International Journal of Software Engineering
and Knowledge Engineering
Vol. 13, No. 1 (2003) 1–25

World Scientific
www.worldscientific.com

VISUALIZING ANIMATION DATABASES

AKANKSHA[*,§], Z. HUANG[*,¶], B. PRABHAKARAN[†,‖] and C. R. RUIZ, JR.[‡,**]

[*] *School of Computing, National University of Singapore, Singapore*
[†] *Department of Computer Science, University of Texas at Dallas, USA*
[‡] *College of Computer Studies, De La Salle University Manila, Philippines*
[§] *akanksha@comp.nus.edu.sg*
[¶] *huangzy@comp.nus.edu.sg*
[‖] *praba@utdallas.edu*
[**] *consruiz@email.com*

Accepted 15 October 2002

We consider a repository of animation models and motions that can be reused to generate new animation sequences. For instance, a user can retrieve an animation of a dog kicking its leg (in air) and manipulate the result to generate a new animation where the dog is kicking a ball. In this particular example, inverse kinematics technique can be used to retarget the kicking motion of a dog to a ball. This approach of reusing models and motions to generate new animation sequences can be facilitated by operations such as querying of animation databases for required models and motions, and manipulation of the query results to meet new constraints. However, manipulation operations such as motion retargeting are quite complex in nature. Hence, there is a need for visualizing the queries on animation databases as well as the manipulation operations on the query results.

In this paper, we propose a visually interactive method for reusing motions and models, by adjusting the query results from animation databases for new situations while at the same time, keeping the desired properties of the original models and motions. Here, a user first queries for animation objects, i.e., geometric models and motions. Then, the user interactively makes new animations by visually manipulating the query results. Depending on the orders in which the GUIs (Graphical User Interfaces) are invoked and the parameters are changed, the system automatically generates a sequence of operations, a list of SQL-like syntax commands, and applies it to the query results of motions and models. With the help of visualization tools, the user can view the changes before accepting them.

Keywords: Animation databases; human computer interactions; metadata of animations; reusing animations; visualization.

1. Introduction

Computer animation is a subject with both theoretic research interest and practical applications in multimedia production, entertainment, simulation, etc. Reusing has become an interesting approach for computer animation with the popularity of the use of motion capture devices and 3D digitizers [16, 11, 12, 8]. The main idea in

reusing animations is to adjust the existing motion sequences for new situations while at the same time, keeping the desired properties of the original motion. In [8], motions are considered as signals so that the techniques of signal processing can be applied to adapt them. A variant of the above method called the motion-displacement mapping was discussed in [16]. In [11, 12], the problem is retargeting motion of one articulated figure to another with the same structure but different lengths.

Database approach for reusing animations is to adjust the query results from animation databases for new situations while at the same time, keeping the desired properties of the original models and motions. Here, motion sequences along with geometric models represented as a hierarchical data structure (called *scene graphs*) are stored in a database. A set of object metadata is defined and used for the storage and query of the animations in the database. New motion sequences and models can be created through a series of operations that manipulate the query results.

1.1. *Need for visualization*

Manipulations of query results (i.e., geometric models and motions) are complex operations. For instance, Fig. 1 shows one example where a 3D humanoid is reaching the bar located in the same position and orientation. In Fig. 1(a), the target was reached with the elbow hanging high (the original motion). By specifying the height limit of the elbow as a constraint, a new animation in Fig. 1(b) can be generated. (For brevity, only the last frame of each motion sequence is shown in the figure.)

In the above example, a new animation is generated by making the lifting motion of the elbow joint to meet a different height constraint. This type of manipulation can be carried out by using a technique called *inverse kinematics*, originally developed for robotics control [9]. Inverse kinematics is a motion editing technique

(a) (b)

Fig. 1. Same target in (a) and (b) but reached with different final postures. In (b), a secondary task is specified for the elbow to constraint its height.

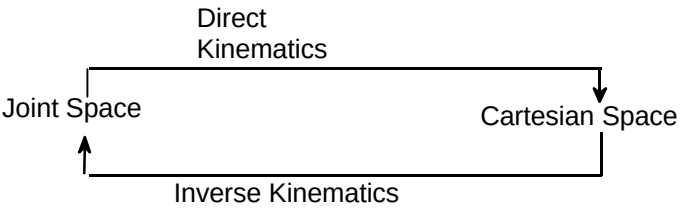


Fig. 2. Direct and inverse kinematics.

especially for *articulated figures*. An articulated figure is a structure consisting of multiple components connected by joints, e.g., human and robot arms and legs. An *end effecter* is the last piece of a branch of the articulated figures, e.g., a hand for an arm. Its location is defined in Cartesian space, three parameters for position and another three for orientation. At each joint, there are a number of degrees of freedom (DOFs). All DOFs form a *joint* (*or configuration*) space of the articulated figure (Fig. 2). Given all the values of DOFs in joint space, the kinematics method to compute the position and orientation of end effecter in Cartesian space is called *direct kinematics*. Inverse kinematics is its opposite.

