ACTOR AND CAMERA DATA TYPES IN COMPUTER ANIMATION

Daniel Thalmann
Dépt. D'I.R.O., Université de Montréal
Montréal, Canada

Nadia Magnenat-Thalmann
Ecole des Hautes Etudes Commerciales
Montréal, Canada

ABSTRACT

New abstract graphical data types for computer animation are presented: animated basic types, actor data types and camera data types. A new high level computer animation language has been based on these concepts. This language, called CINEMIRA, also includes message switching, script subprograms and scene control by a director. CINEMIRA is part of a complete 3D shaded animation system including a 3D digitizing program, the 3D HORIZON graphics editor and the MUltiple Track ANimator system (MUTAN), an interactive system for independently animating three-dimensional graphical objects.

KEYWORDS: 3D computer animation, abstract types, actor, camera.

RESUME

On présente ici de nouveaux types graphiques abstraits pour l'animation: les types de base animés, les types acteurs et les types caméras. Un nouveau langage évolué pour l'animation par ordinateur a été basé sur ces concepts. Ce langage, baptisé CINEMIRA, permet aussi les échanges de messages, la définition de scripts sous forme de sous-programmes et le contrôle de scènes par un réalisateur. CINEMIRA fait partie d'un système d'animation tridimensionnelle avec réalisme. Ce système est composé, entre autres, d'un programme de digitalisation à trois dimensions, d'un éditeur graphique à trois dimensions, HORIZON et d'un système interactif multi-pistes (MUTAN) permettant d'animer de manière indépendante des objets à trois dimensions.

MOTS-CLES: animation par ordinateur 3D, types abstraits, acteur, caméra.

1. INTRODUCTION

Recent developments in the design of programming languages have led to new concepts that are fundamental to the control of motion and temporal events. In particular, research in structured programming and data structures has allowed the design of high level languages such as PASCAL [1], and SIMULA-67 [2]. Work on data abstraction [3,4] is the basis of abstract data types existing in a number of languages such as ALPHARD [5], CLU [6] and ADA [7]. Abstract graphical data types [8,9] have also an important impact on computer graphics methodology. In the abstract graphical 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 the standard graphical types are obtained.

By introducing three-dimensional abstract graphical types into PASCAL anb by providing the means of defining any drawing with them, we obtain the powerful structured graphic language MIRA-3D [10]. MIRA-3D has been used for a certain number of applications, including the three-dimensional computer animation film DREAM FLIGHT [11,12].

For computer animation, other developments in the design of programming languages are fun-

damental. Concepts of synchronisation and message passing exist in SMALLTALK [13], CONCURRENT PASCAL [14], MODULA [15] and ADA. Research has also been done on actor systems like PLASMA [16, 17] and this has important implications for computer animation.

This paper presents new abstract graphical data types: animated basic types, actor data types and camera data types. A new high level computer animation language has been based on these concepts. This language, called CINEMIRA, also includes message switching, script subprograms and scene control by a director.

2. MIRA-3D AND THE FILM "DREAM FLIGHT"

Our film DREAM FLIGHT (Fig. 1-2) is a 3D computer animated film of fiction which was completely produced by computer. The film has been developed using the MIRA-3D programming language to create the objects or motions. MIRA-3D is a powerful structured graphic language. It includes:

- three-dimensional vector arithmetic
- graphical statements
- image transformations
- viewing transformations: perspectives and parallel projections
- standard procedures and functions.

The most important tool in this graphical extension is the 3D abstract graphical type: the figure type. The syntax is as follows:

<figure type>::= figure <formal parameter list>; <block>

The formal parameter list and the block are similar to the corresponding elements in a procedure. However, statements in the block must build the graphical objet either by line drawing specifications or surface specifications (for the removal of hidden surfaces and shading).

To define a figure type, the following steps must be carried out:

1) determine the characteristics of the figure, which then become the parameters.
2) develop an algorithm which allows the user to build the figure with the help of the parameters.

