Behavioral Animation Modeling in the Windows™ Environment

MARCELO COHEN[1]
CARLA M. D. S. FREITAS[1]
FLAVIO R. WAGNER[1]

[1]UFRGS - Universidade Federal do Rio Grande do Sul
CPGCC - Curso de Pós Graduação em Ciência da Computação
Av. Bento Gonçalves, 9500 Porto Alegre, RS, Brasil
flash@music.pucrs.br
[carla,flavio]@inf.ufrgs.br

Abstract. This paper describes an interactive system for the production of modeled animation. It implements the concept of behavioral animation, providing tools for the definition of actors, behaviors and complete scenes. Actors, behaviors and scenes are modeled following an object-oriented approach. Behaviors can be attached to classes of actors. A prototype was implemented in MS-Windows environment, for validation purposes.

Keywords: Computer Graphics, Modeled Animation, Behavioral Animation.

1 Introduction

Animation-related questions have always been research subject for many groups. The description of modeled animation [Thalmann-Thalmann (1985)], specifically, allows the use of various methods to simplify the movement definition. Among those, stands out the behavioral modeled animation.

In this context, there are techniques based on sensor-effector networks [Braitenberg (1984), Wilhelms (1990)], behavior rules [Reynolds (1987)], genetic algorithms [Reynolds (1993), Yaeger (1993)], relations [Green (1993)] and autonomous agents [Costa et al (1995)], among others.

This work presents a new approach to behavioral animation, where the main idea is to see the user as an animation director instead of an animator. That is, the user must have enough control to “direct” the actors, but not be obliged to define each position, orientation, etc., individually.

As the prototype was developed, however, other control levels were included in the system. It is possible to use a simple definition or a very precise one, if needed.

Three main elements are used in the definition of an animation: **actor classes, behaviors** and **scenes**. Each one contributes on its own to the animation composition. Together, they allow the user to direct the animation. An object-oriented approach was used, allowing the definition of actor classes. Behaviors, after defined, can be associated to classes. The next three sections present the adopted approach to the definition of actors, behaviors and scenes. The next three sections present the adopted approach to the definition of actors, behaviors and scenes. Section 5 and 6 present the developed prototype and the obtained results, respectively, while in section 7 conclusions are drawn, comparing the work described herein with existing ones.

2 Actor classes

Since many actors share common features the idea of a class is straightforward. Frequently, animations have many similar characters - even groups of the same character. The class is its visual representation (**geometry)** and its features **(attributes)**. Each actor in the animation is an **instance of an actor class**.

The classes are hierarchically modeled, each one being derived of another existing class, which helps the creation process of new classes. This characterizes an attribute inheritance process. To allow this kind of modeling, we need a superclass, named in this work as the **Standard Class**.

2.1 Geometry

The geometry describes the object’s visual representation. Like [Watt (1992)], we chose the boundary surface representation (*B-Rep*), that is, polygonal modeling (facets).

There are three reasons for this choice: it is the usual representation in almost all 3D modeling systems, it is easily handled, and it is a representation which helps the collision detection, since it is possible to have an exact idea of an object's volume.

2.2 Attributes

The attributes characterize different actor classes and their contents can be changed in the middle of the animation, and can be used for computations or in control structures.

There are two kinds of attributes: *simple*, which store scalar quantities (velocity, weight) and *composed*, which store vector quantities (direction, force) or quantities represented by 3 components (object position, orientation and scale in 3D space). Both only store real numbers (floating point numbers).

As examples of basic attributes, we have: *position* (of an actor in 3D space), *orientation* (in relation to the 3 coordinate axes: X, Y and Z), *scale* (three scale factors, one for each axis), *direction* (vector indicating direction of the movement), *velocity* (combined with the *direction* attribute gives the **motion velocity vector**), *color* (stored as **R, G, B**).

3 Behavior

The behavior is by far the most important part of the method, because each behavior description specifies what action/movement an actor will perform.

Each behavior is created and associated to a selected class, which means that only actors of this class can have that behavior (and of course, actors of derived classes, due to attribute inheritance).

