A STRUCTURED MOTION SPECIFICATION IN 3D COMPUTER ANIMATION

Philippe Bergeron

Département d'Informatique et de
Recherche Opérationnelle
Université de Montréal
Montréal, Québec

Invited paper

ABSTRACT

The main objective of this paper is to set out a structured approach to define the concepts, terminology and notations involved in the description of choreography in a computer animation sequence. Here the meaning of the word *choreography* should not be taken as in dance terminology, since we are using the word as a way to describe ongoing dynamic processes in a scene. This approach has been designed both for script based and interactive computer animation systems. A new way to describe the analogy with the real world of **director** and **actors** is presented. Formal notations of the same concepts are described. The relationship between the **director** and the **actors** is accomplished through the use of **global** and **local lists of events**. A structured example as part of a theoretical scripted system is described in PASCAL. The techniques used to describe choreography are completely independent from the display software. This is why we have emphasized our work in motion specification rather than in the quality of the final image.

KEYWORDS: computer animation, motion specification, choreography.

INTRODUCTION

Over the past few years, computer animation has become one of the most important branches of computer graphics. This technique can provide incredible, dazzling effects for the advertising or entertainment industry. However, even if there is a number of animation languages or interactive systems available on the market, there is no accepted standard method used to describe choreography. Here motion choreography has a more general meaning than in dance terminology; it can be defined as a way of describing ongoing dynamic processes in a scene. It is not restricted to human movements: motion specification of a cube, a bird or a flying saucer can be part of the same choreography. Terminology and notations involved in the description of computer animation choreography are presented in this paper. The author has designed this proposal during the making of *Dream Flight* [1,2,3], a story-telling short-subject completely done in 3D computer animation. The proposal was developed both for script based (e.g. language-driven systems) and interactive 3D computer animation systems. It is independent from the base language of a scripted system. Also, the techniques used to describe choreography are completely independent from the display software. This is why we have emphasized our work in motion specification rather than in the quality of the final image. An analogy with the real world of motion pictures is described. The concepts of **director**, and **actors** are presented in computational notations. The relation between the **director** and the **actors** is accomplished through the use of **global** and **local lists of events**. The **director** keeps the **global list of events** for himself, while each **actor** has his own **local list of events**.

THE DIRECTOR

In the real world, the **director** may direct any number of **actors** in the performance, of course with sometimes more than one at the same moment. Conceptually speaking, the **director** keeps a list of all the **actors** involved in the performance. This list is called the **global list of events** (the GLE). For each **actor**, the GLE provides the time of his start (**actor's birth**) and the length of his play (**actor's life**). Then, adding the **actor's life** to his **birth** provides the time of his **death**. Fig. 1 shows an example of a GLE.

It is evident that in real life there is no such formal list kept by the director. However, since accuracy is necessary in computer animation, the GLE is perfect to create the analogy between real and computer-generated environment. The same GLE is presented in a schematic way in fig. 2.

Each time a new frame is generated, the **director** activates all the **technicians** that are involved in the play (**light technician**, **cameraman**, and so on. The notion of activating a **technician** means setting out parameters depending on the frame number, such as the light and eye positions, that will later be used by the display software. It will not be discussed further in this paper). After, the **director** scans the GLE. If an **actor** is alive at the current moment, the **director** activates his **body**. This is the only message sent to **actors** besides the end. In fact, in real world, on the day of the performance, the **director** sends only two messages (or cues) to each performer: the starting and the stopping cues. The rest of the time, each **actor** is on his own. He knows exactly what he has to do and when to do it, without any intervention from the **director**. The concept of **actor** will be explained in the next section. In real life, activating the **actor's body** simply means that the **body** is on stage and (partially or not) visible to the audience. In the computational process, it means that the **body** is catalogued in a specific file that will later be treated by the display software once the GLE has been completely scanned. Let us give a formal interpretation of a **director** using a loop (called the **director**'s loop):

while <performance not over> **do**
begin
1° Activate the **technicians**
2° Scan the GLE, and activate the **actors** that are alive at this moment
3° Show each body of the active **actors** (* display software *)
4° Interface with the output (physical camera, film recorder)
5° Update the moment value
end

THE ACTORS

In real life, once the **director** has sent the starting cue, the **actor** knows exactly what he has to do without any intervention. He then activates his own chronometer (at least conceptually), so that each of his future moves will depend on the time he started, and not on that of the whole performance. An **actor** is divided into two parts: his **mind** and his **body**. Once he is on his own, the **actor's mind** dictates the moves his **body** must do, and the audience will only see the result from that process: the new position of the **body**. The situation is similar with computer animation.