Graphical variables are defined as variables of graphical type. Four fundamental statements allow the user to manipulate these variables:

1) create <figure> (<actual parameter list>)
2) delete <figure>
3) draw <figure>
4) erase <figure>

The first operation creates the figure by giving values to the corresponding type parameters; the figure may then be drawn, erased or deleted.

Although MIRA-3D is powerful, it took 14 months to produce the 13 minute film DREAM FLIGHT. Abstract graphical types like figures are very useful, but they don't have their own animation. For this reason, we have designed animated basic data types, actor data types and camera data types.

3. ANIMATED BASIC DATA TYPES

Image transformations are defined by functions depending on variables of three basic types: INTEGER, REAL and VECTOR. For example, a translation is defined by a VECTOR translation, and a rotation by a VECTOR (the center) and a REAL number (the angle). Attributes like color, intensity or source lights are also defined by parameters of these types. Viewing transformations are also based on these types. For example, a perspective projection is defined by its center which is a vector. Animation of objects (actors) and cameras can then be based on the animation of basic parameters.

A good way of defining animation of these parameters is to introduce animated basic types. This concept is a generalization of the Newton concept defined in ASAS [18]. 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 which 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. For example, suppose we wish to define a vector that starts at time TSTART and moves with a constant speed SPD from the point PT and stops at time TEND. This is expressed follows:

```
type  TVEC=animated VECTOR (TSTART, TEND:
                            REAL; PT, SPD:
                            VECTOR);
        val PT..UNLIMITED;
        time TSTART..TEND;
        law PT + SPD * (CLOCK - TSTART)
        end;
```