An end effecter location depends on the current joint state. The set of non-linear equations establishing the end effecter location as a function of the joint state is called the direct geometric model in Robotics. Inverting it is possible if the dimensions of joint space and Cartesian space are the same. The motion editing is done using inverse kinematics by specifying constraints to the end effecters. More constraints can be specified if the secondary task is used. After the specification, the inverse kinematics solver can compute the changes for each DOF in the joint space. Thus, an existing motion sequence will be adjusted to meet the constraints. Let us illustrate it by one example of motion retargeting. For ease of understanding, we illustrate it in a 2D case. In Fig. 3(a), it shows a movement of the articulated figure to have the end effecter to reach the target. This motion sequence represents the original motion. Now we move the target point to a new position (Fig. 3(a))

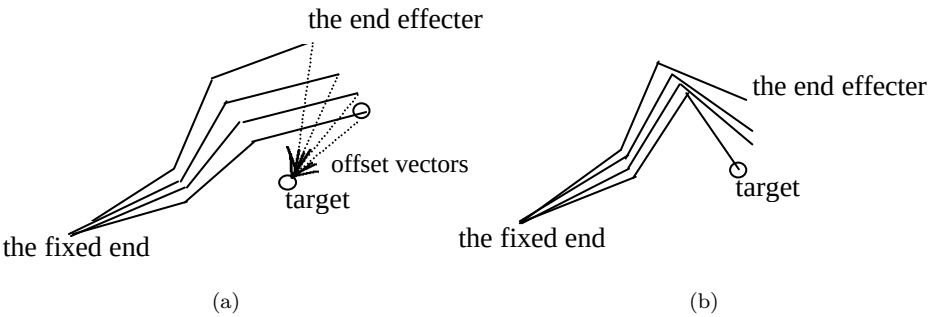


Fig. 3. Retargeting to generate a new motion.

and show how inverse kinematics can be applied to adjust the original motion, i.e., retargeting, generate a new motion (Fig. 3(b)).

Motion in Fig. 3(b) is achieved by applying the following equation for inverse kinematics:

$$\Delta\theta = J^{+}\Delta x + (I - J^{+}J)\Delta z \tag{1}$$

where:

- $\Delta\theta$ is the unknown vector in the joint variation space, of dimension n.
- Δx describes the *main task* as a variation of the end effecter position and orientation in Cartesian space. For example in Fig. 3(b), the main task assigned to the end of the chain is to follow a curve or a line in the plane under the small movements hypothesis. The dimension m of the main task is usually less than or equal to the dimension n of the joint space.
- J is the Jacobian matrix of the linear transformation, representing the differential behavior of the controlled system over the dimensions specified by the *main task*.
- J^{+} is the unique pseudo-inverse of J providing the minimum norm solution which realizes the *main task*.
- I is the identity matrix of the joint variation space $n \times n$.
- $(I - J^{+}J)$ is a projection operator on the *null space* of the linear transformation J. Any element belonging to this joint variation sub-space is mapped by J into the null vector in the Cartesian variation space.
- Δz describes a *secondary task* in the joint variation space. This task is partially realized via the projection on the *null space*. In other words, the second part of the equation does not modify the achievement of the main task for any value of Δz. Usually Δz is calculated so as to minimize a cost function.

The inverse kinematics solver applies the above equation (1) (no secondary task, thus, $\Delta z = 0$) to Fig. 3(a). Using (1) the offset vector Δx of the target and the end effecter position for each frame, and (2) the configuration at each frame as initial posture, it (the solver) automatically generates a new configuration in joint space to reach the new target.

Clearly, the task of using manipulation operations on query results is quite complex in nature. Querying for models and motions in animations is also involved. Hence, there is a need for visualizing the querying and manipulation operations in animation databases.

1.2. *Visualization of animation databases*

Visualization of animation databases is promising for reuse of motions and models. Some reasons are:

(1) The data of animation is visual in nature. The 3D models for animation are represented geometrically. The quality of animation results is judged visually too (though there are efforts to devise automatic evaluation algorithms but the results are still not acceptable).
(2) Reusing animation involves the process of manipulating 3D models and adjusting motion parameters. The human computer interaction by using GUIs and visualization is a natural way.
(3) Reusing animation requires a series of SQL-like commands to access animation databases. For deriving a simple animation, such a command list is too difficult for users to write correctly. By using GUIs and visualization, it will become an intuitive process and the correct command list can be generated automatically.