A behavior is specified by 3 control structures: **command descriptions, constraint definition and event handling**. Each control structure adds an additional element for the movement definition, although not always all three must be used.

3.1 Command descriptions

Command descriptions are done using a simple programming language. This language is inspired on structured languages, such as Pascal, but it can only handle numerical data (simple or composed, as mentioned on section 2.2). There are the following types of instructions:

- attributions: attribute change;
- iterators: repetition blocks (loops);
- conditionals: to perform decision-making during the animation;
- jump to another behavior;
- primitive behaviors: basic actions, predefined and parameterizable, which can be performed by actors. Examples: uniform straight motion, uniformly-varied straight motion, circular motion, spiral motion, projectile launch motion [Ramalho (1990)];
- behavior compositions: direct specification of movement hierarchies, combining primitives and attributions;
- communication with other actors: by message exchange, where each actor can send data (attributes or strings) to other actors;
- creation of a new actor: allows the dynamic creation (during the animation) of a new actor, from a selected class;
- actor elimination: allows the dynamic elimination (during the animation) of the actor from the scene only for visualization purposes. That means the actor is never erased, but only kept hidden.

The combination of the above commands allow the detailed description of a basic behavior for a specified actor class, that is, a behavior that can be linked to any actor of that class. It is worth to note that all behaviors are based exclusively on direct kinematics - velocity producing motion.

As examples, the following pictures show the resulting motion of two kinds of commands:

- a primitive command (projectile launch), specifying: launch orientation (*turn)*, the projectile initial velocity (*vel*), angle of launch in relation to the ground (*ang*) and gravitational acceleration (*g*) (figure 1);

```
LAUNCH (Turn = 0, Vel = 20, Ang =
            80, G = 3)
```

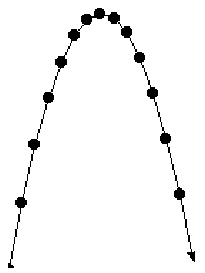


Figure 1 - Projectile launch.

- a behavior composition, generating a spiral motion with decreasing radius, where the actor is lifted (through his Y-coordinate) and, simultaneously, spins around his Z-axis (figure 2).

```
Composition
    SPIRAL (Dir = [0,1,0], Vel = 15,
    Radius = 20, Final = 2, Step = -
    .5)
    Position.y = Position.y + 1
    Orientation.z= Orientation.z  + 8
End
```

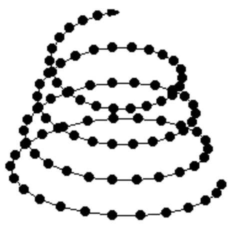


Figure 2 - Behavior composition.

3.2 Constraints definition

A constraint is an additional control structure which can be used in behavior definition. Constraints work as indications of conditions which must be respected during the motion. The idea is to generate the movement defined by the command descriptions, respecting all imposed constraints (conditions).

These constraints can be classified in **seek, approach** and **avoid approaching an actor class**, that means, any actor belonging to that class. Each constraint has a **priority**, which indicates the constraint's relative importance to the others. Therefore, the greater the priority, the greater the constraint relevance. This helps to avoid conflicting situations.

Each kind of constraint imposes a condition: the **seek** constraint, shown in figure 3, forces the actor to go to the direction of *all actors* of the specified class.

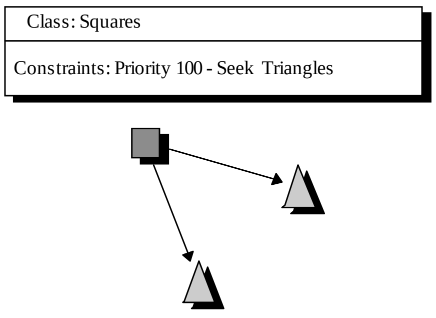


Figure 3 - Seek constraint.

The **approach** constraint (figure 4) adds an additional element: the minimum distance that the actor must keep from its targets.

