution of a problem. It is particularly suited for plotters or displays that work incrementally. The man-machine-interaction is excellent. Figure 14 shows as an example the body of a well-known automobile.

COORDINATE TRANSFORMS

PRADIS uses homogeneous coordinates. A coordinate transform is performed by multiplying the matrix of homogeneous point coordinates by the transform matrix TM—as indicated in Figure 15. The transform parameters have to be specified by the user.

VISIBILITY CRITERIA

Visibility criteria enable the program to detect (and erase) hidden lines. For a satisfactory display of three-
dimensional objects, the hidden line suppression is mandatory. We distinguish between three types of criteria: point tests, surface tests, and combined point/surface tests. In the case of surface tests the primitive which is tested is a surface element as the name indicates. The surface test indicates only, whether or not a line is visible as an entity; i.e., a possible mutual hiding of two surface elements cannot be taken into account. Hence, it is not applicable to concave structures.

Points test partition a line into small segments that are only considered to be visible if none of the arbitrarily selected points on this segment is hidden by a surface element. The point test procedure is generally applicable, however, it requires a prohibitive amount of execution time. Combined point/surface tests try to combine the advantages and avoid the disadvantages of point tests and surface tests.

In PRADIS, we use some new schemes which we developed. The implementation of our hidden line detection algorithms are called FLAVER, FLAKA, and FLAVI. The gist of these three algorithms shall be summarized in the following.

(1) FLAVER—FLAVER uses a hidden line algorithm that is applicable to convex structures. The procedure consists of three main steps. In the first step, the angle between the line-of-sight and the normal line on the surface is calculated resulting in a criterium for totally hidden surfaces. In the second step, a priority is assigned to each one of the remaining surfaces. Thus, the sequence is defined in which the mutual hiding will be determined. This is done in the third step by investigating whether or not the terminating points of a line fall into a surface with higher priority than that of the surface the line belongs to. If only one of the terminating points is hidden, the point of intersection between this line and the surface boundary is determined. This point terminates the visible part of the line. The point is that only surfaces with higher priority than that of the surface the respective line belongs to are to be taken into account, thus saving considerably computation time.

(2) FLAKA—FLAKA works on concave structures; i.e., it works in the more general case where the polyhedrons may have cavities and/or a group of disconnected polyhedrons occurs. In this case, the angle between the line-of-sight and the normal line, unfortunately, does not provide a simple criterium for determining whether a surface is visible. This difficulty is overcome by dividing in a preliminary step all the surfaces of a structure into triangles. On these triangles the last two steps of FLAVER can be applied. The partitioning of all surfaces of the structure results in additional lines (if a rectangle is partitioned into two triangles, an additional line is generated which is the rectangle diagonal). Special care has to be taken to suppress these additional lines (therefore, the triangles are of course not visible).

(3) FLAVI—The intrinsic hidden line detection algorithm of FLAVI takes care of objects that are defined by analytical expressions—be these expressions defined by the user in form of equations or generated by the program according to COONS' theory. The procedure works as follows: First of all a Cartesian coordinate grid of 11 by 11 lines is superposed. If the visibility of the point of intersection between 2(Δu−Δv)-segments has to be determined, only the surface patches have to be considered that have no empty intersection with the respective grid area. Subsequently, surface patches to be taken into account are partitioned into triangles, and the z-coordinates of these triangles are calculated at values of x and y given by the coordinates of the intersection point. If a triangle has a z-coordinate which is greater than that of the intersection point, it hides that point. If no such triangle exists, the point of intersection is visible. All points of intersection of all lines u=constant and v=constant have to be tested in that way. If one of two connected points is visible and the other one hidden, more points located on the connecting contour have to be examined.

ACKNOWLEDGMENTS

The authors are grateful indebted to Mssrs. Grosskopf, Schulz, Castro, Mahramzadeh, Hunger, Burmeister, Eckert, and Kniepen who contributed by their thesis work under our supervision to the successful implementation of the PRADIS system.

REFERENCES

1 H J BEISTER
Zur Technik der Segmentierung von Programmen bei Rechenanlagen mit kleinem Speicher edv 7/69
2 T JOHNSON
SKETCHPAD III—A computer program for drawing in three dimensions
Proc SJCC 1963

3 D EWING D TEDD
A display technique for manipulating engineering drawing in 3 dimensions
CG 70—London

4 R J HUBBOLD
TDD—A 3 D drawing program
CG 70—London

5 J ENCARNACAO J HUNGER
Das interaktive Zeichnen und Entwerfen in drei Dimensionen
Elektronische Rechenanlagen 1 February 1971

6 J CASTRO H MAHRAMZADEH
Beschreibung, Untersuchung und Erprobung der Zerlegung beliebiger Körper mit konkaver Struktur in konvexe Ebenen
Studienarbeit TU Berlin 1969

7 J ENCARNACAO
Untersuchungen zum Problem der räumlichen Darstellungen auf ebenen Bildschirmen
Doktorarbeit TU Berlin 1970

8 R ECKERT
Untersuchung und Verallgemeinerung von FLAVI
Studienarbeit
TU Berlin 1970

9 J ENCARNACAO
A solution to the problem of reconstructing objects from three views and displaying them using a new hidden-line algorithm
DATAFAI 71 March 1971 Nottingham

10 S A COONS
Surfaces for computer aided design of space forms

11 G SCHULZ
Flächeneingabe durch Eckpunkte und Tangentenvektoren und durch neun spezielle Punkte
Diplomarbeit TU Berlin 1970

12 J ENCARNACAO
A survey of and new solutions for the hidden-line problem
CG-Symposium Delft 1970

13 G GROSSKOPP
Beschreibung, Untersuchung und Erprobung der Visibilitätsverfahren FLAVER und FLAKA Diplomarbeit
TU Berlin 1970