In this paper, we propose a visually interactive method for animation database in which a set of graphical user interfaces (GUIs) is developed that converts the human-computer interactions into sequences of operations on the query results of motions and models. These GUIs translate the user defined operations and parameters into SQL-like commands. The GUIs also allow users to modify the properties of geometric models and motion sequences that are returned as results to a user query. For example, a user query can be for a running motion of a human model. The resulting running motion can be modified through another GUI by changing certain properties, say, the speed of running.

Organization of the paper: Next, we describe the features of the animation database and the operations that help in reusing for creating new animation sequences. In Sec. 3, we discuss the design and implementation GUIs that help to query animation databases and to generate new animation sequences. In Sec. 4, we present the implementation of an animation database system supporting model and motion reuse with the visualization features presented in Sec. 3. We outline related research efforts in Sec. 5 before concluding our paper in Sec. 6.

2. Animation Database Features and Operation Set

Animation sequences involve animation objects, which are referred to as geometric models. A scene graph is a hierarchical structure to describe a geometric model. In [1], we augment the scene graph model to represent an animation sequence of a specific geometric model by defining a new node, called *Interpolator Node*, for the motion, e.g., a walking sequence for a human body model. A relational database approach for representing and storing augmented scene graph has been adopted. The database design, based on the Entity-Relationship diagram, is such that the database can be indexed by propagating the metadata of the objects upward. In this way, a scene would have all the metadata of its children (models and motion). The search would be conducted first in the scene level and will go down to the object level only if a match is found.

Fig. 4. An animation reuse example applying a walking sequence of a woman [6] to a man [24].

A set of manipulation operations has also been defined using the animations which can be modified. These operations manipulate either the spatial characteristics of the augmented scene graph or the motion characteristics. The operations can be broadly classified into three categories: spatial, temporal, and motion adjustment. The augmented scene graph is used as an intermediate structure providing a consistent framework for the approach. Below, we show some of the operation statements with the help of an example. Figure 4 shows a snapshot of the animation generated.

INSERT *Andy* TO *Scene* PARENT *room* WHEN [6, 12] SAVE AS *Andy_in_room*
GET *walking* FROM *Nancy* SAVE AS *walking1*
USE *walking1* TO *Andy_in_room.Andy*
CROP *Andy.walking1* BY 50
JOIN *Nancy.walking* WITH *translation* (0, 0, 20)

Spatial operations on animations involve changing the position, size, and orientation of models. In our implementation, they are *INSERT*, *DELETE*, *EXTRACT*, and *EDIT*. The operations that manipulate motion are *USE*, *GET*, *JOIN*, and *DISREGARD*. Another temporal operation, PROJECT, facilitates the projection of a portion of an animation. In most cases, when motion of a model is reused to another model, it is necessary to adjust the motion to the new scene. Motion is usually very specific and hence it is desirable to have a set of operations to help the user alter the motion according to the new scene. These operations can be used in various combinations to alter the existing animations and generate the animation sequences required by the user. An example containing these operations is given in Sec. 3.3.9. The details of these operations can be found in [1].

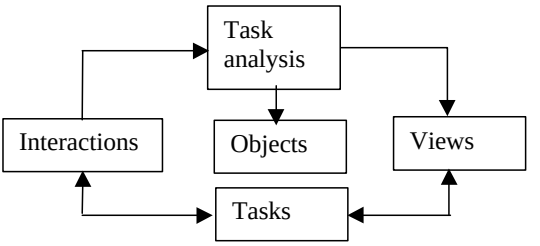


Fig. 5. Phases of the OVID model.

3. Design and Implementation of Graphical User Interfaces

In the previous section, we have discussed the operations for animation production. The operation syntax is quite similar to the popular query language SQL. We use the OVID (Object, View, and Interaction Design) methodology [18] to describe our design and implementation since our system is object-oriented. The OVID addresses the steps of interface design by analysis of the tasks that will be carried out using the interface. An *object model* is created which includes the descriptions of all objects that users will use to perform their tasks, the properties of the objects and the interactions between them. The model is presented in the form of views. Each view implements a subset of the tasks to be performed. Users utilize input/output (I/O) mechanisms to interact with the views and carry out the tasks. Figure 5 depicts the different phases of the process. The arrows indicate the transfer of information between the phases.

Employing this model, several GUIs have been designed to use the operations more efficiently and easily. These interfaces facilitate the reuse of animation sequence and the visualization feedback.

3.1. *Task analysis*

The first step, the task analysis, in the development is identification of the tasks that will be carried out. For the animation reuse system, the task list is given in Table 1 below:

3.2. *Object model*

From the above task list, five objects are identified: the user, scene graph, motion, model and database.

User:	The person using the system
Animation database:	The storage for the animation
Scene Graph:	The hierarchical structure to describe an animation
Model:	The spatial properties of an animation
Motion:	The temporal properties of an animation

The database contains one or more scene graphs of motions and models. They can

Table 1. The task list.