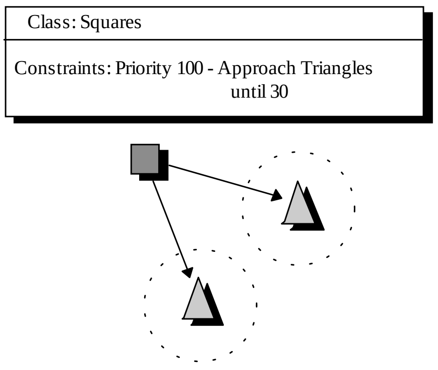


Figure 4 - Approach constraint.

Finally, the **avoid approaching** constraint performs the opposite effect: tries to make the actors approach the least distance from their targets. It is important to note that it is not about **avoiding the targets -** because that implies a direction change **-** but just get as far as possible.

In practice, the constraints evaluation generates a resulting vector, computed from a weighted vector sum of all constraints. This weighted sum uses each constraint priority as a weight. The goal is to obtain a new movement direction, in order to comply with all constraints. But it was observed that this is seldom obtained, because constraints tend to generate conflicting situations like, for example, simultaneously approaching and avoid approaching different classes.

3.3 Events

Many behaviors create situations that demand immediate response from the actors. This means that the behavior evaluation must be prepared to handle events.

Events are special situations which can happen during the animation, requiring the actor's attention.

There are four kinds of events: **proximity** of another actor, **collision** with another actor, **presence** of a new actor (by the *create* command) and **message receiving**. When any event happens, some action will have to be performed, in order to respond to the event. Such response is defined as command among the following: **attribution**, **behavior jump**, **sending of a new message**, **creation of a new actor** and **elimination of the actor**. Like the constraints, events also have an associated priority, which means that events with a higher priority will be processed earlier.

The *proximity of another actor* event requires the specification of the minimum distance from that the event will be generated. This has the objective of avoiding comparisons between actors that are too far from each other. Likewise, the *collision with another actor* event must know which actor class to test a collision with. The creation of a new actor is detected by the *presence of an actor* event. Finally, the *message receiving* event needs the expected message definition. Each message is either a string list and/or an attribute list. In all events it is possible to specify the **ANY** special class, pointing that any actor in the system must be considered, independently of its class.

4 Scenes

A *scene* specifies the actors' initial parameters and their behaviors. Also it is needed the position (in 3D space) from where they will be observed. So, the scene is, effectively, a set of actors and visualization parameters, the last being used by the image synthesis procedure.

4.1 Actors

As mentioned earlier, each created actor is an **actor class instance**, having all of its defined attributes. Each attribute can be initialized by the user with the desired values.

An actor will be represented in the visualization as the three-dimensional object described by its class geometry. This cannot be changed, but parameters such as orientation, scale and color can contribute to differentiate the actor from other actors of the same class.

For an actor to effectively take part of the animation, some behavior must be associated to it. In this sense, any behavior defined for its class (or for the classes where its class was derived from) can be used.

4.2 Scene visualization

Each scene to be visualized must provide the observer's position and orientation to the system. In the prototype, the visualization is done by a synthetic camera package. Therefore, it is also needed the *zoom* (focal angle) parameter.

Every active actor (those who were not canceled by the *Eliminate actor* command) is drawn, represented by its class geometry, at the coordinates stored in its *Position* attribute, at the scale defined by its *Scale* attribute and with the orientation defined by its *Orientation* attribute. The drawing color is anything that is stored in the *Color* attribute.

4.3 Behavior evaluation

From the elements which make a scene, that is, actors and their respective behaviors, it is now possible to describe how the behavior evaluation is effectively done.

First, we must imagine each actor as an **independent processing unit**, executing a single process (its current behavior). The scene's behavior evaluation is, theoretically speaking, a simulation of many processes being executed simultaneously, that is, a **multiprogramming system**, where each processing element executes its own process, independently of the others.

At each simulation step, that is, for each animation frame to be generated, we have to evaluate each actor's behavior in order to create its movement (position, orientation, etc.) In practice, as the system was developed on a single processor environment (80486 CPU), we had to use a *multitasking control*, where many processes are concurrently executed.

The evaluation follows the steps described below, for each actor:

- Execution of the current command in the behavior's command list;
- Constraint evaluation;
- Event handling.