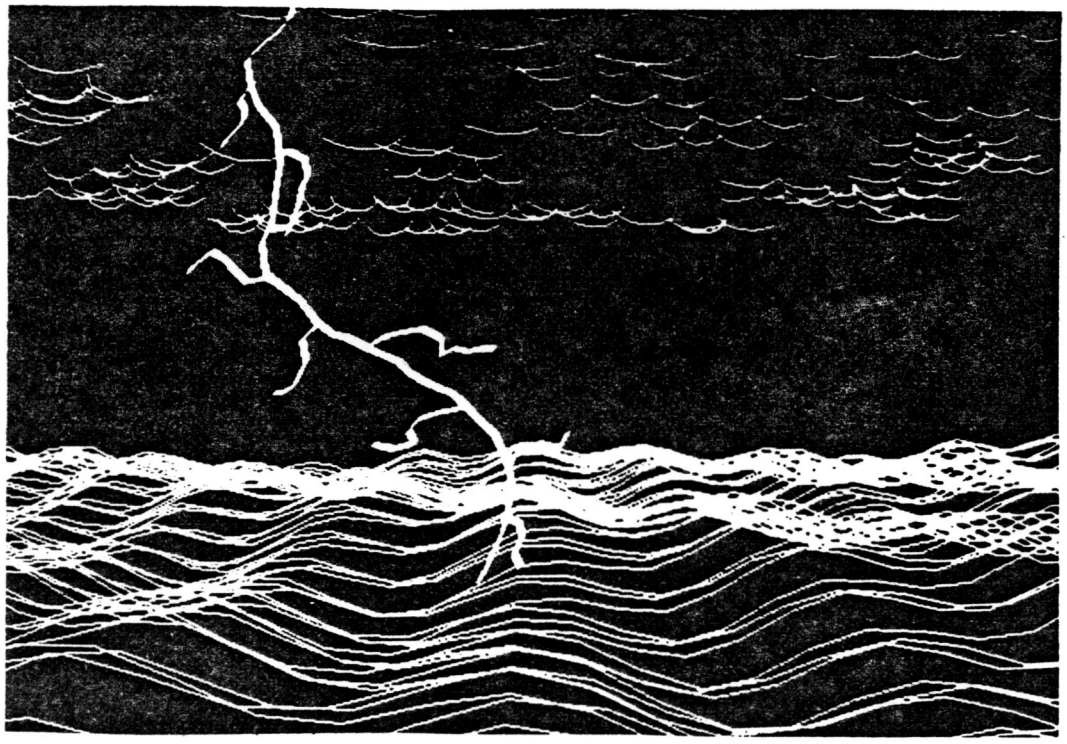

Figure 1: A frame of DREAM FLIGHT film

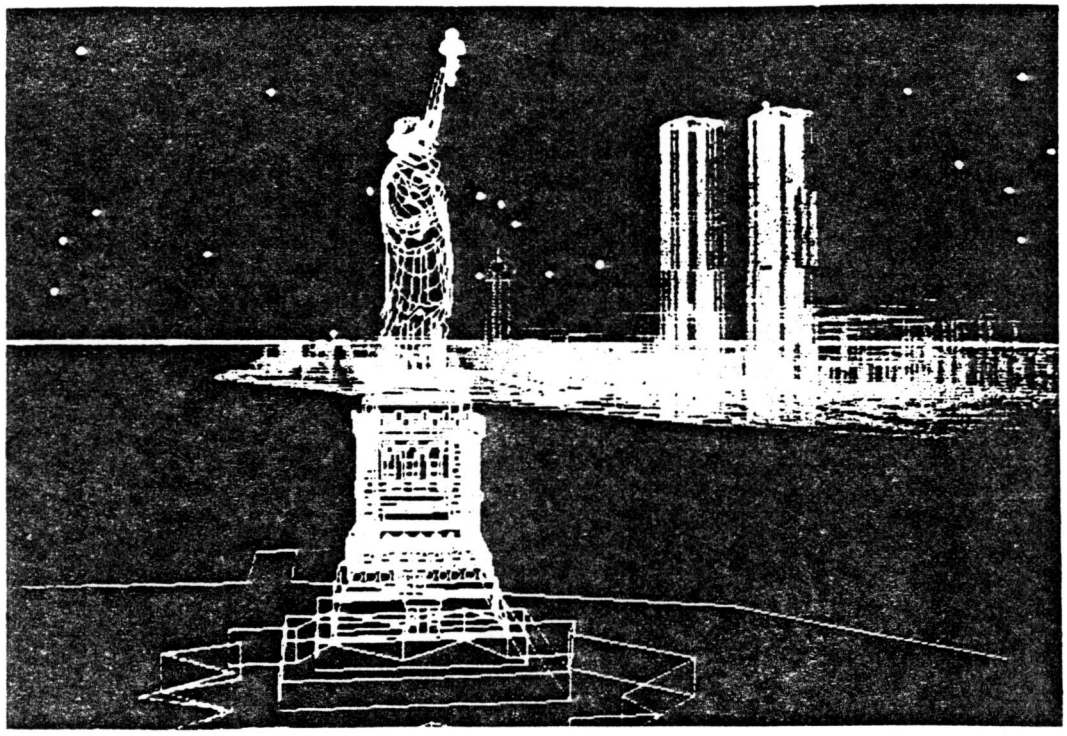

Figure 2: A frame of DREAM FLIGHT film

As the end position and the ending time are mutually redundant values, they are not both required. It is possible to use UNLIMITED to avoid specifying a value.

Expressions in the law can also involve "CLOCK" that is the current time and "CURVAL" that is the current value of the animated basic variable.

Initialization of the animated basic variables are performed by the init statement. It is at this stage that values of the parameters are given.

```
e.g.  init VEC (10, 13, <<0,10,4>>,
                <<3,0,0>>)
```

Animated basic variables can be used wherever a variable of the same basic type would be used. For example, if A and B are two vector variables, the following statement is possible:

```
A:=B cross VEC
```

4. ACTOR DATA TYPES

An actor type is an animated abstract graphical data type. The syntax is very similar to the figure syntax, except that the life-time limits of the actor have to be specified. Animated basic types and variables can be defined within an actor type.

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 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. If the actor type CAR is defined as:

```
type CAR = actor (TINIT: REAL);
           time TINIT..20;
           ...
```

actors of CAR type will be on the scene between the time TINIT and 20.

As in the case of animated variables, actor variables are initialized by the init statement. For example, two different variables of CAR type can be initialized:

```
var CAR1,CAR2: CAR;
   :
   :
init CAR1(10);
init CAR2(12);
```

It is also possible to use an actor type as another PASCAL type. For example we can define:

```
var CARS: array [1..20] of CAR
```

We now give an example of an actor type: we define an actor that is a stone (of icosahedric shape) that falls from a position P until it arrives at the level 0(Y=0), for example a water surface. The actor type STONE is define in Fig. 3.