The concept of computerized **actor** was first introduced by Carl Hewitt [4,5], and used in the description of the ASAS language developed by Craig Reynolds at the Massachusetts Institute of Technology [6,7]. This concept suggests an analogy with a theatrical performer. Let us present more formally this concept in computational notations, keeping the real world analogy. The style of notation in this section was inspired by the paper of Mudur and Singh [8]. Here, the notations are extended to the **actor**'s concept. An **actor** is

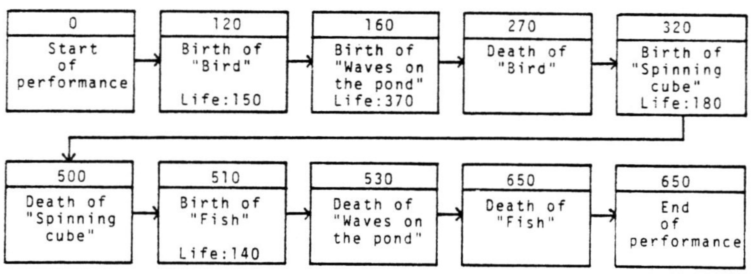
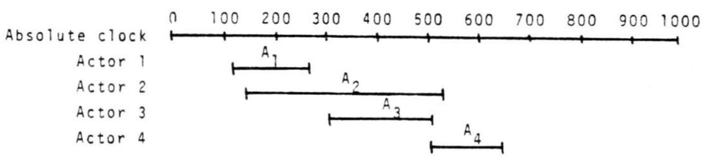


Fig. 1



where A_1 : Bird
A_2 : Waves on the pond
A_3 : Spinning cube
A_4 : Fish

Fig. 2

an independent element (it could be a procedure in a scripted system) responsible for a set of 3D models in a sequence. These 3D models may suggest the **actor's body**.

Notation: The **actor's body** B is a set of 3D models $M_1, \ldots, M_c$ logically dependent but not necessarily physically dependent. Most of the times in this paper, we will refer to B as being only one M. In a computer-generated environment, B can be any dynamic 3D model, and is not restricted to models simulating human beings.

The **actor's body** transformation from frame to frame is called a **B-transformation** (B for **body**). A **B-transformation** is expressed as a set of mathematical functions. These will be referred to as the **alter-body** functions (Mudur and Singh use the term alter-image, which suggest a 2-D world). These **alter-body** functions should be compared to the **actor's mind** since it is them that dictate the moves the **body** must do. Some interesting research has been done to include time explicitly as a fourth dimension [9]. In order to keep the analogy with the real world (mind/body via transformation/model), we will avoid taking this path.

Notation: Let S be the set of **alter-body** functions $F_1, \ldots, F_b$ applied to B. Then $(S.B)^n$ represents the result of dividing S into n parts (n frames) and to apply each "smaller" S to the **body** B in order to obtain the desired transformation. We assume a **B-transformation** to be characterized by its S:

$$(S.B)^n = (F_b.F_{b-1}\ldots F_2.F_1.B)^n$$
$$= (F_b.(F_{b-1}.(\ldots F_2.(F_1.B)\ldots)))^n$$

For example, a simultaneous scaling (F_1) and translation (F_2) of n=96 frames on B can be expressed as $(S.B)^{96} = (F_2.(F_1.B))^{96}$ (see fig. 3).

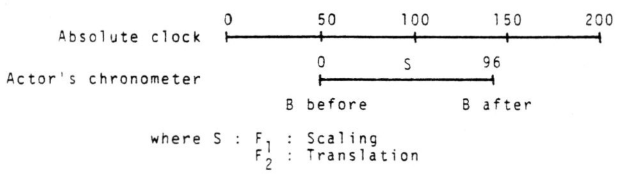


Fig. 3

In a **B-transformation**, the same S is applied to B, regardless of the frame number (in other words, each F must be a continuous function defined the same way). But in real life, an **actor** rarely uses the same set S of moves in a whole performance. For example, an **actor** walks to a table in 5 seconds, then the "result" (which is the **body** B' after applying the set S of "walking **alter-body** functions" to the initial body B in n=120 frames) takes a glass of water in 3 seconds. In the **actor's** computational process, it is possible to apply successively (in time) a series of **B-transformations**, which will be referred to as the **local list of events** (the LLE).