1.	Create new scene	Open scene graph to create new scene
2.	Query model	Search for model in the database according to specification
3.	Select model	Choose the model to be inserted
4.	Insert model	Insert model into scene graph
5.	Delete model	Delete model from scene graph
6.	Extract model	Extract model from one scene to insert into another
7.	Change position of model	Edit the position properties of model
8.	Change orientation of	Edit the orientation properties of model
9.	Change size of model	Edit the size of model
10.	Query motion	Search for motion in the database according to specification
11.	Select motion	Choose the motion to be used
12.	Use motion	Apply the motion to a model in the scene graph
13.	Get motion	Extract motion from one scene/model to use to another
14.	Disregard motion	Delete the motion interpolators
15.	Project motion	Use only specified duration of complete animation
16.	Join motion	Combine to motions to use on a single model
17.	Reduce duration of motion	Crop the duration of motion
18.	Increase duration of motion	Replicate motion a certain number of time to get longer
19.	Change speed of motion	Change speed of motion
20.	Retarget motion	Target motion to new model
21.	Open scene	Open the scene graph of an existing scene
22.	Generate VRML	Generate the VRML Text for animation
23.	Save scene to file	Save the VRML Text to a file
24.	Save scene to database	Save the animation as a record in database

be part of one or more scene graphs. The motions belong to models. This object model is presented in Fig. 6.

3.3. *Views and GUIs*

After the development of the object model, the views with more detailed object models, which include the views, are developed. The model with the motion adjustment view and the query view is given in Fig. 7.

Views are identified to accommodate key groupings of tasks and to embody key relationships between objects such that each view performs tasks leading to a specific outcome. For this system, seven views are identified: *Query, Scene Graph, Motion Adjustment, Timeline, Retarget Motion, Motion Mapping, and Model Metadata Views.*

3.3.1. *Query view*

The query view, shown in Fig. 8, is an integral part of the system. Through this view, the user is able to interact with the database in order to get required models

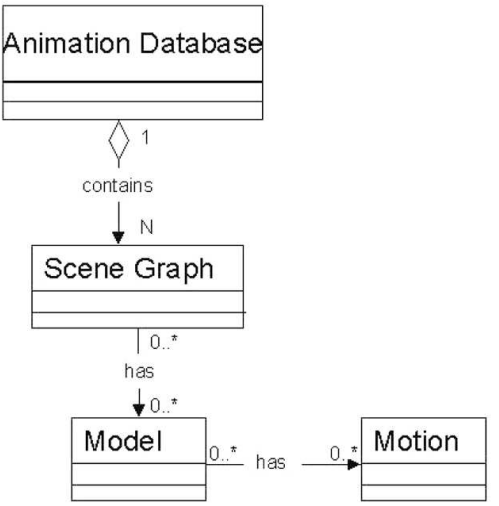


Fig. 6. Object model for the system.

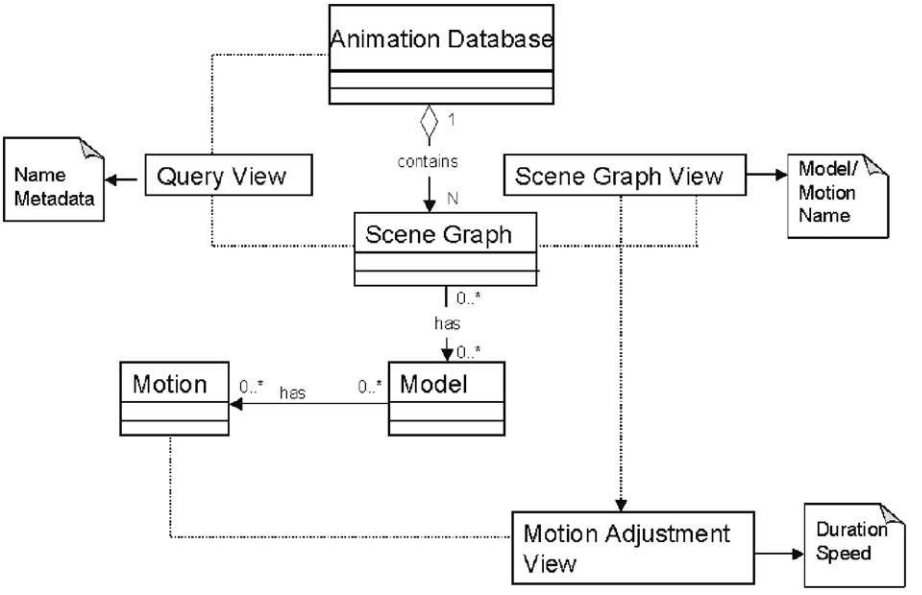


Fig. 7. Detailed object model with views.

and motions. Within the query view there are two views: one for model query and the other for motion query.