In many cases, it is difficult to decide in advance when an actor must start moving. In our example, suppose that the stone has to begin moving when a person (another actor) drops it. This can be performed by removing the TSTART parameter in the STONE actor and replacing the lower time limit TSTART in the TVEC type by SIGNAL. The PERSON actor type must contain a start ROCK statement at the right time.

More generally, a start A statement has the immediate effect of replacing all lower SIGNAL time limits of the actor A by the current runtime. It can make the actor appear and/or start the motion of animated variables declared within the actor type. Similarly, a stop B statement has the immediate effect of replacing all higher SIGNAL time limits of the actor B by the current time. It can make the actor disappear and/or stop the motion of animated variables declared within the actor type.

5. CAMERA DATA TYPES

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. These can be those of the GSPC Core System [19] and their parameters can of course be animated variables. Fig. 4 shows an example of camera type.

Some viewing procedures other than the GSPC procedures have also been predefined:

```
PARACAMERA (EYE, INTEREST, ZOOM)
PERCAMERA (EYE, INTEREST, ZOOM)
```

```
type STONE=actor(P:VECTOR;TINIT,TSTART:REAL);
             time TINIT..UNLIMITED;
             type TVEC=animated VECTOR;
                    val P..<<P.X,0,P.Z>>;
                    time TSTART..UNLIMITED;
                    law P-0.5*9.81*SQR(CLOCK-TSTART)
                  end;
             var VEC:TVEC;
                 ICOSA:ICOSAHEDRON;
           begin
             init VEC;
             create ICOSA (VEC,CFA,DIR);
             include ICOSA
           end;
```

Figure 3: The STONE actor type

```
type TCAM=camera(TINIT,TEND:REAL);
            time TINIT..TEND;
            type TVEC=animated VECTOR(VSTART,VSTEP:VECTOR);
                   val VSTART..UNLIMITED;
                   time TINIT..TEND;
                   law CURVAL+VSTEP
                 end;
            var V1,V2:TVEC;
          begin
            init V1 (<<0,0,1>>,<<0,0.1,0>>);
            init V2 (<<20,25,-20>>,<<0,1,-0.5>>);
            VIEWPORT(<<0,0>>,<<1,1>>);
            WINDOW(<<-20,-5>>,<<20,35>>);
            PLANENORMAL(V1);
            PERSPECTIV(V2)
          end;
```

Figure 4: A camera type

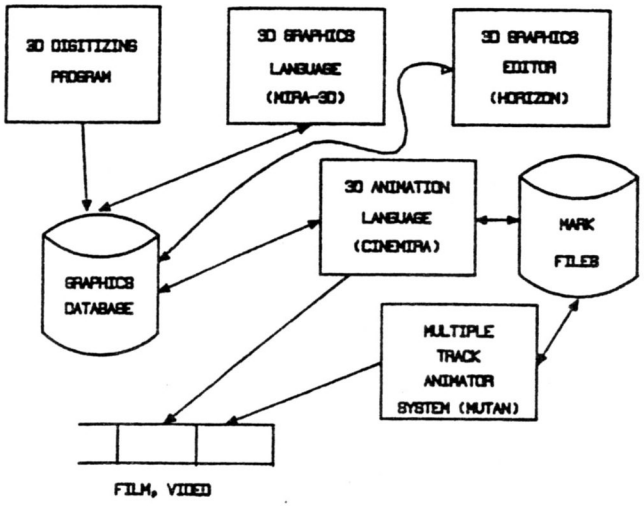

Figure 5: A 3D animation system

These two procedures allow the user to easily define visual parameters for parallel and perspective projections by giving the position of the eye, an interest point and a zoom value. Rectangular and polygonal clippings can also be specified in a camera type. It is also possible to clear or color a rectangular or polygonal area. These capabilities unable special effects to be created similar to those created with optical printers. When several cameras are running at the same time, cross-dissolves effects are easily produced.