Notation: Let t be the number of **B-transformations** applied to B. Let n_j be the length in frames of the jth **B-transformation**. Let S_j be the set of **alter-body** functions of the jth **B-transformation**. We shall adopt this notation to represent the life of an **actor** A:

$$B_t = (S_t.(S_{t-1}\ldots(S_j\ldots S_2.(S_1.B_0)^{n_1})^{n_2}\ldots)^{n_j}\ldots)^{n_{t-1}})^{n_t}$$

where $\sum_{j=1}^{t} n_j$ = length of **actor**'s life in frames

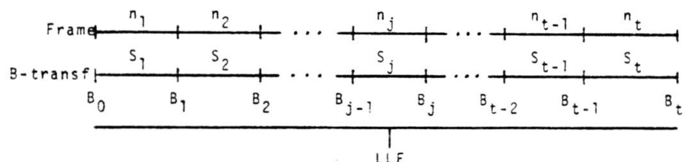


Fig. 4

Fig. 4 shows a schematic presentation. One can see that B_0 is the **actor's body** at his **birth** and B_t the **body** at his **death**. Only the elements to which a **B-transformation** can be applied can move independently. Since many independent elements can be part of an **actor's body** B, it is possible to apply a transformation only to certain points of B.

An animator can add as many **alter-body** functions as needed to any existing **B-transformation**. What happens then if he wants an **alter-body** function to start in the middle of a **B-transformation** and to finish in the middle of another one? A natural solution would be to add a comparison test on the frame number within the **B-transformation**. The new **alter-body** function F would be executed only if the current frame number is greater than the middle value of the **B-transformation**. This is not possible because of the definition of a **B-transformation** (recall that each F must be a continuous function defined the same way, regardless of the frame number). This fact introduces the notion of **parallel LLEs**. It forces the animator to build a new LLE parallel to the existing LLE.

Notation: Let l be the number of LLE applied to B.
Let t_j be the number of **B-transformations** of the j^{th} LLE.
Let n_{ij} be the length in frames of the j^{th} **B-transformation** of the i^{th} LLE.
Let S_{ij} be the set of **alter-body** functions of the j^{th} **B-transformation** of the i^{th} LLE.

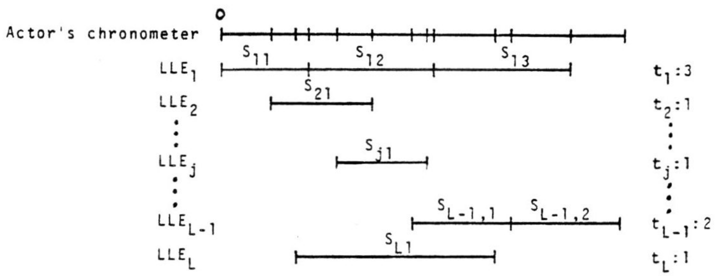


Fig. 5

A schematic presentation is shown in fig. 5. The length of **actor's life** in frames is not any more dependent on the length of a single LLE. The animator can create as many LLEs as necessary. However, he should be careful since the order in which the LLEs are executed on a B is important if they do not commute. Usually, a maximum of two LLEs is sufficient (as in *Dream Flight*).

