A graphical extension of Pascal enables programmers to build complex 3-D objects and to describe their motions.

The Use of High-Level 3-D Graphical Types in the Mira Animation System

Nadia Magnenat-Thalmann and Daniel Thalmann Université de Montréal

A good computer graphics and animation language is not just a structured language with graphical procedure calls; it must also provide a way to structure data. It has already been shown that type-oriented languages offer a good way to check data, and good data structures are just as important as good control statements. Pascal¹ is an excellent type-oriented language, and certainly a popular one. New types can be defined on the basis of existing types. By introducing three-dimensional graphical types into Pascal, and by providing the means to define any drawing with them, we obtain a powerful, structured computer graphics and animation language.

In recent years, the abstract data type² has emerged as an excellent approach to the design of quality software. The term "abstract data type" means the modularized data type with formal semantics. In the abstract data type approach, the design of an application programming system begins with its specification as a set of complex abstract data types. Then a refinement process is repeated until basic graphical types are obtained.

Pascal is a good structured language, but it has no abstraction capability. For this reason, we have defined primitive abstract graphical data types³ called *figures* that are similar to the classes in Simula. We have introduced them as a graphical extension of Pascal called Mira-2D.⁴ These concepts have been implemented with positive results, and numerous applications have been developed. Figure 1 shows a picture produced by Mira-2D programs.

Then 3-D graphical types were added to Pascal to create Mira-3D. This language allows the user to easily build very complex three-dimensional objects. Drawings



Figure 1. Robots in a picture produced by Mira-2D programs.

can be produced in a wireframe mode or in a hiddenlines-removed and shaded mode. The next step was to introduce abstract data types to describe motions of threedimensional objects (*actor* types) and of virtual cameras (*camera* types). This article describes our three-dimensional, high-level graphical data types and their impact on computer animation methodology.

Mira-3D and the figure types

As a three-dimensional graphical extension of Pascal, Mira-3D includes

- · three-dimensional vector arithmetic,
- · graphical statements,
- image transformations,
- viewing transformations (perspectives and parallel projections), and
- · standard procedures and functions.

The most important tool in this graphical extension is the 3-D graphical type—the *figure* type. The syntax is described in Figure 2. The word "figure" is a keyword. The formal parameter section, the declaration, and the body are similar to the corresponding elements in a procedure.

To define a figure type, we must (1) find the characteristics of the figure, which become the parameters, and (2) find the algorithm that allows the user to build the figure with the help of the parameters.

To build the figures, new statements have been introduced: *moveabs*, *moverel*, *lineabs*, and *linerel* to draw vectors, and *include* to define an existing figure as part of a new one. For example, a pyramid with four vertices can be defined as

Attributes can be given to a figure; the most important are line style, intensity, line width, and color. Graphical variables are defined as variables of graphical type. Four





fundamental statements allow the user to manipulate these variables:

create <figure> (<actual parameter list>)

(3) draw < figure>

(4) erase <figure>

The *create* operation dynamically creates the figure by giving values to the corresponding type parameters; the figure can then be drawn, erased, or deleted.

A certain number of standard and frequently used 3-D figure types have been introduced. The most important are

- (1) the line type,
- (2) the planar figure types: *triangle, square, circle, ellipse,...,*
- (3) the plane type defined by three vectors,
- (4) the box type defined by four vectors,
- (5) the sphere type defined by a center C and radius R (approximated by polygons),
- (6) the five regular polyhedra types: tetrahedron, cube, octahedron, dodecahedron, icosahedron, and
- (7) the surface types: cylinder, cone, parabolic cone, parametric surface, revolution surface, Bezier surface, Coons surface, and B-spline surface.

Hidden lines can be removed and images can be shaded.

The Mira-3D extension has been implemented by developing a preprocessor that consists of a 6000-sourceline program in Pascal developed on a CDC Cyber 173 and a DEC VAX-780. The output is a standard Pascal program. The runtime library is a 10,000-Pascal-sourceline library that is almost device independent. However, the dependent part can be easily rewritten and has already been adapted for HP2648A, Tektronix 4027, DEC GIGI, Norpak, and AED terminals, Hewlett-Packard plotters, and various printers. Only the viewing procedures have been taken from the GSPC Core system.⁵

Two case studies: giraffes and bridges

Here is an example of a giraffe that has been drawn using the previously defined 3-D graphical types. The giraffe type has been defined as

GIRAFFE = figure (AXIS, ORIGIN, NECKORIG, TAIL:VECTOR; MOVEMENT: INTEGER; NECKORIEN-TATION: ORIENTATION);

where

AXIS is a vector that defines the giraffe's orientation, ORIGIN is the origin of the new axis system,

NECKORIG is the origin of the giraffe's neck,

TAIL is the vector that gives the tail direction,

MOVEMENT is an integer that defines specific movements for the giraffe, and

NECKORIENTATION is an array of three vectors defining the orientation of the neck, which is divided into three parts.