5 Prototype

To validate the method's functionality, a prototype was developed, with the following main goals:

- use of a graphical user interface for all tasks;
- facilities to actor creation, behaviors and scene edition;
- facilities to watch the full animation (or part), without a sophisticated rendering process;
- possibility of the recording of the animation frame-by-frame, to ease the communication with other animation or rendering systems;

The prototype was developed in the *MS-Windows*™ environment, using the *Borland Delphi* programming language [Borland (1995)], which helped make its utilization more intuitive, since that environment is well known to a large number of users.

The event handling is not accomplished through a rigorous formalism, because this was not considered relevant to the prototype functionality.

The prototype operation is done in three main work areas, split by "tabs": actor class edition, behavior and scene.

5.1 Actor class edition

The screen for actor class edition (Figure 5) is divided in work areas:

- class and geometry handling (1);
- attribute editing/deleting (2);
- three-dimensional geometry view (3);
- visualization controls (4);

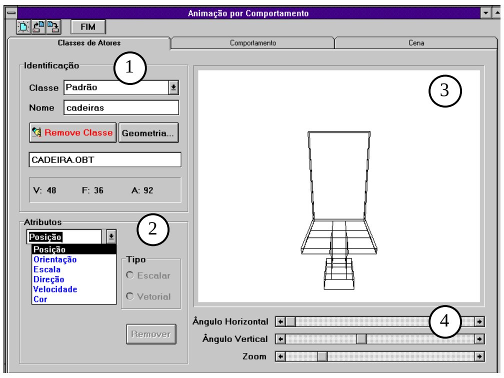
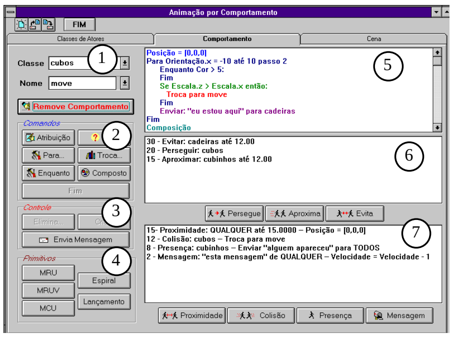


Figure 5 - Actor class editing screen.

5.2 Behavior edition

In this screen (Figure 6) all kinds of behaviors can be edited, including constraints and events. It has seven main work areas:

- behavior creation/selection/deletion (1);
- command selection (2);
- control commands selection (3);
- primitive movement selection (4);
- behavior description edition (5);
- constraint edition/visualization (6);
- event edition/visualization (7).



```
Posição = [0,0,0]
Para Orientação.x = -10 até 10 passo 2
    Enquanto Cor > 5:
    Fim
    Se Escala.z > Escala.x então:
      Troca para move
    Fim
    Enviar: "eu estou aqui" para cadeiras
Fim
Composição
```

```
30 - Evitar: cadeiras até 12.00
20 - Perseguir: cubos
15 - Aproximar: cubinhos até 12.00
```

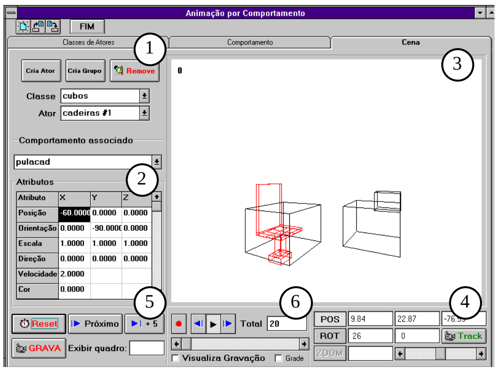
```
15- Proximidade: QUALQUER até 15.0000 – Posição = [0,0,0]
12 - Colisão: cubos – Troca para move
8 - Presença: cubinhos – Enviar "alguem apareceu" para TODOS
2 - Mensagem: "esta mensagem" de QUALQUER – Velocidade = Velocidade - 1
```

Figure 6 - Behavior edition screen.

