DIRECTOR-ORIENTED 3D SHADED COMPUTER ANIMATION

Nadia Magnenat-Thalmann
Hautes Etudes Commerciales
Université de Montréal
Canada

Daniel Thalmann
Département d'Informatique
et de Recherche Opérationnelle
Université de Montréal
Canada

(Invited paper)

ABSTRACT

Artists who do not know how to program a computer have to use an artist-oriented system offering the possibility of making most scenes of a computer-animated film. This system must provide facilities for controlling main actor motions, virtual camera handling and lighting.

In the case of very complex motions, procedural models can be programmed by a specialist but it is essential that the control of these procedural models be assured by the animator in the interactive system. This means that the new motion has been added to the system; in fact, the system is extensible.

We have designed such a system, called MIRANIM, composed of an animator-oriented system called CINEDIT and an animation sublanguage called CINEMIRA-2.

KEYWORDS : Animation, actors, cameras, lights, director.

INTRODUCTION

Most authors (1,2,3) distinguish two types of computer animation: computer-aided animation and modelled animation.

The first type of animation is used in the cartoons of Walt Disney or Hannah-Barbera. In this type of two-dimensional character animation, the computer may have a role to play in three main areas: the creation of drawings, the production of inbetweens, and painting. In each of those areas, computer-assisted animation and interactive user-oriented systems are normally used: e.g. graphics editors, key-frame systems and paint systems.

The second type of animation corresponds to animation sequences in which three-dimensional models move about in three-dimensional space. The process is very complex without a computer. Modelled animation has been used in television advertisements (4) and for special effects in films like "Star Trek II", "Return of Jedi" and especially in "TRON". 3D computer animation involves three main activities:

1) object modelling
2) motion specification and synchronization
3) image rendering.

Although there are problems in computer-aided animation (1) , various animator-oriented systems have been developed (5,6,7) and they work. They can be used by non computer-scientist artists. In modelled animation, the problem is more complex and always centers around the question: programming language or animator-oriented system ? The production of three-dimensional computer-animated films using a graphical programming language is time-consuming. For example, it took 14 months to produce the 13 minutes film DREAM FLIGHT (8) although we have used structured programming with the MIRA-3D language (9) based on high-level graphical types (10).

Moreover, such an approach implies that animators also have to be programmers. User-friendly interactive systems have the great advantage of being dedicated to artists but they impose limits on the creativity of artists who would like to exploit all the possibilities of a computer. Special effects like the ones shown in Fig. 1 and 2 are difficult to produce without programming. Apart from 3D Key-frame Animation Systems like BBOOP (11) or MUTAN (12), there are not many examples of artist-oriented 3D systems except ANTS (13,14). Part of the spectacular effects in TRON, for instance, have been produced by Information International Inc. using ASAS (15) which is a typical programming language used for computer animation.

Our approach is the following one: artists who do not know how to program a computer (they are the majority) have to use an artist-oriented system. This system must offer the possibility of making most scenes of a computer-animated film. It must provide facilities for controlling main actor motions, virtual camera handling, lighting. In the case of a very complex motion, procedural models can be programmed by a programmer but it is essential that the control of this procedural model be assured by the animator in the interactive system. This means that the new motion has been added to the system that is then an extensible system. We have designed such a system (called MIRANIM) composed

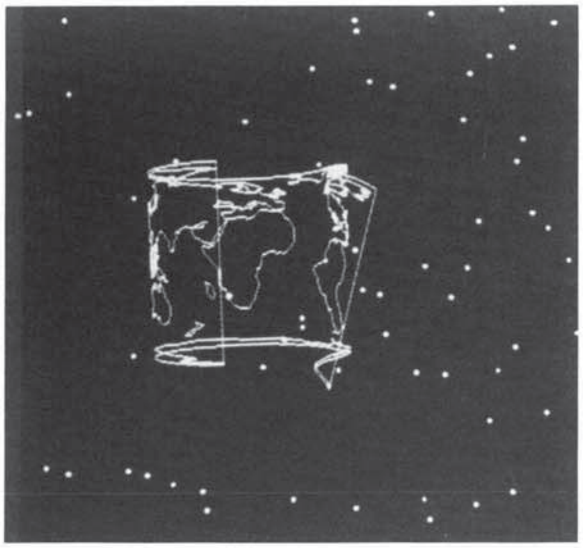

Fig.1 Formation of the earth (from DREAM FLIGHT)