The giraffe is modeled as follows:

head	box
neck	series of circular cones
thigh	truncated elliptic cone
kneecap	sphere
foreleg	cylinder
hoof	two bodies of revolution
tail	a cylinder and a truncated con

Figure 3 shows two views of a giraffe.

Bridges are difficult structures to define using parameters because they are very dependent on such



Figure 3. Two views of a giraffe drawn using 3-D graphical types.

hard-to-describe factors as landscape, building materials, building techniques, and the imagination of architects and engineers. A procedure for drawing any bridge would require more than 100 parameters. The *bridge*⁶ graphical type presented here allows the user to reproduce almost any kind of bridge—generally with a high degree of detail. The bridge type is defined as

type BRIDGE = figure (ROAD: ROADTYPE; PIER: PIERTYPE; SUPER: SUPER: TYPE; PARAPET: PARATYPE; ORIENTA-TION: REAL)

Road defines the characteristics of the road: height, width, length, and number of joints. *Piers* are composed using a complex system of basic figures such as cylinders, cones, parallelepipeds, circles, parabolic arcs, and lines. *Super* defines the bridge superstructure; simple bridge, traditional lattice bridge, lattice bridge with two peaks, cantilever bridge, steel-arch bridge, suspension bridge. The characteristics of the *parapet* are the number of supports and the parapet height. *Orientation* defines the orientation of the bridge by rotating it around the y-axis.

An example of a bridge drawing produced using the bridge graphical type is given in Figure 4.

Abstract graphical types in computer animation

Dream Flight⁷ is a 3-D computer-animated fiction film completely produced by computer. It is the story of a creature living on another planet and dreaming that he flies across space like a bird and arrives on Earth. Typical scenes are set in Paris and, as in Figures 5 and 6, New York. Others show natural scenes such as ocean, trees, or birds. The film was developed using the Mira-3D programming language to create objects and motions.



Figure 4. A bridge drawing produced using the bridge graphical type.



Figure 5. Arrival in New York (frame from Dream Flight).



Figure 6. The Verrazano-Narrows Bridge (frame from Dream Flight).



Figure 7. Sitting in a forest (frame from Dream Flight).

We can use the first scene of the film to explain how static and dynamic objects are constructed. In this scene (Figure 7), Hipi is sitting in a forest at night and throws stones into a small pond. He sees a bird and imagines that he is flying like the bird. This scene involves dynamic objects like Hipi, the bird, and the waves on top of the water. It also involves several static objects: stones, trees, the pond, the horizon, and the spherical sky with stars.

A bird is represented by an abstract graphical type:

type BIRD = *figure* (FRAME:INTEGER; H:HALF-BODY; W:WING; C,D:VECTOR);

where H is the right halfbody, W the right wing, C the rotation center of the right wing, and D the direction of rotation of this wing. Since the right wing always starts in the maximal vertical position, it is necessary to determine only the angle of rotation downward. This angle depends on the frame. To determine the angle, we use a rotation with the Catmull⁸ acceleration/deceleration law.

Here is an excerpt of the code that is executed when a variable of bird type is created.

BIRD = figure	e (FRAME:INTEGER;
H:H/	ALFBODY: W:WING:
C,D:	VECTOR);
var	
· RI	ELATIVE:0CYCLE:
FF	RACTION.
BI	ETA :REAL;
W	2 :WING:
RI	GHTPART.
LI	EFTPART:FIG;
begin	
RI	ELATIVE: = FRAME mod CYCLE:
if	RELATIVE > (CYCLE div 2) then
	RELATIVE: = CYCLE-RELATIVE:
FF	RACTION: = (RELATIVE*2)/
	CYCLE;
BE	TA: = LAW(ACCEDECE,
	ANGLEMAX, FRACTION);
RC	DTATION(W,C,BETA,D,W2);
U	NION (H,W2,RIGHTPART);
de	lete H,W2;
SY	'MYZ(RIGHTPART, LEFTPART);
inc	lude RIGHTPART, LEFTPART
end:	

ent

type

In this code, *cycle* is the number of frames required to make one wing flap up and down, *frame* is the current frame number, *beta* is the rotation angle of the wing, *anglemax* is the maximum angle, and *frac* is the phase fraction. For example, a bird is created and drawn according to the following sequence:

```
procedure DRAWBIRD (FRAME:INTEGER);
var FIRSTBIRD:BIRD;
begin
create FIRSTBIRD (FRAME, RIGHTBODY,
RIGHTWING, C, D);
TRANSLATION (FIRSTBIRD, < <0, 0, FRAME •
BIRDSTEP > >, FIRSTBIRD);
draw FIRSTBIRD;
delete FIRSTBIRD;
end;
```