The Query GUI: It is the starting point of all animation reuse process. It contains an ActiveX object Web Browser that invokes the default web browser of the system

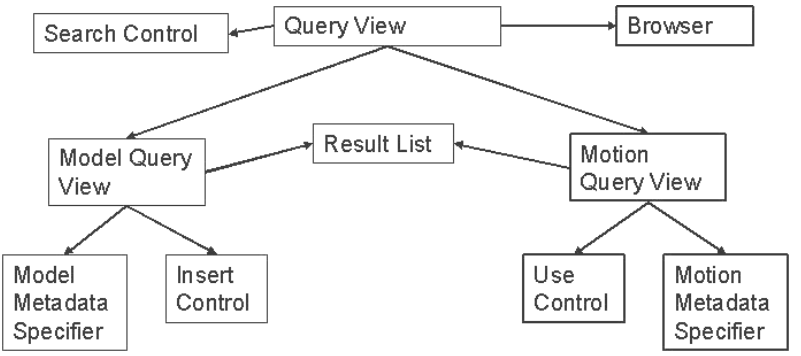


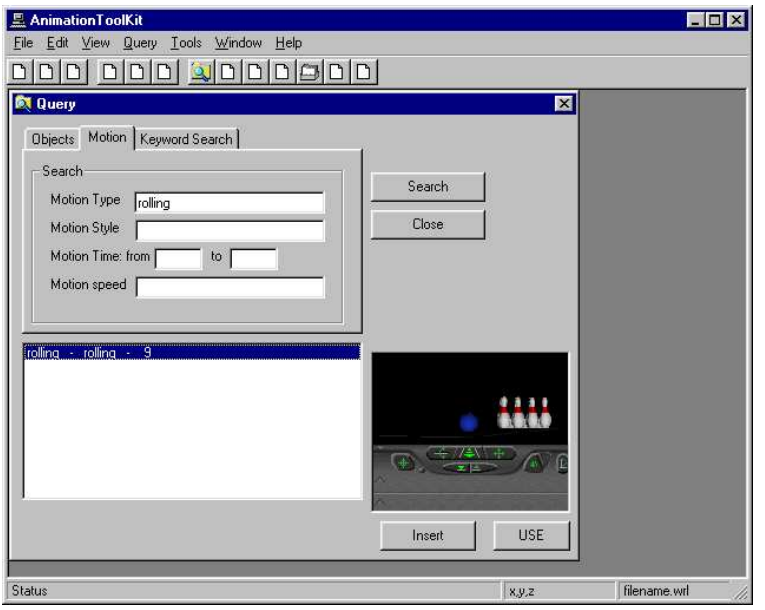
Fig. 8. Query view.

with the installed VRML browser, such as Cosmo Player or MS VRML viewer. The interface also displays a ranked list of query results. Users can query for models or motions by specifying one or more of the metadata requirements. The search for the animation objects is automatically converted to an SQL-like statement to retrieve the best matching objects from the database. Retrieved objects are shown as a list. By clicking on an object the user selects it and can view it in the browser. Figure 9(a) depicts the selection of a rolling motion after the search. Once the user finds an object, s/he can INSERT (for models) or USE (for motion) it in the scene graph. For instance, Insert operation on an animation scene will be carried out using the pseudo-code listed in Fig. 9(b).

3.3.2. *Scene graph view*

The motions and models inserted from the query view are displayed in the scene graph view (Fig. 10). From the scene graph view, the properties of the motions and models can be viewed and changed using other views. VRML text can also be generated through this view.

The Scene Graph GUI: The scene graph can be activated directly by selecting a new file from the menu or via the query GUI. The scene graph is the anchoring point of reuse. All operations other than USE and INSERT are carried out, directly or indirectly, via this interface. Depending on the object and type of interaction with this interface, different views will pop up; these views cater to the various modification requirements. Users can DELETE or EXTRACT models and DISREGARD or GET motions from the scene graph. These interactions will lead to the generation of the respective operation command. The user can also explicitly tell the VRML Text Generator to generate the VRML text using this interface. Figure 11(a) shows this window in detail and Fig. 11(b) lists the pseudo-code of generating the VRML text from the scene graph.



(a)

```
Procedure INSERT(ObjectID as Integer) {
   Open Database for reading
   Query the database table Object
      Inner Joined with VRMLTEXT through SQL for the ObjectID

   Create a temporary scene graph node
   Read from the database record the necessary record fields
     to fill in the attributes and metadata required by the node:
     ObjectID, Category, ID, Name, Type, Size, Position, Color and
     VRMLText
   Parse through the current scene graph

   If no duplicates are found Then
       Insert the temporary node into the scene graph
   Else
       Change the unique key of the temporary node and declare
           as a new instance
       Insert the modified temporary node
   End if
   Close Database
}
```

(b)

Fig. 9. Query (motion) GUI and pseudo-ode for inserting an object from the database into the scene graph.

