One Method of the Human Body Model Animation

R. Berka, I. Jelínek
Department of Computers
Faculty of Electrical Engineering
Czech Technical University
Karlovo nám. 13
121 35 Prague 2
Czech Republic

e-mail
berka@cs.felk.cvut.cz
jelinek@cslab.felk.cvut.cz

Abstract
In the first part of the paper we show a new approach to the modelling of the human body. The modelling of the human body has two parts - the model structure description (the model of the human body we call "figure") and the model motion control. The second part of this article is devoted to methods of the the figure's structure formal description and to control of the figure's behavior. For this purpose we have proposed a special formal language FDL (Figure Description Language). The next part informs on the implementation our ideas in program VITALIATOR. The implementation is consistently based on the object oriented approach. In the last part of our contribution we discuss questions connected with control of all actor's body parts and parallel movement control of more figures.

Keywords
computer animation, object-oriented graphics, human body animation, modelling

1 Introduction

Three-dimensional computer animation has advantages not available in traditional animation - animation can be produced directly from the model of an animated object. This possibility is very advantageous in the three-dimensional model animation and visualization. A special interesting kind of modelled animated object is the human-body.

The human-body is an articulated 3D object. The description of the human-body model structure and the model movement is a serious problem. In this report, we will describe a special formal method for controlling the human-body model.

The problem of the human-body model movement control is not a new one. The first approaches and methods are described in [1], [2], [3], [4], [5]. The system NUDES [1] is based on stick figure models which consist of a hierarchy of ellipsoids. This approach has an advantage in very fast hidden-surface problem solving, but movement of every part of each stick must be controlled independently, the view of the model on the screen is very primitive.

An interesting method is presented in [1]. The human-body is described and modelled by a special notation - the LABANOTATION. Human-body is structured in a hierarchical way, every part of this structure can be identified by its own identifier and controlled by five kinds of movement operators. The animation of the human-body model can be described in a simple language. Every statement of this language consists of the identifier of the human-body model part and a movement operator. But, controlling of the movement of the model is very complex and complicated. The similar method is presented in [4]. The parts of hierarchical structure are not identified by the name, but every part of the human-body model is described by a relativity parameter to the predecessor in the hierarchy. The parameter of relativity has values of position, angle, twisting etc. The problems of this approach are the similar ones as in the publication [5].

In our research we solve the problem of human-body model movement implementation by a new approach. Computer graphics is a typical application where the object oriented paradigm is very usefull [6]. As the human-body is an articulated 3D object with a well defined structure, and the control of human-body model is a typical polymorfism task, we decided to use an object oriented method for the implementation of our approach.

2 THE HUMAN BODY MODEL

2.1 THE SKELETON DEFINITION

For easier control, the human body is in our idea modelled by a simple skeleton. The main components of this skeleton are nodes. The nodes represent the joints of the real human body and they play an important part in data representation of this model and in its control (see Fig. 1).

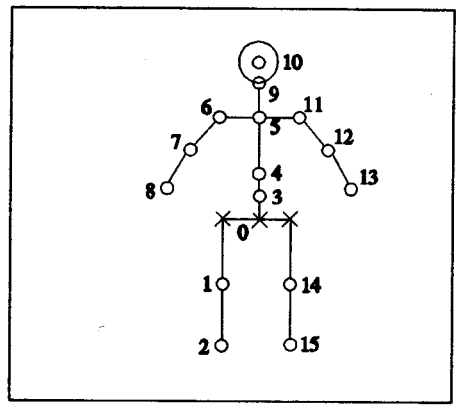

Figure 1: Figure model

Looking over the Fig. 1 we can imagine the skeleton as the oriented ternary tree where nodes of the tree correspond with nodes of our model, and its edges are oriented from the root of the tree to the leafs (see Fig. 2).

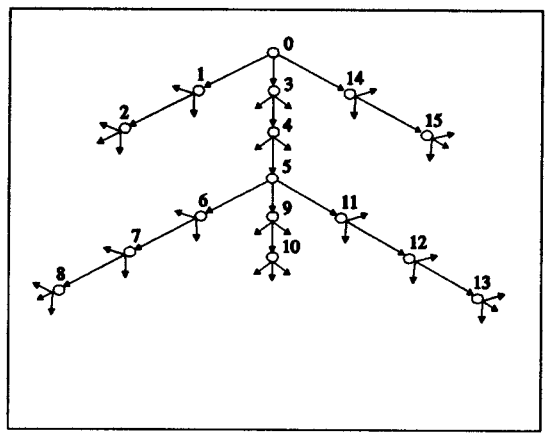

Figure 2: Ternary tree representing the model

Every node has its own coordinate system. The node is placed in the space relatively to coordinates of its predecessor.