To clarify explanations, all the transformations involved in the future examples are standard graphic transformations. However, one must understand that the same concepts are valid with any transformation written by the animator. Let us give an example:

LLE_1 ->	S_{11}: F_1 = Scaling/3/its centre		0-100
	F_2 = Translation/(20,30,-10)		
	S_{12}: F_1 = RotationY/0.5 rev/Origin		101-200
	S_{13}: F_1 = StretchX/4/Origin		201-300
LLE_2 ->	S_{21}: F_1 = Interpolation/Bird		150-250
	F_2 = Scaling/0.5/its centre		
	F_3 = RotationZ/1 rev/its centre		
	S_{22}: F_1 = Translation/(40-,30,12)		250-350

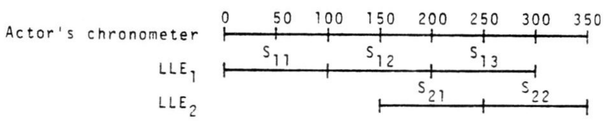


Fig. 6

Fig. 6 represents the same LLEs with schemas. We discover that at frame 225, all the **alter-body** functions of S_{13} ($S_{13}.F_1$) and S_{21} ($S_{21}.F_1$, F_2 and F_3) are successively applied to B. One can see why the order in which the LLEs are specified is so important.

An **actor's body** can stop moving for a moment. The **B-transformation** that handles this time interval is called the **identity B-transformation**. Formally, this means that given any paire of frame numbers within that **B-transformation**, the **actor's body** topological and geometrical information is absolutely identical. All the prop sets (or decor) are static objects. They could also be considered **actors** having one global **identity B-transformation**.

An **actor's body** can also become invisible for a moment without dying. This is called a **null B-transformation**. Unlike the **identity B-transformation**, the **body** must contain no topological information. It is equivalent to not activating the **body**. As an example, a static blinking object would be composed of a cycle containing an **identity** followed by a **null B-transformation**.

EXAMPLE DIRECTOR-ACTORS

In order to make the principles clear, let us show an example of a relationship between a **director** and an **actor**. The length of the performance is 960 frames (40 seconds). Let us describe the GLE:

Birth	Life	Description
115	250	TheCube
400	150	Juggler
500	400	Spiral formation

Fig. 7 shows the GLE.

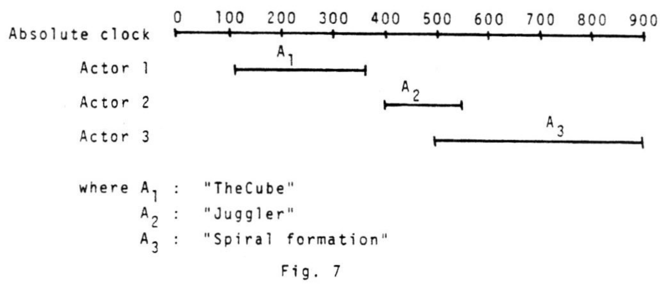


Fig. 7

Let us now present in detail the actor A_1 "TheCube" by first describing its LLEs:

		Frame
LLE_1 ->	S_{11}: F_1 = Translation (5,5,0)	0-70
	S_{12}: Identity B-transformation	70-100
	S_{13}: Null B-transformation	100-160
	S_{14}: F_1 = RotZ/0.25rev/its centre	160-250
	F_2 = Translation/(0,-5,0)	"

LLE_2 -> S_{21}: F_1 = Translation/(42,23,-12) 190-250

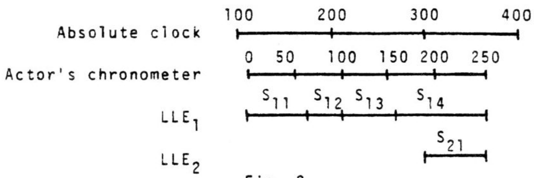


Fig. 8

Fig. 8 shows the same LLEs using the schematic presentation. Finally, fig. 9 show the **actor's life** (without LLE_2) using drawings.

SCRIPT BASED AND INTERACTIVE SYSTEMS

This approach has been designed both for script based and interactive 3D computer animation systems. However, up to this day, the author believes that scripts are the best way to control 3D computer animation. Of course, unlike in interactive systems, animators have only indirect control over the motion. But this is a minor drawback compared to all the advantages. One of the most important advantage in using such system is its flexibility. The set of available commands is virtually unlimited. The facility of writing new commands is dependent on the base language. With an interactive system, a complex motion requires elaborate and cumbersome use of available commands. Another major advantage is the possibility to create and update a "digital stockroom". Animators writing their script can insert any new portion of code in the stockroom