3.3.3. *Model metadata view*

The model metadata view, Fig. 12, provides users with the capability to change the properties of the models. The changes can be previewed in a browser before they are applied.

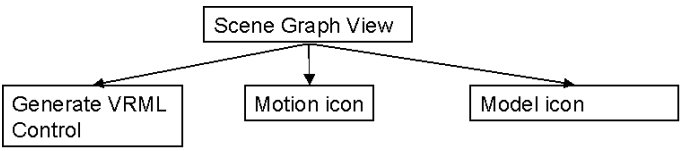


Fig. 10. Scene graph view.

Model Metadata GUI: It is activated when the user double clicks on a model icon in the scene graph. It generates the EDIT operation of the spatial operation set. Users can modify the spatial properties, size, orientation and position, of the models using this. Figure 13 shows this window in detail. For the example given, the EDIT statement generated after making the modifications and accepting them will be similar to the following:

EDIT *Dog_Wagging* POSITION (0, 0, 0) SIZE (1, 1, 1) ORIENTATION (0, 0, 0, 0) OF *Scene*

One or more of POSITION, SIZE and ORIENTATION options will be used depending on the parameters changed by the user. Changes made by the user will then be reflected in the VRML file. Users can preview the changes in the browser. Figure 14 also shows the *VRML Text* GUI which pops up when the generate VRML button is clicked. This GUI shows the VRML text that is generated for the scene.

3.3.4. *Motion adjustment view*

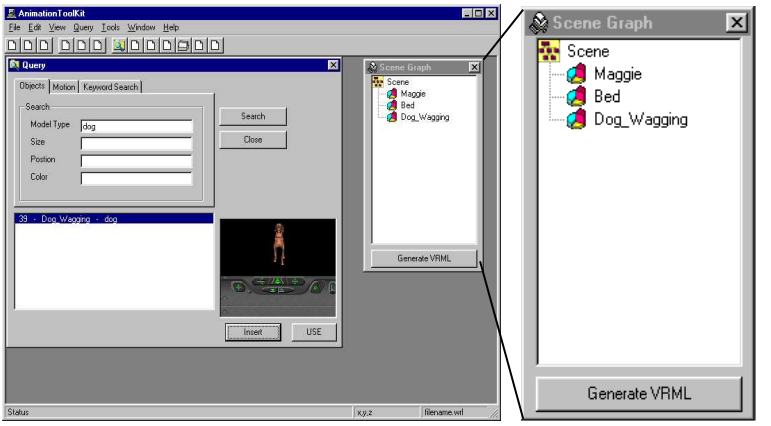
Similar to the model metadata view, the motion adjustment view allows users to alter the temporal properties of the motion; i.e., the duration and the speed. This view is shown in Fig. 14.

Motion Adjustment GUI: Similar to the Model Metadata GUI, this GUI is activated when the user double clicks on the motion icons. As the name suggests, this GUI caters to the operations in the Motion Adjustment operation set. This GUI supports the CROP, DUPLICATE and CHANGE SPEED operations. One or more SQL-like statements will be generated depending on the parameters to which changes have been made. These statements will modify the timing values associated with a motion in the VRML file. A detailed view of this interface can be seen in Fig. 15. The statements generated for this example will be:

CHANGE SPEED green_b*all.rolling* BY 2
DUPLICATE green_b*all.rolling* BY 2

3.3.5. *Motion retarget view*

The retarget motion view, Fig. 16, facilitates the retargeting of a motion to another scenario, in terms of position and/or model. The effect of the retargeting can be previewed in a browser before they are applied.



(a)

```
Procedure SceneGraph2File(Root as Node) {


    Declare PrototypeText, NodesText
            RouteText as String
    Declare AnimationEngine as String
        javascript node that controls motion times

     Traverse all the nodes of the scene graph by DFS
      For each node {

        If node.type = object then
           Append node.prototypetext to PrototypeText
           Append node.text including new translation,
                  scaling an orientation to NodesText
       Else if it is a motion
                 Append node.text NodesText
                 Append node.routetext to RouteText
                 Get temporal information and update AnimationEngine

      End if
    }
   Open File
   Save to File ( "#VRML V2.0 utf8" )
   Save to File ( PrototypeText + NodesText +
                  RouteText + AnimationEngine )
   Close File
}
```

(b)

Fig. 11. The scene graph GUI and the pseudo-code to convert the scene to file.