There is a base set of operations defined over each node to control its state. All changes of position of the node are always performed by rotation around the one of three axes (X, Y or Z) in coordinates of the predecessor (operation ROT see section 3.2). The position-change of any node means position-change of all its successors, but other nodes stay untouched.

The root node plays a special role in the tree. Every other node is positioned in the space relatively to the root node. The operation MOVE, which shifts whole structure in the space, can be performed only with the root node.

2.2 THE MOTION CONTROL

The figure behavior in the scene we can control performing some operations over the model. The first two operations, MOVE and ROT, were mentioned in previous text.

MOVE has effect only in connection with the root node. ROT has three forms - ROTX, ROTY, ROTZ round axes X, Y or Z. ROT applied to the root node rotates the whole figure in coordinates of the root node. Application to every other node performs rotation of this node round any axis in its predecessor's coordinates.

Some other operations over the figure's structure are shown in section 3.2 concerning the figure behavior describing language.

The position of the node is determined by its placement relative to the root node and by the position of the root node in the world coordinates. This is the foundation of transformations for each node from its coordinates to the world coordinates. The transformations are expressed by transformation matrices. Fig. 3 shows a simple structure where the relation between local coordinates of node L_i and world coordinates is given by concatenation of transformations between every two following predecessors L_j and L_{j-1} of node L_i ($j \in \langle i,1 \rangle$, $i \in \langle 2, k \rangle$, where k is the number of all the model's nodes).

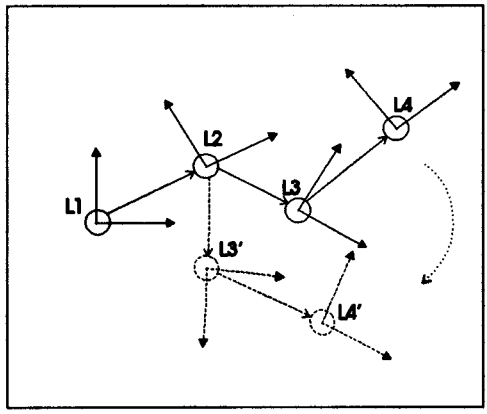

Figure 3: Fragment of the model structure

When any node changes its position, transformation matrices corresponding to all successors of this node must be re-computed. Matrices corresponding to other nodes keep their original state.

Describing structure and behavior of one or more figures in the scene, we need any formal language to determine how many actors are in the scene, what is their shape, and when and what the figures must do. The next chapter is devoted to this problem.

3 FORMAL DESCRIPTION

3.1 FIGURE DESCRIPTION

The formal language describing the skeleton's structure is generated by the grammar $\mathcal{G}$:

$$\mathcal{G} = \{ \mathcal{N}, \mathcal{T}, \mathcal{R}, \mathcal{S} \}, \quad \text{where}$$

$\mathcal{N} = \{ \mathcal{S}, \mathcal{O}, \mathcal{P} \}$..... is the set of non-terminal symbols

$\mathcal{T} = \{$!, $, @, &, x, y$\}$. is the set of terminal symbols

$\mathcal{S}$ ..... is the start symbol from set N

$\mathcal{R}$ ..... is the set of rules

1. $\mathcal{S} \rightarrow$!x$\mathcal{OOO}$
2. $\mathcal{O} \rightarrow$ &y$\mathcal{PPP}$
3. $\mathcal{P} \rightarrow$ @
4. $\mathcal{P} \rightarrow$ \$$\mathcal{O}$

The script analysis starts by the finding of symbol "!". The first rule represents the rise of the root node. Symbol "x" substitutes information about figure position, orientation, and color and shape of the root node. Three symbols "$\mathcal{O}$" mean three successors of the root node.

Other nodes are generated after the second rule. Symbol "&" is a prefix starting every new node in the script. Symbol "y" represents information about the position (relatively to coordinates of its predecessor), color and shape of the part between the node and its predecessor.

Three symbols "$\mathcal{P}$" represent three successors of the node. In this case the successor can or cannot be generated. It depends on the symbol found after generation of the current node. The symbol "@" means the current node is the leaf of the tree, but the symbol "$" is followed by any information about the successor of a current node.

3.2 THE FORMAL DESCRIPTION OF THE FIGURES' BEHAVIOR

Describing behavior of actors in the scene we need a simple formal description. This description should contain a time factor defining time-proportions among actions in the scene. Next, we request the notation containing a description of the figures' structure and a description of their behavior.