5.3 Scene edition

Finally, the scene edition is performed at the following screen (Figure 7). The most important work areas are:

- actor creation, selection and deletion (1);
- attribute edition (2);
- three-dimensional scene visualization (3);
- visualization controls (4);
- animation control and recording (5). Allows the individual recording of the frames that can be exported as *POVRay* (ray-tracing package) data files;
- preview generation and exhibition (6). This function allows a quick view of the final sequence, storing each frame in memory and displaying them as fast as possible.



Atributo	X	Y	Z
Posição	-60.0000	0.0000	0.0000
Orientação	0.0000	-90.0000	0.0000
Escala	1.0000	1.0000	1.0000
Direção	0.0000	0.0000	0.0000
Velocidade	2.0000		
Cor	0.0000		

Figure 7 - Scene edition.

6 Results

The results shown here relate only the actors/behaviors description method and the resulting animation. In short, the description method's expressiveness. There are no results with realistic scenes from the esthetic point of view, since this was not the work's objective. Neither the fidelity of the motion's physics has gone beyond kinematics, as mentioned. One analysis of the interface complexity on easiness of use is also beyond the scope of the present work.

As an example, we'll consider a behavior described in the following way:

- **Actors:** "Cubes", "Cylinders" and "Cones", all derived from the standard class, with no additional attributes;
- **Behaviors:** two behaviors were defined, one for the "Cylinder" class and another for the "Cones" class.

The first just produces a spinning motion around the Y-axis, which has the goal to keep the cylinders forever spinning. The second behavior makes the "Cones" actor try to approach the "Cubes", avoiding excessive approach to the "Cylinders".

```
Behavior SpinCylinder
Class Cylinders
Description:
While 1:
    Orientation.y = Orientation.y + 15
End while
```

```
Behavior MoveCone
Class Cones
Description:
While 1:
    Position = Position
End while
Constraints:
    Seek Cubes (priority 30)
    Avoid Cylinders until 40 (priority 20)
```

This scene was created by positioning cylinders between the cone and the cube. Since the goal is to take the cone to the cube position, the first must avoid the obstacles (cylinders), while following his path. The following pictures show three moments (frames) of the animation: the beginning (Figure 8), frame 45 (Figure 9) and frame 60 (Figure 10), where the resulting motion can be observed.

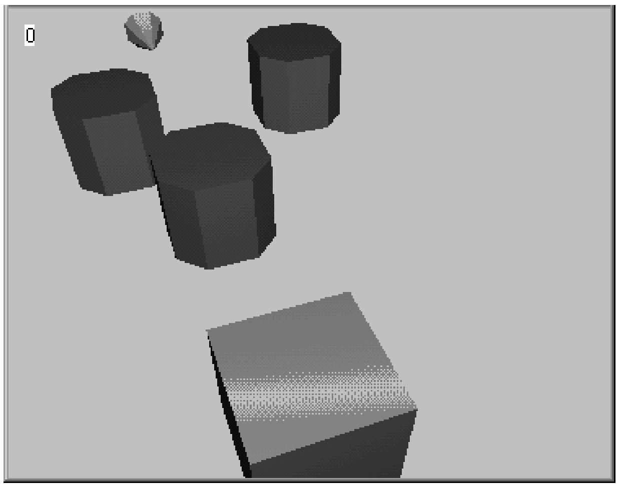


Figure 8 - Beginning of animation (frame 0).

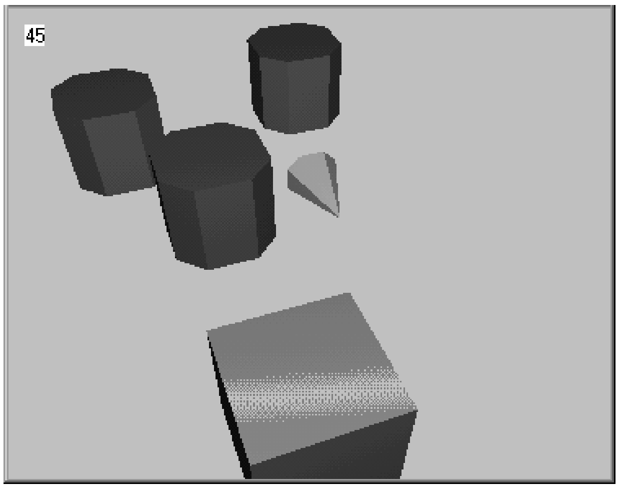


Figure 9 - Middle of animation (frame 45).