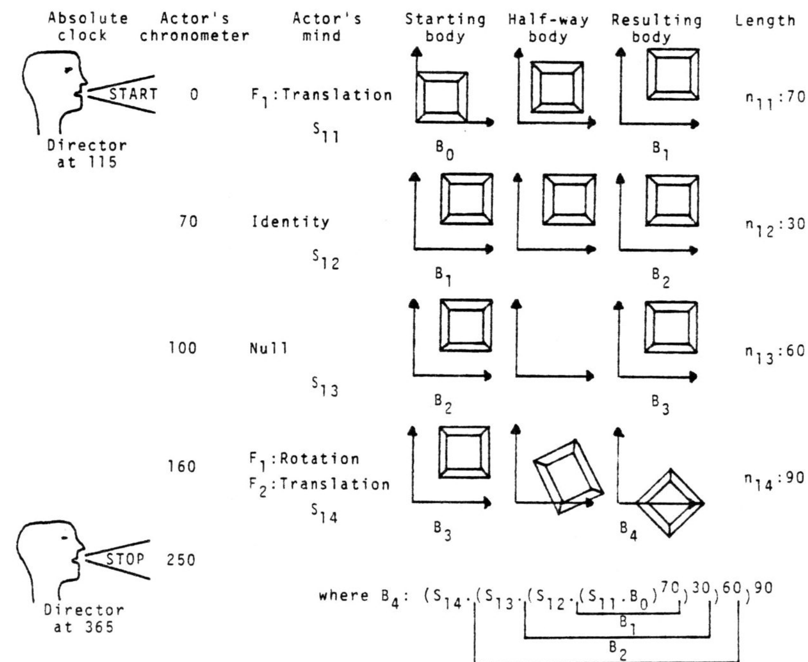


Fig. 9

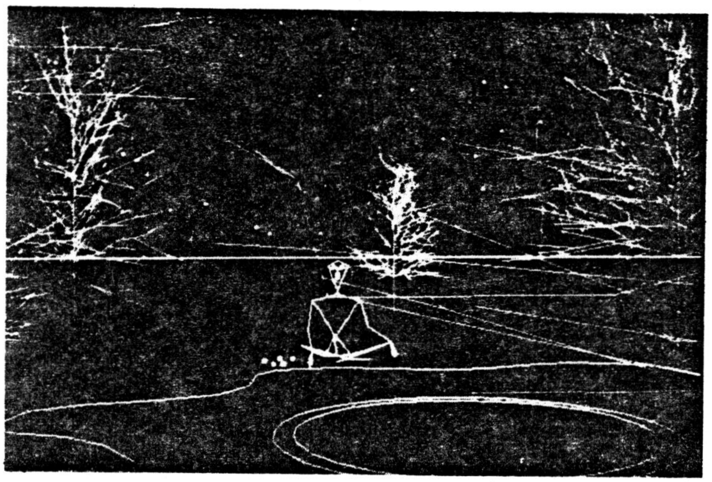
if they think this code is of general use. These portions can be **actors**, **B-transformations**, or any utility function. Later on, anyone can search through the stockroom. An animator may find a portion of code which he would have written otherwise. He may take it as a whole or make minor changes without rewriting the function.

Most of the existing animation languages such as ASAS [6,7] (a LISP extension), or CINEMIRA [11] (a PASCAL extension) can follow our approach. However, since these languages are usually not widely distributed, any high-level common language can follow this proposal. Moreover, it is not necessary for the base language to provide a way of programming concurrent events. The notion of parallel processes such as **actors** can be simulated via any common language.

Let us now describe the concepts of **director** and **actors** in a form of a PASCAL program, using an example with two **actors**: "TheCube" and "Bird". Only the **actor** "TheCube" will be described in detail. Its LLEs are exactly the same as in the previous section. The **light technician** and the **cameraman** are mentioned but not described.

```
program Script (input, output);
const
    RunTime  =300; (* Length os sequence      *)
    StepFrame=  2; (* Test: 2:1 speed ratio   *)
    MaxBTrans=  4; (* Max B-trans. for 1 LLE  *)
    MaxLLE   =  2; (* Max LLEs for 1 actor    *)

type
    Point= record X,Y,Z:real end;
    Model=...(* structure of a model: points, edges,...*)
    Actor= record Birth,Life:0..RunTime;
                  Event:array [1..MaxLLE,1..MaxBTrans]
                        of record Frame:0..RunTime;
                                  FirstTime:boolean
                        end end;
var TheCube, Bird: Actor;
    CubeModel, CurrentBody: Model;
    Fraction:real;
    CurrentFrame:0..RunTime;
(* External procedures:
Let             (var P:Point; X,Y,Z:real                   );
Translate       (A:Model; P:Point; var B:Model             );
RotateZ         (A:Model; P:Point; R:real; var B:Model);
Scale           (A:Model; P:Point; V:real; var B:Model);
Activate        (A:Model                                   );
RotateYPoint    (A:Point; P:Point; R:real; var B:Model);
Camera          (Eye,Int:Point; Spin,Zoom:real             );
ShowActiveBodies                                            ;
TakeOneFrame                                                ;
*)
```