For this purpose, we made very simple language containing 9 instructions. Some of these instructions are corresponding to operations over-the-figure described in the sections 2.1 and 2.2 about the human body model.

The instructions have the format as follow:

- Instructions rotating by component-of-body round any axis
  - ROTX <T> <Fi> <Pj> <An> <NS>
  - ROTY <T> <Fi> <Pj> <An> <NS>
  - ROTZ <T> <Fi> <Pj> <An> <NS>
- Instruction shifting whole figure
  - MOVE <T> <Fi> <Pi> <X> <Y> <Z> <NS>
- Instruction defining length of one time tick
  - DEFT <T>
- Instruction creating the figure
  - MAKF <T> <figure formal description after grammar $\mathcal{G}$ >
- Instruction deleting the figure
  - KILF <T> <Fi>
- Instruction drawing the figure
  - SHOW <T> <Fi>
- Instruction hiding the figure
  - HIDE <T> <Fi>

Note: Each action starts when T equals to a global time counter value.

The meaning of used symbols:

T ..... time counter value
Fi ..... figure identifier ($i \in \langle 1, N \rangle$, where N is the number of all figures)
Pj ..... body part identifier ($j \in \langle 2, k \rangle$, where k is the number of all the figure's nodes)
An ..... rotation angle increase
X,Y,Z ... coordinates increase (shifting)
NS ..... number of steps (phases) the An should be divided to

This language is very easy to use. A screenplay described by this formal notation contains both - the structure of each figure, and its behavior in the scene.

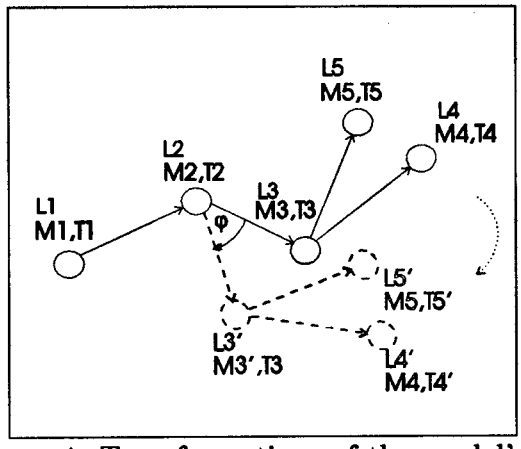

Figure 4: Transformations of the model's parts

The model described in previous sections and formal notations described in this section, could be used in different forms of implementation. The ideas of ternary tree, nodes as joints of a real human body, or the addressing of nodes in notation offer itself to be implemented in any object oriented language. In the next chapter we will show implementation of these methods using C++.

4 THE PROGRAM VITALIATOR

4.1 THE FIGURE'S DATA REPRESENTATION

The data representation for the human body is drafted as a ternary tree. Every node is represented by the object of class *Limb*. Information contained in the data structure of this object, express position of node in coordinates of predecessor, in tree-structure, color and shape of component between predecessor, and current node.

The data structure of the node contains two important transformation matrices M_i and T_i ($i \in \langle 1, k \rangle$, where k is the number of all figure's nodes). The matrix M_i represents transformation from coordinates of the current node to coordinates of its predecessor.

The matrix T_i is a composition of matrices M_i from all predecessors and represents a transformation from coordinates of the node to world coordinates (see Fig. 4).

$$T_i = \prod_{j=1}^{i-1} M_j \tag{1}$$

The object of class *Limb* owns methods to receive and to process messages from the neighbor node. Every message has a uniform format containing type of the message, a part identifier, a number of steps and a data area. Reading the message type information, the receiving object determines what algorithms will be used for the next transformations. The part identifier serves for decision whether the received message concerns this node or not. The data area contains one real number (representing a rotation angle for a node) or three real numbers (representing changes of three axes X, Y or Z). The second type of the message (translation) is accepted only by the root node.

The number of steps determines to how many phases the position change (translation or rotation) will be divided.

The root node has special functions in the described structure to fix the whole skeleton in the space of the world coordinate system. The next part of the root node is to receive messages from superior data structures and to distribute them to the tree-structure of other nodes.

The message format for the root node is similar to the format for other nodes, but in addition the root node receives a figure identifier. The figure identifier has the meaning of information which determines if the received message is addressed to the current figure or not. The root node is represented by an object of a class *Figure* inheriting some methods and data from the class *Limb*.