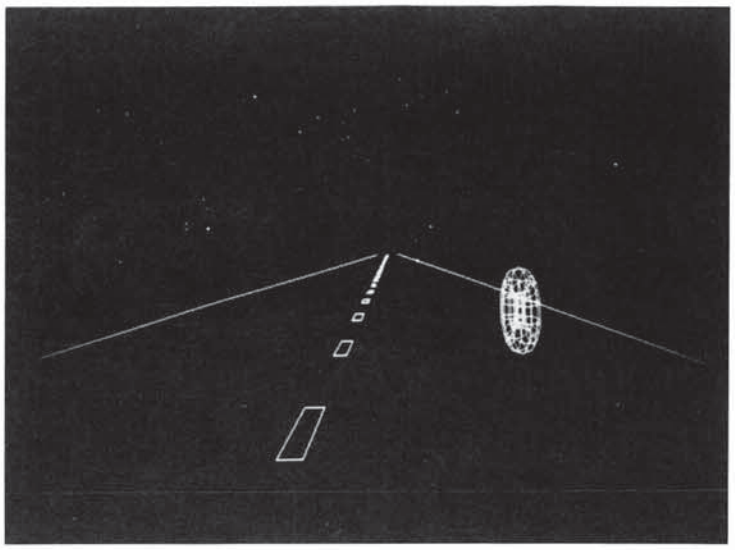

Fig.2 Motion of a wheel (from NIRVANA)

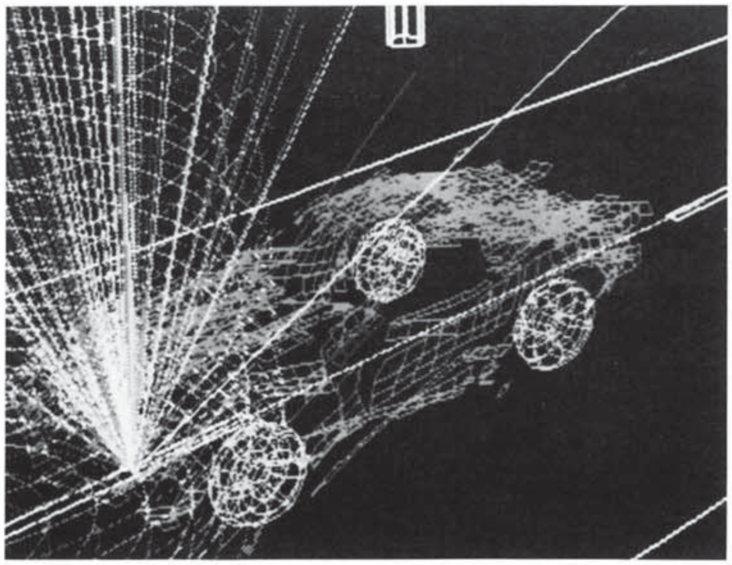

Fig.3 An explosion (from NIRVANA)

of an animator-oriented system called CINEDIT and an animation sublanguage called CINEMIRA-2.

DESCRIPTION OF THE ANIMATOR-ORIENTED SYSTEM CINEDIT

This system allows the animator to specify a complete script without any programming. The animator can create actors with their motions and transformations as well as virtual cameras with their motions and characteristics. Decors can also be built interactively. Multiple light sources can be defined which can be moved around at will. Eight modes are defined in the system and there are specific commands for each mode:

1) Variable mode

This mode allows the animator to create constants and animated variables. Such variables are defined by an evolution law describing how their values are changing during a period of time. Among the available laws, there are Catmull laws and the main physical motions. These animated variables play a considerable role, because they drive the motion of actors, cameras and lights. For example, to define a point PT that turns from 0 to time 10 with a circular motion around an axis passing through the vector <8,4,0> and plane normal vector <0,0,1>, with an angular velocity of 1 rd/sec and angular acceleration of 0 rd/sec^2 we type:

VEC, PT, A, 0, 0, 0
- defines point PT as a vector with an initial value.

LAW, MYLAW, MVTCIRC, 8,4,0,0,0,1,1,0
- defines the law MYLAW as a specific circular motion.

EVOLUTION, PT, MYLAW, 0,10
- associates the law to the point from time 0 to time 10.