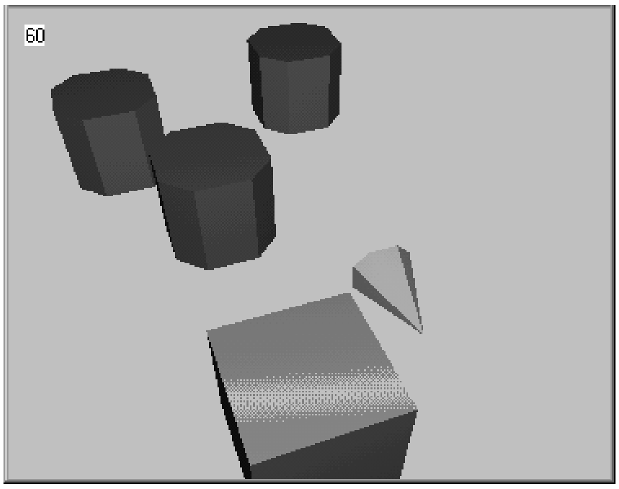


Figure 10 - End of animation (frame 60).

7 Conclusions

The present work presented a new approach for modeling behavioral animation. This approach, implemented and validated by the prototype, did show some drawbacks, due to its experimental nature.

Despite of these drawbacks, the system is easily extensible, including new functions, new recording methods, new behavior evaluation methods, etc.

Some ideas are proposed for future work:

- substitution of the collision detection algorithm by a more precise one (the current algorithm just checks collisions between the actors' bounding boxes) [Moore-Wilhelms (1988)];
- constraint evaluation reformulation, trying to avoid conflicting situations or, at least, provide an alternative solution;

- event handling reformulation, using a more rigorous and formal treatment [Kalra-Barr (1992)];
- possibility of camera motion (create a special class for cameras, for example).

References

V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology*, MIT Press, 1984.

M. Costa, B. Feijó, D. Schwabe, *Reactive Agents in Behavioral Animation*, Anais SIBGRAPI'95, 1995, pp. 159-165.

Borland International, *Delphi User's Guide*, 1995.

M. Green, S. Hanqiu, *The Use of Relations for Motion Control in na Environment with Multiple Moving Objects*, Proc GRAPHICS INTERFACE'93, 1993.

D. Kalra, A. H. Barr, *Modeling with Time and Events in Computer Animation*, Proc EUROGRAPHICS'92, **13.**, Academic Press, 1992.

M. Moore, J. Wilhelms, *Collision Detection and Response for Computer Animation*, Proc. SIGGRAPH'88, 22(4), ACM Press, 1988, pp. 289-298.

F. Ramalho Júnior, *Os Fundamentos da Física*, 5^{a} ed. Ed. Moderna, 1990.

C. Reynolds, *Flocks, Herds and Schools: a Distributed Behavioral Model*, Proc SIGGRAPH'87, Computer Graphics, **21**(4), 1987, pp. 25-34.

C. Reynolds, *An Evolved, Vision-Based Model of Obstacle Avoidance Behavior*, Proc ARTIFICIAL LIFE'93, **3.**, 1993.

N. Magnenat-Thalmann, D. Thalmann, *Computer Animation: Theory and Practice*, Springer-Verlag, 1985.

J. Wilhelms, R. Skinner, *A "Notion" for Interactive Behavioral Animation Control*, IEEE Computer Graphics and Applications, 1990, pp. 14-22.

A. Watt, M. Watt, *Advanced Animation and Rendering Techniques: Theory and Practice*. ACM Press, 1992.

L. Yaeger, *Petworld and other subjects in artificial life*, personal communication by electronic mail, Oct. 1993 (larryy@apple.com).