An object of class *Figure* can perform special transformations with a whole skeleton, which are not necessary for other nodes (for example, translation or rotation of the whole figure around one of the coordinate axes).

4.2 GENERATION OF THE FIGURE USING THE FORMAL NOTATION

In the previous chapters, the figure's model and its representation is described using an object-oriented paradigms. The description of the figures and their behavior comes to VYTALIATOR's input in form of a script containing the simple commands (see section 3.2). The analyzer reads the text and builds the dynamic data structure representing the figure. All objects representing the figure are generated by this way.

The constructor of class *Figure* (see the previous section) is called for every figure to be placed in the scene. This constructor calls a constructor of class *Limb* three times to build three successors of the root node (see Fig. 1 and Fig. 2). The constructor of class *Limb* calls itself recursively also three times and whole structure is completed when an analyzer read all data from the script.

4.3 THE FIGURE CONTROL TECHNIQUE

The script mentioned above contains both commands to create figures and commands to control behavior of these objects in the scene. The commands are read and sent by the messages to the figure in form of instructions. Every message contains identifier of figure, identifier of node and time counter value (saying when the message should be processed).

Determining when the message will be processed we need a new object as interface between figure and its environment. This object is called *Queue Manager* and its main component is a queue of instructions. This queue stores instructions containing a time counter value giving information when the instruction will be sent by message into the structure. The time is represented by an integral value of a counter which is managed by the superior data structure. The rest of instruction contains information about the type of instruction, the number of steps and operands.

Receiving a message *Queue Manager* stores it to the end of the queue as an instruction. If the value of global time counter corresponds with the time counter in the first instruction from the queue *Queue Manager* sends this instruction into the structure of the figure. The object of class *Limb* which identifies itself as a receiver

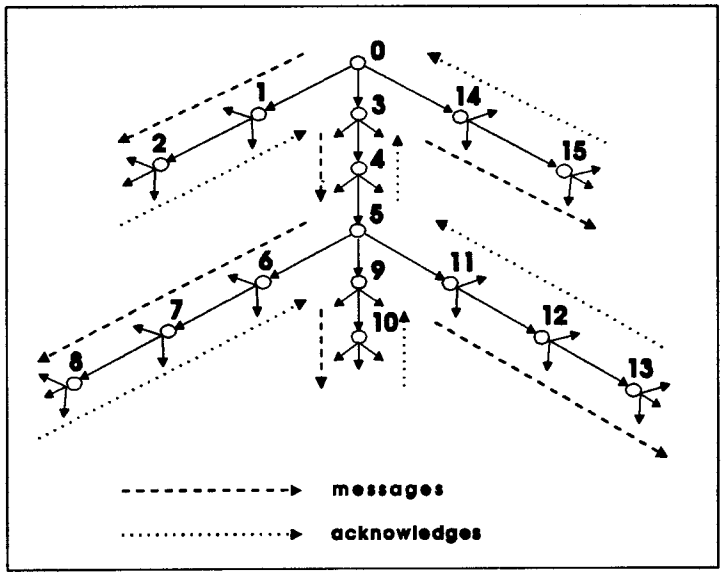

Figure 5: Data flow in figure's structure

of this message perform transformation determined by the received message. The transformations are executed only partly and the rest of the calculation is inserted, as the last instruction in set of instructions with the same time value, to the queue of *Queue Manager*. Fig. 5 shows data flow in the figure's structure which receives messages with instructions and returns results of performed operations.

Using this control system we can achieve a pseudo-parallel effect when two messages in different nodes of the structure, but with the same time value, are processed simultaneously. In this case two different figure's components move at the same time.

4.4 THE DATA STRUCTURE FOR THE MULTI-ACTOR CONTROL

In the previous text we presented the basic data structure, the simple description, and the control technique for one figure in the scene only. In the following part we will show how to extend the current data structure to control more then one figure in the scene.

In the system controlling more than only one figure, it is necessary to base some global structure and methods which can communicate with all figures via the messages. In this global structure the figure is represented by *Queue Manager* receiving the messages. The discussed global structure is solved in our project by a new class *Scheduler*. The object of class *Scheduler* manages all actors via messages. Reading an instruction from input, *Scheduler* sends it as a message to all figures. The figure, represented by *Queue Manager*, identifying itself as receiver of this message puts the instruction into the queue.