For interactive using inverse kinematics for motion retargeting, it is necessary to develop GUI with visualization of 3D models and the animation, including the original and resulting motion sequences. Using GUI the user can specify the articulated figure and new position/orientation for which the motion is retargeted. There are three major components of this GUI, visualization window, scene graph, and control

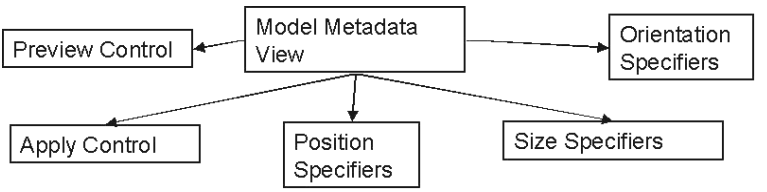


Fig. 12. Model metadata view.

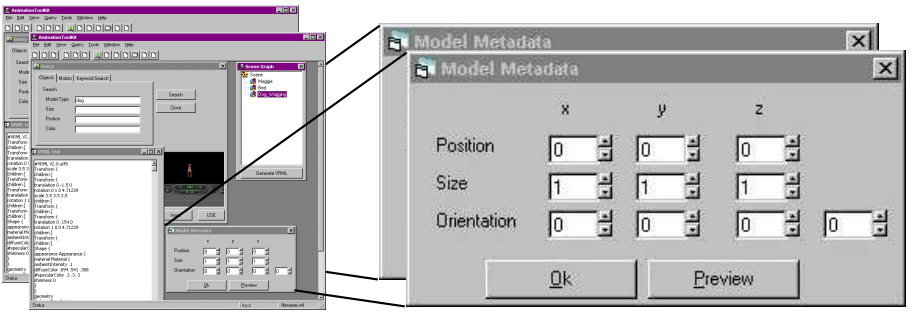


Fig. 13. Model metadata GUI.

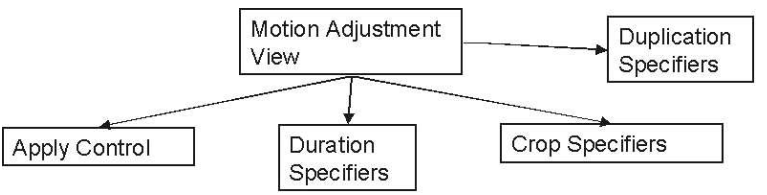


Fig. 14. Motion adjustment view.

panel. One example is shown in Fig. 17(a), where we retarget the kicking motion to the ball. From the control panel, the model (ball) and its position can be specified and used as the new target of the end effecter of the front left leg. The 3D model and animation are displayed in the visualization window, being able to be viewed from different viewpoints and zooming factors. In this way, users can intuitively adjust the motion sequences for reuse purpose. Once the result is satisfactory, it will be recorded and stored in the database. In Fig. 17(b), the pseudo-code is listed for motion mapping.

3.3.6. *Motion mapping view*

Using the motion mapping view seen in Fig. 18, the interpolators of one model-motion combination can be mapped to another one. The mapping can also be auto-assigned.

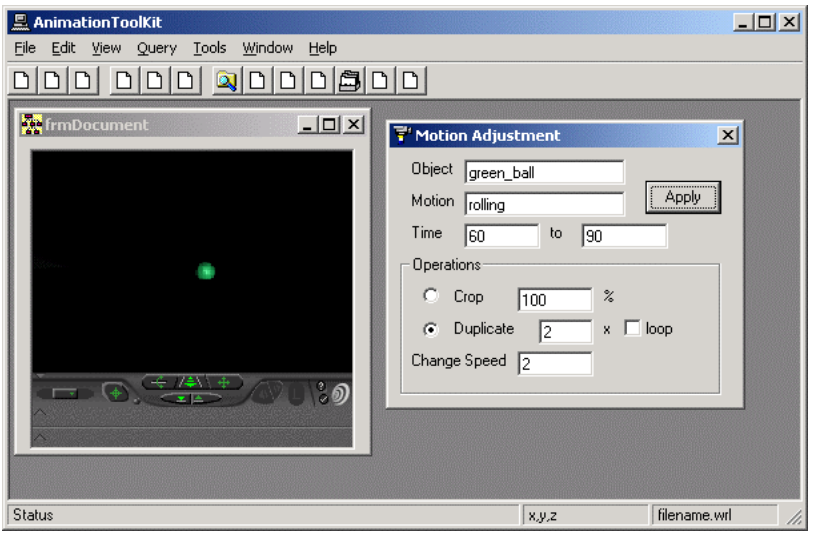


Fig. 15. Motion adjustment GUI.