In this procedure, a variable *firstbird* is dynamically created in memory; then, a translation is applied to this variable. The translation vector <<0,0,FRAME*BIRDSTEP>> has a z-component, depending on the

frame number. The *draw* statement means that the content of the variable is displayed. Finally, the variable is deleted in memory.

The parameters in the *create* statement correspond to the parameters in the definition of the bird type.

Trees are also described by 3-D graphical types:

type TREE = figure (var BRANCHES:TEXT; NBRANCHES:INTEGER; POSITION:VECTOR; HEIGHT, LENGTH:REAL);

where *branches* is a file of kinds of branches, *nbranches* is the number of branches, *position* is the position of the trunk, *height* is the height of the trunk, and *length* is the length of the branches.

A forest can be defined by the following declaration:

var FOREST: array [1 . . NBTREES] of TREE

Actor and camera data types

Abstract graphical types like figures are very useful, but they do not have their own animation. For this reason we have designed animated basic data types, actor data types, and camera data types. The design of these types has been influenced by the work on data abstraction^{2,9} and by research in actor systems. ¹⁰⁻¹³ We could say that a figure is an actor without animation.

The following example will introduce these concepts:

```
type STRANGESQUARE =
   actor (A,B,C,D:VECTOR);
     time T1 . . T2;
     const V = ... (*speed*)
     type
        TVEC = animated VECTOR (P1, P2; VECTOR);
                   val P1 . . P2; (*limits of the
                               animated vector*)
                   time T1 . . T2;
                  law P1+V*CLOCK
                end
     var VERTEX:TVEC;
     begin
        init VERTEX (C, (A + C)/2);
        moveabs A;
        lineabs B, VERTEX, D, A
     end:
```

This defines a square that is animated between the times T1 and T2. One vertex moves in the direction of the center with a constant speed V. This vertex is defined as an animated vector, that is, an animated basic type. This concept is an extension of the Newton concept defined in the animation language ASAS. 12,13 An animated basic type is a basic type defined in such a way that each variable of this type (called an "animated basic variable") is animated. Three basic types can be animated: integer, real, and vector types. An animated type is defined by giving the starting and ending values of the number or the vector, the starting and ending times, and a function or law that describes how the value varies with time. During the specified time interval, variables of animated basic types are automatically updated to the next value according to the function.

Initialization of the animated basic variables is performed by the *init* statement. At this stage the initial values of the parameters are given: e.g., *init* VERTEX (C, (A+C)/2).

Animated basic variables can be used wherever a variable of the same basic type would be used.

An actor type is an animated abstract graphical data type. The syntax is very similar to the figure syntax, except that the lifetime limits of the actor must be specified. Animated basic types and variables can be defined within an actor type, which is the best way of using parameters that vary in time. An actor can reference another actor, but the latter cannot be defined within the first actor.

An actor can be constructed using *figures*, and these figures can be manipulated. The actor block can contain any declaration except actor and camera types, and it can contain any Mira-3D statement. However, the viewing procedures cannot be invoked because it is not the role of an actor to manipulate visual parameters. The time interval exactly defines when the actor exists. Here is an excerpt of the bird actor type:

```
type
  BIRD = actor (H:HALFBODY; W:WING; C,D:VECTOR;
          T1,T2:REAL);
         time T1 . . T2;
         type
           ANG = animated REAL;
                    val 0.0 . . ANGLEMAX;
                    time T1 . . T2;
                    law ACCDEC( . . . )
                  end;
           POS = animated VECTOR;
                    val ORIGIN . . UNLIMITED;
                    time T1 . . T2:
                    law BIRDSTEP*BIRDSPEED
                  end;
         var
           TRANS
                         :POS:
           BETA
                         :ANG;
           W2
                         :WING;
           RIGHTPART,
           LEFTPART
                         :FIG;
         begin
           init TRANS;
           init BETA;
           ROTATION(W,C,BETA,D,W2);
           UNION(H,W2,RIGHTPART);
           DELETE H,W2;
           SYMYZ(RIGHTPART,LEFTPART);
           include RIGHTPART, LEFTPART
        end:
```

A bird has only to be initialized by the statement

init FIRSTBIRD (H, W, C, D, 10, 16)

rather than by the procedure *drawbird* shown earlier. The bird is started at time 10 and stopped at time 16. Animation frames are produced automatically, and the bird can be synchronized with other actors.