2) Object mode

This mode offers the same possibilities to create objects as a limited 3D graphics editor does. In fact, basic objects must be built outside the animation system. But they can be modified in this object mode by rotations, translations, scaling and coloring.

3) Decor mode

This mode allows the animator to build a decor with objects and to display it in order to get an idea of its aspect.

4) Actor mode

This mode is the most important one. The animator defines actors (animated objects) and then gives a list of transformations which have to be applied to each actor. At present, 16 kinds of transformations are available including rotation, sizing, translation, shears, torsion, traction, flexion, stochastic transformation, color changing. The parameters of a transformation may be animated variables; the latter are also generally used to specify the time dependence. For example, in a rotation, the angle can be an animated real number and the direction of the axis can be an animated vector.

The number of transformations associated to an actor are not limited and the transformations are driven by the animated variables. For example, suppose we would like to perform the following transformations on a tree:

1) changing the size
2) having a flexion
3) changing the color from green to red.

This can be performed by the following commands:

ACTOR, TREE, TREEOBJ
- defines the actor tree with the object TREEOBJ as basis.

SIZE, TREE, V
- V is an animated vector which defines how the size is changing.

FLEXION, TREE, V1,V2,V3,V4
- V1,V2,V3 and V4 are flexion parameters.

COLOR, TREE, VC
- VC is an animated vector which changes from the HLS value of green to the HLS value of red according to a law.

5) Camera mode

In this mode, the animator can define one or several virtual cameras. Each camera has an eye point and an interest point which can be animated vectors. Moreover, clipping, spin, viewport and zoom can be specified for a camera as well as being animated. The eye point or the interest point of a camera can also follow the motion of a specific actor. By using several cameras at the same time, special effects like wipes can easily be achieved.

6) Light mode

In this mode, the animator can define one or several light sources and their motion(s).

7) Animation mode

This mode is the director's mode; starting time and duration of actors, cameras and decor are decided this way. It is also in this mode that shooting or playback is activated.

8) Control mode

This mode allows the animator to enter in other modes, to save or retrieve actors, cameras, decors or to obtain a list of the script under the form of an "abstract table" of all variables, laws, actors, transformations, cameras, lights and so on.

THE CINEMIRA-2 SUBLANGUAGE

Based on our experience with the CINEMIRA language in actor and camera data types (16), we have designed a sublanguage called CINEMIRA-2, less complex than CINEMIRA. This sublanguage is limited to the programming of entities to be used by CINEDIT; it is not possible to write a program in CINEMIRA-2. What is innovative with this approach is that an entity programmed in CINEMIRA-2 is directly accessible in CINEDIT. For example, an animator would like to introduce a transformation EXPLOSION in a scene where an actor CAR is running, has a crash and is destroyed by an explosion. The animator would like to control the EXPLOSION by parameters like the speed of the car V and the distance to an obstacle D. We assume that the animator does not know how to program. He asks a programmer to implement a transformation EXPLOSION (V,D) who does so. Then, the animator can define in the actor mode:

```
EXPLOSION, V, D
```

where V is an animated vector and D an animated real. Fig. 3 shows an explosion.

Apart from actor transformations, CINEMIRA-2 allows the programming of six kinds of entities:

1) animation blocks
2) laws
3) transformations
4) objects
5) subactors
6) cameras.

Commands in CINEDIT allow the animator to have access to these entities. These commands have parameters corresponding to the parameters of the entities defined in CINEMIRA-2. Types of parameters can be one of the following:

1) VECTOR
2) REAL
3) OBJ for graphical objects
4) ACT for actors
5) CAM for cameras.

An animation block is a subprogram executed at each frame.

```
e.g.  block BALL (CENTER: VECTOR);
      var SPH: SPHERE;
      begin
         create SPH (CENTER,3);
      end;
```

The animator can activate a block in the editor by calling the command BLOCK, and then choose the parameters.

```
e.g.  BLOCK,BALL,10,5,CENTER
```

- where 10 is the activation time, 5 the duration time and CENTER an animated vector.

Laws can be defined in a similar way to functions in PASCAL. However, the type of the function is necessarily REAL or VECTOR and these laws are functions of the time (CLOCK).

```
e.g.  law    MVTACC (SPDINIT, ACC: VECTOR):
                                   VECTOR;
      begin
         MVTACC := 0.5 * ACC * SQR (CLOCK) -
                                SPDINIT * CLOCK
      end
```

These laws can then be used by the animator with the command LAW of the editor.