Camera types can be used just like other PASCAL types.

e.g. var CAM: array [1..3] of TCAM;

The start and stop statements can also be applied to camera variables.

6. CINEMIRA SCRIPTS

Animated basic types, actor data types and camera data types have been introduced in the CINEMIRA language. This language is a high level computer animation language based on script subprograms.

A CINEMIRA script is a subprogram dedicated to computer animation. A program can invoke several scripts, but only one at the same time. A script is under the control of a director which is normally not apparent, but can communicate with other entities of the system by messages. A script has a name that is an identifier and can have parameters like a procedure. A script block is composed of declarations (constants, types, variables and subprograms), statements and scenes. The declaration part can include all declarations allowed in MIRA-3D, animated basic types and figure types. Actor types and camera types as described in previous sections of this paper must be defined in scripts. A script is a sequence of scenes. The scenes can be preceeded by a sequence of statements to initialize objects that are common to several scenes. The scene has a name and consists 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.

```
e.g.  create  TREE(...);
      create  SNOW(...);
      create  SKY;
      decor   TREE,SNOW,SKY
```

A shoot statement performs the shooting phase. Decor, actors and cameras are automatically placed during the shooting 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). It is also possible to include specific statements in the shoot statement by adding a do clause like:

shoot until< expression> do< statement>

This feature can be useful for some special effects, but it is usually to be avoided because statements cannot manipulate actors and cameras, but only figures.

A script can contain more than one scene. Each scene can have initializations and must have a shoot statement. Actors, cameras, decor and animated basic variables can be activated for several scenes or parts of scenes. In this case, they have to be initialized before the first scene and not within a scene.

Synchronization of actors and cameras can sometimes be completely forecasted. This means that synchronization mechanisms can be programmed using only init, start and stop statements - However, this kind of synchronization is insufficient, because of the lack of dialog between actors and cameras. On of the best ways of implementing this kind of synchronization is to use the concept of message switching already existing in PLASMA and ASAS. CINEMIRA allows actors and cameras to switch messages by using the two statements send and receive. A message can be also sent (or received) within the body of a scene. In this case, the sender (or the receiver) is necessarily the director. The director can take any message even if he was not the presumed receiver.

7. A 3D SHADED ANIMATION SYSTEM

CINEMIRA is part of a general computer animation system as shown in Fig. 5. Basic figures can be created by using the HORIZON [20] graphics editor or by using the MIRA-3D programming language. They can also be created by using a 3D digitizing program. Fig. 6 shows an example of car before it is shaded. MUTAN [21] is a MUltiple Track ANimator system for motion synchronization. It is an interactive system for independently animating three-dimensional graph-

ical objects. It is also a good tool for synchronizing motion with sound, music, light or smell. To make this possible, MUTAN handles several tracks at a time. All animation constraints for a graphical object are recorded on each track. A program in CINEMIRA will be able to read and write MUTAN tracks.

ACKNOWLEDGEMENTS

This work is sponsored by the Natural Sciences and Engineering Research Council, the Government of Quebec (FCAC) and the Business School of Montreal.

REFERENCES

[1] Jensen, K. and Wirth, N. "PASCAL-User Manual and Report", Springer-Verlag, 1974.

[2] Dahl, O., Myrhang and Nygoard, "The SIMULA 67 Common Base Language", Norwegian Computing Centre, Oslo, 1968.

[3] Liskov, B. and Zilles, S. "Programming with Abstract Data Types, Proc. SIGPLAN Symposium on Very High Level Languages, March 1974.

[4] Guttag, J. "Abstract Data Types and the Development of Data Structures", Comm. ACM, Vol. 20, No 6, 1977.

[5] Wulf, W.A.; London, R. and Shaw, M. "Abstraction and Verification in ALPHARD", Carnegie-Mellon Univ., Dept. Comp.Sc., 1976

[6] Liskov, B.; Snyder, A; Arninson, R. and Schaffert, C. "Abstraction mechanism in CLU", Comm. ACM, Vol. 8, 1977, pp. 565-576.