A camera type is also an animated abstract type. Its syntax is exactly the same as the syntax of an actor type, but the actor keyword is replaced by the camera keyword. Time limits have the same meaning as for an actor. Animated basic types and variables can be defined within a camera type, but no actor types or other camera types can be used. The statements cannot manipulate figures and actors because this is not the role of a camera. The goal of a camera type is to define the values of the visual parameters and how they vary with time. Typically, statements in a camera type are viewing-procedure calls; they can be those of the GSPC Core system, and their parameters can of course be animated variables.

Animated basic types, actor types, and camera types have been introduced in the Cinemira¹⁴ language, which is a high-level computer animation language whose graphical basis is Mira-3D. A Cinemira script is a subprogram dedicated to computer animation, and it consists of a sequence of scenes. Each scene has a name and is made up of a sequence of statements that serve mainly to initialize actors, cameras, and decor. The decor is a collection of graphical objects that do not move or change during the entire scene. In Cinemira, a decor is defined by the statement *decor* < figure list >, where the figure list is an enumeration of figure variables. The advantage of the decor statement is that it puts together all the objects that have no animation. For example:

create HOUSE(...); create SKY; create SUN(...); decor HOUSE, SKY, SUN;

The *shoot* statement performs the shooting phase, and decor, actors, and cameras are automatically placed during this phase. The shoot statement can therefore take a very simple form: *shoot until* < expression >, where the expression is the upper time limit of the scene in seconds. The lower limit is the upper limit of the previous scene (0 at the beginning).

The Mira animation system

The production of three-dimensional computeranimated films using a graphical programming language is time consuming. For example, it took 14 months to produce the 13-minute film *Dream Flight*. User-friendly



Figure 8. The Mira animation system.

interactive systems allow the user to develop 3-D computer-animated film more rapidly. However, they impose limits on the creativity of artists who would like to exploit all the possibilities of a computer. This is why we designed the animation system shown in Figure 8. This system comprises user-friendly interactive programs and the programming languages Mira-3D and Cinemira. Basic figures, stored as complex lists of descriptors, can be created in three ways:

- by using the Horizon 3-D interactive graphics editor,
- (2) by programming with the Mira-3D graphical language, and
- (3) by using a 3-D interactive reconstruction program.

Mutan,¹⁵ a multiple-track animator system for motion synchronization, is an interactive system for independently animating three-dimensional graphical objects. It is in fact a 3-D key-frame animation system with in-betweens calculations.

Mutan is also a good tool for synchronizing motion with sound, light, or speech. To make this possible, Mutan handles several tracks at a time, with all animation constraints for a graphical object recorded on each track. A program in Cinemira will be able to read and write Mutan tracks that correspond to actors. This means that a scene produced by Cinemira can be viewed and modified by Mutan. An actor can even be added with Mutan as a new track.

A new film called *Nirvana* is currently being developed using the Mira animation system. Figure 9 shows frames from this film. The Corvette was built by the 3-D reconstruction program. Scenes are produced by programming and using the Mutan system, and images can be shaded, as shown in Figure 10.

We have shown with the Mira-3D language how highlevel 3-D types can be used in computer graphics and computer animation. We have also described actor and camera data types. The programmer is now able to use high-level 3-D graphical types to build objects and to use actor and camera data types to describe motions in animation. With this approach, complex computer animation scenes and special effects can be produced in an elegant, efficient, and reliable manner.

Acknowledgments

We would like to express our gratitude to the referees for their helpful comments. We are grateful to Christian Pellegrini, Alain Choquette, and Mario Fortin, who have worked on the Mira implementation, and to the students who participated in the creation of the images used in this article: Philippe Bergeron, Nicolas Chourot, Louis Langlois, Francois Marceau, Daniel Ouimet, and Christian Roy. We would also like to thank Ann Laporte, who revised the English text. The research was supported by the Natural Sciences and Engineering Council of Canada and by Quebec FCAC Foundation.



Figure 9. Explosion of a Corvette (three frames from Nirvana).