Procedural objects can be defined under the form of graphical types as already shown in (9,10), and can be modelled either as line-drawing objects or as 3D shaded objects. Fig. 4 and 5 show examples.

In the editor, the animator can create procedural objects and of course he can choose any parameter he wants by the command PROBJECT.

Subactors can either be used as parts of an actor in the editor or as a complete actor. Synchronization between actors and subactors is assured by parameters. These parameters can be animated variables. For example, we can define, in the editor, an actor CAR with a velocity V. The car possesses 4 wheels which are subactors. These wheels have a rotation speed depending on the speed of the car. The wheels are implemented in CINEMIRA-2 under the form:

```
type  WHEEL = subactor (CENTER,V : VECTOR)
              begin
                 .
                 .
                 .
              end;
```

In the editor, the CAR with animated wheels is created as:

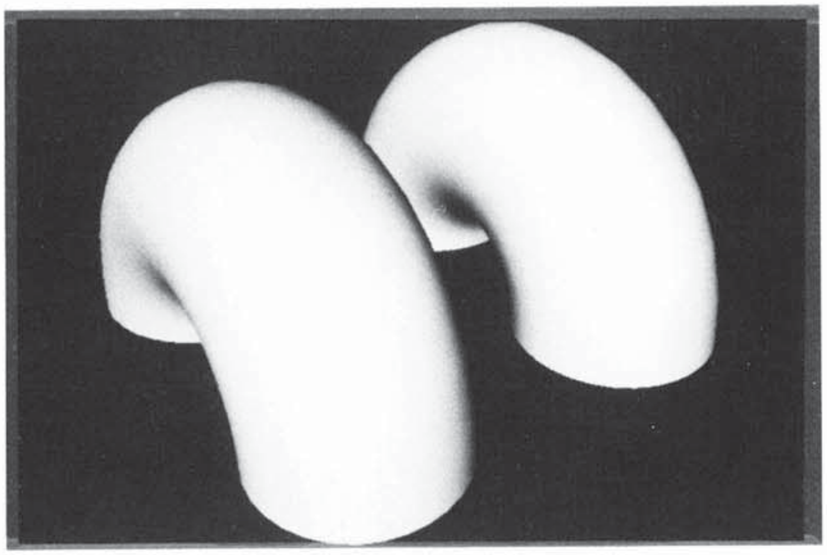

Fig.4 3D shaded objects implemented by graphical types

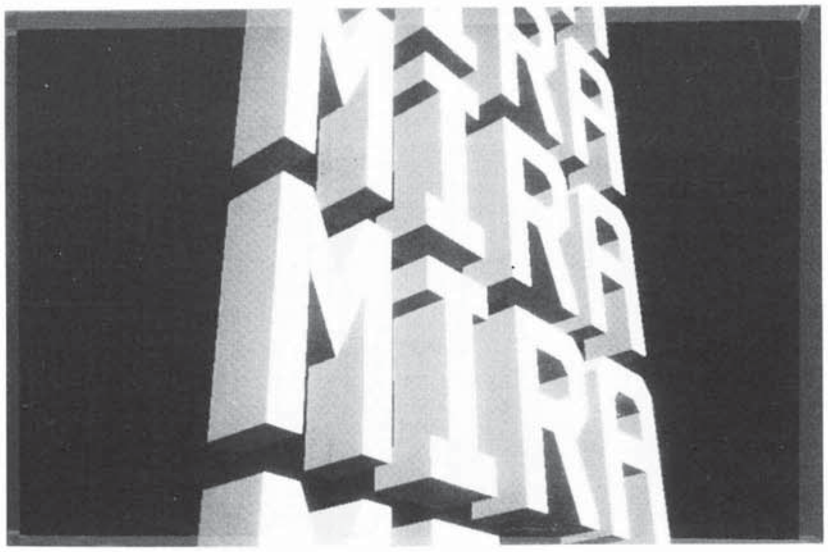

Fig.5 3D shaded text implemented by graphical types

```
ACTOR, CAR, CARFILE
TRANSLATION, CAR, A, POSITION
SUBACTOR, WHEEL1, WHEEL, POINT1, SPEED
SUBACTOR, WHEEL2, WHEEL, POINT2, SPEED
SUBACTOR, WHEEL3, WHEEL, POINT3, SPEED
SUBACTOR, WHEEL4, WHEEL, POINT4, SPEED
HIERARCHY, CAR, WHEEL1, WHEEL2,
                        WHEEL3, WHEEL4
```

Cameras are generally defined in the editor with numerous facilities to produce special effects like these provided by optical printers. However, it is also possible for an animator to define procedural cameras.