Scheduler sends messages into linked structure of *Queue Managers* in the cycle. In one cycle every *Queue Manager* of the structure can process one instruction from the queue independently whether *Queue Manager* is the receiver of the message or not. *Queue Managers* return acknowledge, containing information about the status of their queues. If any *Queue Manager* has full queue it indicates this situation by a returned message. The same principle is used when all *Queue Managers* have empty

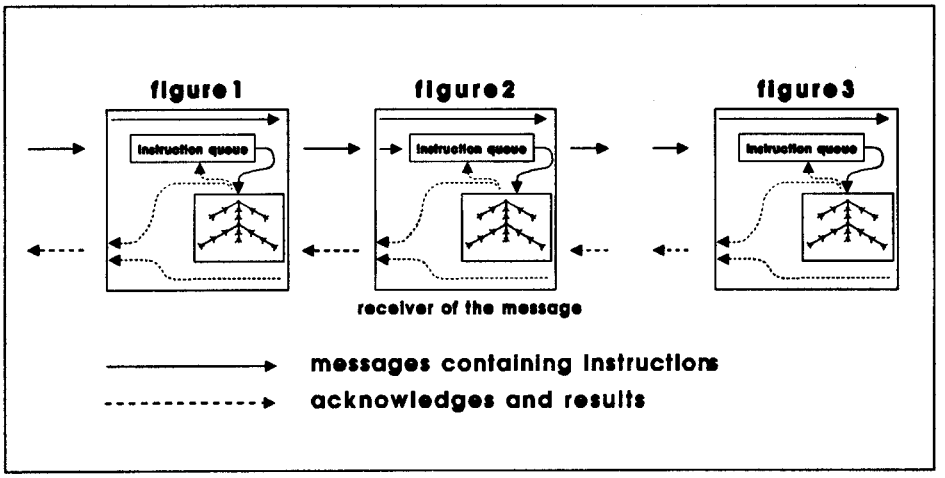

Figure 6: Data flow among linked figures

queues or whether actualizing of the global time-counter is necessary.

The time value of the first instruction in the queue represents the local time of *Queue Manager* and this instruction is processed only if the local time value is lower or equal to the global time value. The need of the global time-counter actualization appears when all *Queue Managers'* local time value is higher than the global time value. This is the principle of the time synchronization. The data flow among all figures is shown on Fig. 6.

Using the *Queue Manager* to control all the components of the figure we can achieve pseudo-parallel effects in the structure of the figure. Using the *Scheduler* we can achieve a pseudo-parallel effect in the whole scene. This means the parts of figures and whole figures can move independently of other figures and their parts.

5 Conclusion

In this paper we have shown the modelling technique for the animation of a human body, methods for the formal description of the data structures, and a formal description of an actor's behavior in the scene using the object-oriented paradigm.

These methods have become the basis for realization of our program called VITALIATOR. VITALIATOR is implemented in C++ and enables the animation of a human body in real-time. The structure of the animated body and its behavior are described by a screenplay language which is being read in the form of input text. First results with the VITALIATOR prompts that demands for control of figures in the scene aren't as high as we expected. The memory requirements and computing time requirements are linear dependent on the number of actors in the scene.

On the other hand the drawing algorithms are very time-consuming and their use must be supported by hardware.

In the future we will try to solve these problems using some hardware with support of any frequent operations. As we have indicated, the problems would be specifically with drawing time, and therefore the hardware support should be directed into a graphic sub-system (Z - buffer or some form of parallelism).

References

[1] Thalmann, D. - Magnenant-Thalmann, N.: *Computer Animation, Theory and Practice*, Springer Verlag, 1990

[2] Jelínek,I.: *Computer Graphics Applications*, Czech Technical University, Prague 1991, in Czech

[3] Watt, A. - Watt, M.: *Advanced Animation and Rendering Techniques*, Addison-Wesley, 1992

[4] Thalmann, D. - Magnenant-Thalmann, N.: *Synthetic Actors in Computer-Generated 3D Films*, Springer Verlag, 1992

[5] Thalmann, D. - Magnenant-Thalmann, N.: *State-of-the-art in Computer Animation*, Proceedings of Computer Animation '89, Springer Verlag, 1989

[6] Nenadál, K. - Václavíková, D.: *Borland C++, Object Programming and Language Description*, Grada, Prague 1992, in Czech

[7] Berka,R.: *Object oriented animation*, MSc Thesis, CTU Prague, 1994, in Czech

[8] Berka,R., Jelínek,I.: *VITALIATOR - The Human Body Animating System*, CTU Seminar '94, CTU, Prague 1994, pp. 45 - 46

[9] Berka,R., Jelínek,I.:*Formal Methods in Human Body Model Animation*, 10. SSCG, Bratislava, June 1994, Slovakia, pp. 220 - 225