Scene in *Dream Flight*:
- Three alive **actors**: the alien, the stone thrown by the alien into the water, and the waves.
- Five prop sets (or static actors): the star field, horizon, pond, trees, and ground stones.

```
procedure ActiTheCube (RelFrame:integer);
var
    TempPoint:Point;
begin
    with TheCube do
    begin
         if RelFrame <= Event[1,1].Frame then
         begin
             Fraction:=RelFrame / Event[1,1ē.Frame;
S11          Let(TempPoint, Fraction*5, Fraction*5, 0);
             Translate(CubeModel, TempPoint, CurrentBody)
         end else
             if RelFrame <= Event[1,2].Frame then
             begin
                 if Event[1,2].FirstTime then
                 begin
                     Event[1,2].FirstTime := false;
S12                  Let(TempPoint, 5, 5, 0);
LLE1                 Translate(CubeModel, TempPoint, CubeModel)
                 end;
                 CurrentBody := CubeModel
             end else
S13              if RelFrame <= Event[1,3].Frame then
                     CurrentBody := Null
                 else
                     if RelFrame <= Event[1,4].Frame then
                     begin
                         Fraction := (RelFrame-Event[1,3].Frame)/(Event[1,4].Frame-Event[1,3].Frame);
S14                      RotateZ(CubeModel, Origin, Fraction*0.25, CurrentBody);
                         Let(TempPoint, 0, Fraction*-5, 0);
                         Translate (CurrentBody, TempPoint, CurrentBody)
                     end;
         if RelFrame >= Event[2,0].Frame then
             if RelFrame <= Event[2,1].Frame then
             begin
LLE2             Fraction:=(RelFrame-Event[2,0].Frame) / (Event[2,1].Frame-Event[2,0].Frame);
S21              Let(TempPoint, Fraction*42, Fraction*23, Fraction*-12);
                 Translate(CurrentBody, TempPoint, CurrentBody)
             end
    end;
    Activate( CurrentBody)
end;
begin
    with TheCube do
    begin
         Birth:=115;                    Event[1,4].Frame:=250;
         Life :=250;                    Event[2,0].Frame:=190;
         Event[1,1].Frame:=70;          Event[2,1].Frame:=250;
         Event[1,2].Frame:=100;         Event[1,2].FirstTime:=true;
         Event[1,3].Frame:=160;
    end;
    with Bird do ...;

    ... (* creation of propsets (static actors) and models *)

    CurrentFrame := 0;
    while CurrentFrame <= RunTime do    (* director's loop *)
    begin
         LightTechnician(CurrentFrame);          Optional when the actor
         Cameraman(CurrentFrame);                knows his death

         with TheCube do
A1           if (CurrentFrame >= Birth) and (CurrentFrame <= Birth+Life) then
                 ActiTheCube(CurrentFrame-Birth);
Director's loop
         with Bird do
A2           if (CurrentFrame >= Birth) and (CurrentFrame <= Birth+Life) then
                 ActiBird(CurrentFrame-Birth);
         ShowActiveBodies;                  (* Display software         *)
         TakeOneFrame;                      (* Interface with the output *)
         CurrentFrame:=CurrentFrame+StepFrame
    end
end.
```

It is essential in 3D computer animation to be able to perform quick run-through tests of sequences. Unless the system is nearly real-time, it has to allow verification of sequences without checking each single frame. The way the software of the example is written can be classified as absolute, by opposition to incremental. It permits quick run-through tests, that is, a direct access to any frame number within a **B-transformation**. This is why we add the variable StepFrame (instead of the value 1) to CurrentFrame in the **director**'s loop. For example, a sequence with the value 4 assigned to StepFrame will run with a 4:1 speed ratio. If all the **actors** have only one LLE, then the value of StepFrame must be a positive integer smaller than or equal to the length in frames of the shortest **B-transformation** of any **actor**. Otherwise we might "jump" over a **B-transformation**, creating an false effect. There are more restrictions on StepFrame if some **actors** have several LLEs.