In this case, he defines cameras as entities of a camera type.

```
e.g.    type  TCAM = camera (P : VECTOR);
                     begin
                        if  NORME(P) < LIMIT
                          then PERCAMERA
                                 (P,INT,ZOOM,SPIN)
                          else  PARCAMERA(F(P),
                                 INT,ZOOM,SPIN)
                     end;
```

A camera of the TCAM type can be activated in the editor by the command

PROCAM, CAM1, TCAM, POINT
- POINT is an animated vector.

IMPLEMENTATION

CINEDIT is a 12000 source lines program written in MIRA-SHADING (17), that is a structured language providing the programmer the means to specify, manipulate and animate 3D shaded objects by using high-level graphical types.

Like MIRA-SHADING, CINEMIRA-2 has been implemented by the development of a preprocessor written in ISO PASCAL. This preprocessor produces standard PASCAL modules and a specification file. To extend CINEDIT with new entities, programmed in CINEMIRA-2, the following steps have to be taken:

1) Precompile new entities.
2) Compile PASCAL output.
3) Link output with CINEDIT relocatable code.

At runtime, access to the new entities by CINEDIT is performed by checking the specification file produced by the CINEMIRA-2 preprocessor. Fig. 6 shows the complete organization of this process.

CINEDIT and CINEMIRA-2 have been implemented on VAX 11/780 for various terminals. Shaded images are produced with an AED 767 terminal.

CONCLUSION

CINEDIT is a three-dimensional animation system which requires no knowledge in Computer science of its user. This does not mean that any person can sit down to the terminal and produce a complex computer-animated film in a short time. CINEDIT has been designed for animators and directors who are accustomed to design, motion and colors. The animator always has to prepare his/her script before using CINEDIT. However, when such a script is ready, when the role of all variables, actors, cameras, lights and decors has been defined, it is a straightforward task to use CINEDIT.

CINEDIT consists of a powerful set of commands and numerous predefined motions, laws and transformations. However, any director-oriented system has its limitations. With our approach, the artist can ask a programmer to write any procedural entity he/she needs and this entity can be controlled by the animator as a predefined entity. Consequently, our system offers both advantages of being user-oriented and extensible.

ACKNOWLEDGEMENTS

We are grateful to Serge Lafrance who has worked on the CINEDIT implementation, to Mario Fortin who has developed the CINEMIRA-2 preprocessor and to Louis Langlois who has worked on the realistic effects. The project has been supported by NSERC and FCAC.

REFERENCES

(1) Catmull, E. "The Problems of Computer-assisted Animation", Computer Graphics, Vol. 12, No. 3, 1978, pp. 348-353.

(2) Magnenat-Thalmann, N. and Thalmann, D., "Principles of Computer Animation", Springer-Verlag Tokyo Inc., 1984.

(3) Lansdown, R.J. "Computer Aided Animation: A Concise Review", Proc. Computer Graphics'82, Online Conf. Ltd, London, 1982, pp. 279-290.

(4) Crow, F.C. "Shaded Computer Graphics in the Entertainment Industry", Computer, IEEE Press, Vol.11, No. 3, 1978, pp. 11-22.

(5) Burtnyk, N. and Wein, M. "Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation", Comm. ACM, Vol. 19, No. 10, 1976, pp. 564-569.

(6) Stern, G. "Softcel: An Application of Raster Scan Graphics to Conventional Cel Animation", Proc. SIGGRAPH'79, pp. 284-288.

(7) Catmull, E. "New Frontiers in Computer Animation", American Cinematographer, October 1979.

(8) Thalmann, D.; Magnenat-Thalmann, N. and Bergeron, P. "Dream Flight: A Fictional Film Produced by 3D Computer Animation", Proc. Computer Graphics'82, Online Conf. Ltd, 1982, pp. 353-368.