[7] U.S. Dept. Defense, "Reference Manual for the Ada Programming Language", 1980, Gov. Printing Office Order L008-000-00354-8.

[8] Thalmann, D. and Magnenat-Thalmann, N. "Design and Implementation of Abstract Graphical Data Types", Proc. COMPSAC'79, IEEE Press, pp. 519-524.

[9] Magnenat-Thalmann, N. and Thalmann, D. "The Use of 3D Abstract Graphical Types in Computer Graphics and Animation", Proc. INTERGRAPHICS'83, Tokyo.

[10] Magnenat-Thalmann, N. and Thalmann, D. "MIRA-3D: A Three-Dimensional Graphical Extension of PASCAL", Software-Practice and Experience, 1983.

[11] Thalmann, D.; Magnenat-Thalmann, N. and Bergeron, P. "Dream Flight: A Fictional Film Produced by 3D Computer Animation", Proc. Computer Graphics'82, London, Online Conf., 1982, pp.352-367.

[12] Magnenat-Thalmann, N.; Bergeron, P. and Thalmann, D. "Above Sea and Undersea Computer Animation Scenes", Proc. 1983 Intern. Computer Color Graphics Conf., Talahassee Florida, 1983.

[13] Goldberg, A. and Kay. A. "SMALLTALK-72 Instruction Manual Learning Research Group, XEROX, Palo Alto, 1976.

[14] Brinch Hansen, P. "The Programming Language Concurrent PASCAL", IEEE Trans. on Software Eng., vol.1, no2, 1975, pp.199-207.

[15] Wirth, N. "MODULA: A Language for Modular Multiprogramming", Software-Practice and Experience, 7, 1, 1977, pp.3-35.

[16] Greif, I. and Hewitt C. "Actor Semantics of PLANNER-73", Proc. ACM SIGPLAN-SIGACT Conf. PALO ALTO, 1975.

[17] Hewitt C. and Atkinson, R. "Parallelism and Synchronization in Actor System", ACM Symposium on Principles of Programming Languages, pp.267-280.

[18] Reynolds, C.W. "Computer Animation with Scripts and Actors", Proc. SIGGRAPH'82, 289-296.

[19] Status Report of the Graphic Standards Planning Committee of ACM/SIGGRAPH, Computer Graphics, vol. 19, no 3, 1979.

[20] Magnenat-Thalmann, N.; Larouche, A. and Thalmann, D. "An Interactive and User-Oriented Three-dimensional Graphics Editor", Proc. Graphics Interface'83, Edmonton.

[21] Fortin, D.; Lamy, J.F. and Thalmann, D. "A Multiple Track Animator System for Motion Synchronisation", Proc. SIGGRAPH-SIGACT Workshop on Motion, ACM, Toronto, 1983.

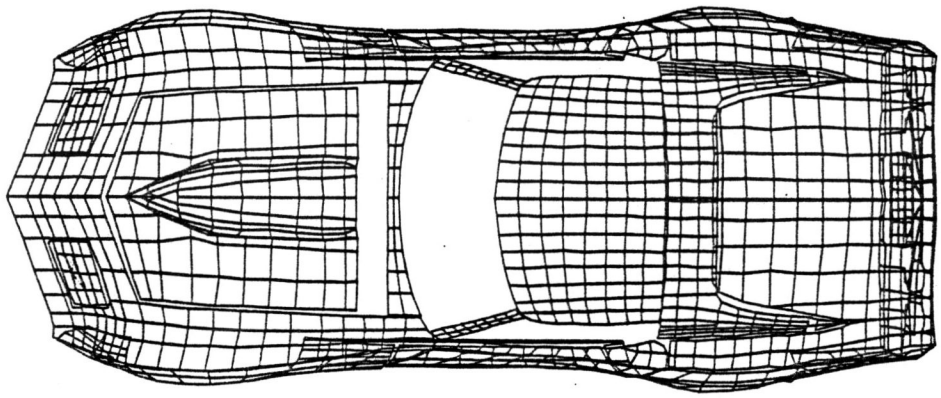

Figure 6: A CORVETTE (digitized by Nicolas Chourot)