References

- K. Jensen and N. Wirth, PASCAL User Manual and Report, Springer-Verlag, New York, 1974.
- J. Guttag, "Abstract Data Types and the Development of Data Structures," Comm. ACM, Vol. 20, No. 6, 1977.
- D. Thalmann and N. Magnenat-Thalmann, "Design and Implementation of Abstract Graphical Data Types," Proc. Compsac '79, IEEE Computer Society, Los Alamitos, Calif., pp. 519-524.
- N. Magnenat-Thalmann and D. Thalmann, "A Graphical PASCAL Extension Based on Graphical Types," Soft-



ware—Practice and Experience, Vol. 11, 1981, John Wiley & Sons, New York, pp. 53-62.

 "Status Report of the Graphics Standards Planning Committee of ACM/ SIGGRAPH," Computer Graphics, (Proc. Siggraph '79), Vol. 13, No. 3, Aug. 1979.



- D. Thalmann, D. Ouimet, and C. Roy, "Drawing Bridges by Computer," *Computer-Aided Design*, Vol. 14, No. 4, 1982, pp. 195-199.
- D. Thalmann, N. Magnenat-Thalmann, and P. Bergeron, "Dream Flight: A Fictional Film Produced by 3-D Computer Animation," *Proc. Computer Graphics* '82, Online Publications, Ltd., pp. 352-367.
- E. Catmull, "The Problems of Computer-Assisted Animation," Computer Graphics (Proc. Siggraph '78), Vol. 12, No. 3, Aug. 1978, ACM, pp. 348-353.
- B. Liskov and S. Zilles, "Programming with Abstract Data Types," Proc. Sigplan Symp. Very High Level Languages, Mar. 1974.
- C. Hewitt and R. Atkinson, "Parallelism and Synchronization in Actor Systems," Proc. ACM Symp. Principles of Programming Languages, 1977, pp. 267-280.
- I. Greif and C. Hewitt, "Actor Semantics of PLANNER-73," Proc. ACM Sigplan-Sigact Conf., Jan. 1975.
- C. W. Reynolds, "Computer Animation with Scripts and Actors," Computer Graphics (Proc. Siggraph '82), pp. 289-296.
- C. W. Reynolds, "Computer Animation in the World of Actors and Scripts," thesis, Massachusetts Institute of Technology, 1978.
- N. Magnenat-Thalmann and D. Thalmann, "CINE-MIRA: A 3D Computer Animation Language Based on Actor and Camera Data Types," Pub. Dép. Inform. et Rech. Opér., Université de Montréal, 1983.
- D. Fortin, J. F. Lamy, and D. Thalmann, "A Multiple Track Animator System for Motion Synchronization," *Proc. ACM Siggraph-Sigart Workshop on Motion*, 1983, pp. 180-186.



Nadia Magnenat-Thalmann is an associate professor at the Graduate Business School of the Université de Montréal in Canada. From 1977 to 1979, she taught information science courses at the Université Laval in Quebec. She is also president of Matamor Software, Inc., of Montreal, a firm that offers systems services in computer graphics and computer animation. Her research interests include the design of

human-machine interactions, graphical turnkey systems, and computer animation. The author of numerous publications, she has written several textbooks in computer science.

Magnenat-Thalmann holds a BA in psychology and an MSc in biochemistry from the University of Geneva in Switzerland. In 1977, she obtained a PhD related to computer graphics. She is a member of ACM, Eurographics, and the Institute of Management Sciences.



Daniel Thalmann is an associate professor at the Université de Montréal, Canada. He taught at the Swiss Federal Institute of Technology in Lausanne and at the University of Nebraska at Lincoln. He presently teaches computer graphics, animation techniques, and software engineering. His research fields are computer graphics, computer animation, and the design of programming languages. He has pub-

lished more than 30 papers in these fields and is author or coauthor of six books.

Thalmann received his diploma in nuclear physics and his PhD in computer science from the University of Geneva, Switzerland. He is a director of the Canadian Man-Machine Communication Society, and a member of the IEEE Computer Society, ACM, NCGA, and Siggraph. He also serves as editor of the journal *Informatique et Images*.



Topics discussed in over 135 papers include simulation aids, testing for LSI-MOS, VLSI placement, and the human factor in color CAD systems. 815 pp.

Order #447

PROCEEDINGS—20th Design Automation Conference

June 27-29, 1983

Members-\$33.00 Nonmembers-\$66.00



The theme for this year was "Automating Intelligent Behavior: Application & Frontiers." Session topics included AI applications, computer vision, intelligent interfaces and speech recognition, man-machine systems, and knowledge representation and management. 285 pp.

Order #472

PROCEEDINGS—Trends & Applications 1983

May 25-26, 1983

Members—\$22.50 Nonmembers—\$30.00

Use order form on p. 89.