(9) Magnenat-Thalmann, N. and Thalmann, D. "MIRA-3D: A Three-dimensional Graphical Extension of PASCAL", Software-Practice and Experience, John Wiley, Vol. 13, 1983, pp. 797-808.

(10) Magnenat-Thalmann, N. and Thalmann, D. "The Use of 3D High-level Graphical Types in the MIRA Animation System", IEEE Computer Graphics and Applications, Vol. 3, No. 9, pp. 9-16.

(11) Stern, G. "Bboop: A System for 3D Key Frame Figure Animation", SIGGRAPH'83 tutorial, pp. 240-243.

(12) Fortin, D.; Lamy, J.F. and Thalmann, D. "A Multiple Track Animator System", Proc. SIGGRAPH/SIGART Interdisciplinary Workshop on Motion: Representation and Perception, Toronto, 1983, pp. 180-186.

(13) Csuri C. et al. "Towards an Interactive High Visual Complexity Animation System", Proc. SIGGRAPH'79, pp. 289-299.

(14) Hackathorn R. et al. "An Interactive Microcomputer Based 3D Animation System", Proc. Canadian Man-Computer Communications Society Conf., 1981, pp. 181-191.

(15) Reynolds C.W. "Computer Animation with Scripts and Actors", Proc. SIGGRAPH'82, pp. 289-296.

(16) Thalmann, D. and Magnenat-Thalmann, N. "Actor and Camera Data Types in Computer Animation", Proc. Graphics Interface'83, pp. 203-210 (expanded version to appear in ACM Transactions on Graphics).

(17) Magnenat-Thalmann, N.; Thalmann, D.; Fortin, M. and Langlois, L. "MIRA-SHADING: A Language for the Synthesis and the Animation of Realistic Images", Technical Report, H.E.C., Montreal, 1983.

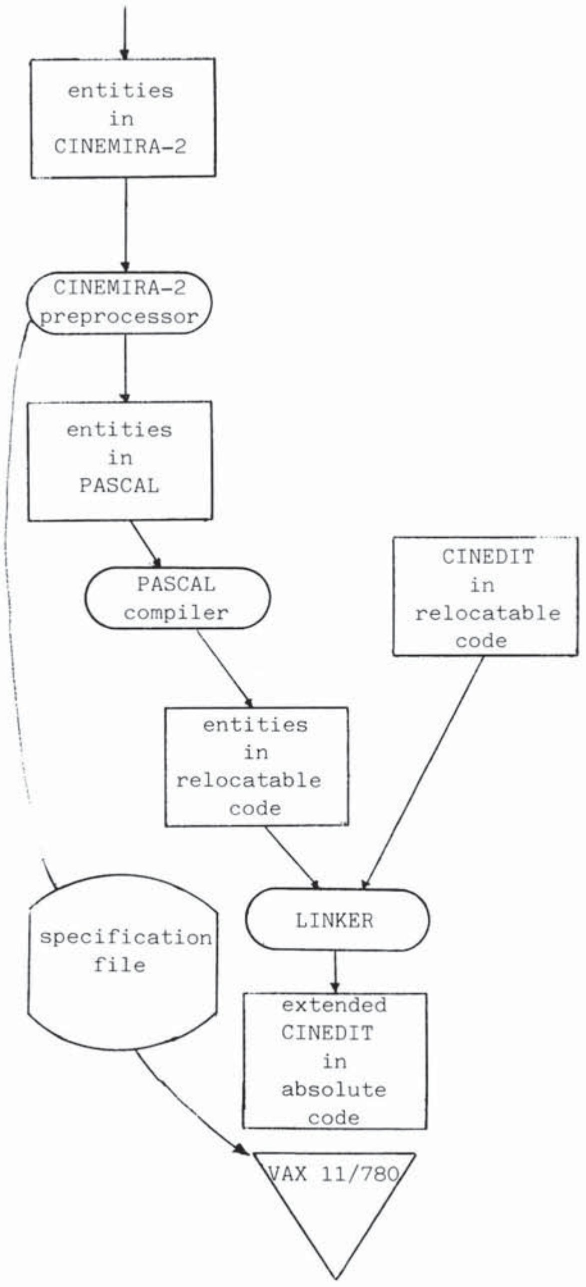

Fig. 6 : Organization of the extension process in MIRANIM