There is no doubt for the author that the best way to control 3D computer animation choreography in the future will be by using interactive systems. The designers of such system should be aware of animator's needs such as flexibility. Following our terminology, one should be able to enter interactively a GLE and all the **actor's** LLEs. Moreover, the animator should be free to specify any kind of **B-transformation**. Unfortunately, this point seems to be the most important drawback of existing 3D interactive systems. Usually, the set of available transformations is limited to a standard menu. Any complex transformation can be created only by using a subset of the existing transformations.

CONCLUSION

The author has presented a structured approach on standardization of the concepts, terminology and notation involved in the choreography of a computer animated sequence. This approach is designed both for script based and interactive 3D computer animation systems. It is independent from the base language for a script based system. It is mostly limited to regular motion. Future research should be made to standardize notations for blending functions creating irregular paths through space (curved interpolation), message passing between **actors**, and animation of parameters defining the display software (animating the shadow, for example). The computer animation community must look forward to developing a structured, formal and complete standard for any type of computer animation systems and for any motion. Animation is probably the most "esoteric" field of computer graphics. From place to place, computer animators do not speak the same language. A standard would encourage exchange of ideas as well as stimulating verbal or written communication of specific types of choreography.

ACKNOWLEDGEMENTS

The author is grateful to Pierre Lachapelle and professor Gilles Brassard of Université de Montréal who gave some helpful advices. The concepts have been developed as a thesis requirement for a Masters degree in Computer Science under the supervision of Gilles Brassard.

REFERENCES

[1] BERGERON, P., MAGNENAT-THALMANN, N. and THALMANN, D. The Movie DREAM FLIGHT. *16mm film. Available from Ecole des Hautes Etudes Commerciales, 5255 Decelles, Montreal, PQ, Canada, H3T 1V6. 12 min.*

[2] THALMANN, D., MAGNENAT-THALMANN, N. and BERGERON, P. "DREAM FLIGHT: A Fictional Film Produced by 3D Computer Animation". *Proceedings of Computer Graphics'82, Online Conference, London. October 1982. pp. 353-367.*

[3] MAGNENAT-THALMANN, N., BERGERON, P. and THALMANN, D. "Above Sea and Undersea Computer Animation Scenes." *International Computer Color Graphics Conference'83, Tallahassee, Florida. March 1983.*

[4] GREIF, I. and HEWITT, C. "Actor Semantics of PLANNER-73". *Proceedings of ACM SIGPLAN-SIGACT Conf., Palo Alto, CA, January 1975.*

[5] HEWITT, C. and ATKINSON, R. "Parallelism and Synchronization in Actor System". *ACM Symposium on Principles of Programming Languages 4, L.A. CA, January 1977.*

[6] REYNOLDS, C. "Computer Animation in the World of Actors and Scripts". *SM Thesis, MIT (Architecture Machine Group), May 1978.*

[7] REYNOLDS, C. "Computer Animation with Scripts and Actors". *Proceedings of the ACM SIGGRAPH'82 Conf., Boston, MASS, July 1982.*

[8] MUDUR, S.P. and SINGH, J.H. "A Notation for Computer Animation". *IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-8, no.4, April 1978.*

[9] FOURNIER, A. "A Proposal for a Four-Dimensional Graphics System". *Proceedings of the 7th Conf. of Canadian Man-Computer Communications Society (CMCCS). June 1981.*

[10] "Status Report of the Graphics Standards Committee", *Computer Graphics vol. 13, no 3, August 1979.*

[11] THALMANN, D. "Actor and Camera Data Types in Computer Animation". *Proc. CMCCS'83, Edmonton, Alberta (these proceedings). May 1983.*