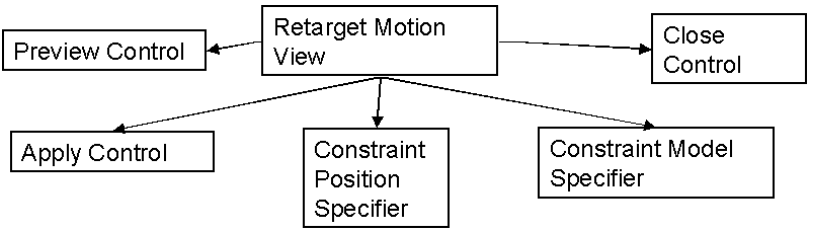


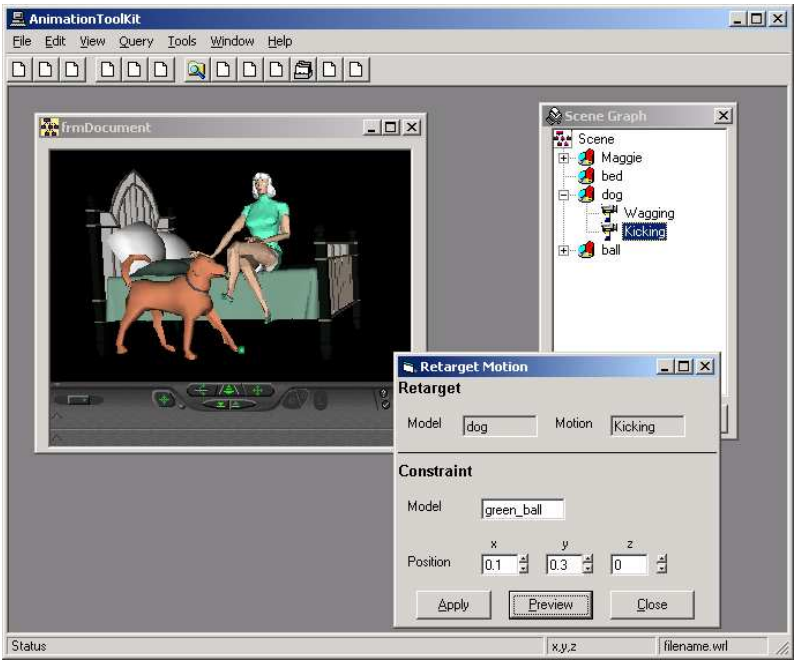
Fig. 16. Retarget motion view.

Motion Mapping GUI: This GUI is activated when a motion is used on a model other than the original model. Using this interface the interpolator nodes of one model can be linked to that of the other model. A sample *Motion Mapping* GUI is shown in Fig. 19. The pseudo-code for motion mapping is similar to Fig. 17(b). The USE statement generated after using and mapping the motion will be as follow:

USE *wiping* TO *Scene_Maggie_on_Bed_with_dog_ball.Maggie*
MAP *Barmaid.r_shoulderXOI.OrientationInterpolator* TO
Maggie.r_shoulderXOI.OrientationInterpolator

3.3.7. *Time line view*

The time line view, Fig. 20, allows the adjustment of the timing of the different motions. It helps in the generation of the project and join operations.



(a)

```
Procedure Use_Motion (MotionNode as SceneGraphNode,
                      ObjectNode as SceneGraphNode ) {
      Parse through the VRML Text Description of the Motion Node
      and extract the defined interpolator nodes and their type
      (Orientation or Position)
      Parse through the VRML Text Description of the Object Node
      and extract the defined joints/segments/nodes
      Using the GUI, a user can assign the correspondence
      between a joint and an interpolator node; otherwise, the
      default one will be applied
      Extract the Timer used in the MotionNode
      Clear the MotionNode.RouteText

      For every Joint in the Object Node {
          If Joint has an assigned Interpolator node Then
               MotionNode.RouteText= MotionNode.RouteText +
               "ROUTE " + TimerName +".fraction_changed TO" +
               Interpolator +".set_fraction

               If  Interpolator.Type = Orientation Then
                 MotionNode.RouteText = MotionNode.RouteText +
                 "ROUTE " + Interpolator +".value_changed TO" +
                 JointName +".set_rotation
               Else Type = Position
                 MotionNode.RouteText = MotionNode.RouteText +
                 "ROUTE " + Interpolator +".value_changed TO" +
                 JointName +".set_translation
               End if
      }
      Update Scene Graph Nodes
}
```

(b)

Fig. 17. Motion retargeting GUI and pseudo-code for motion mapping in VRML.

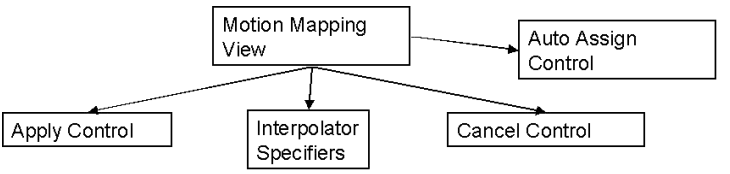


Fig. 18. Motion mapping view.

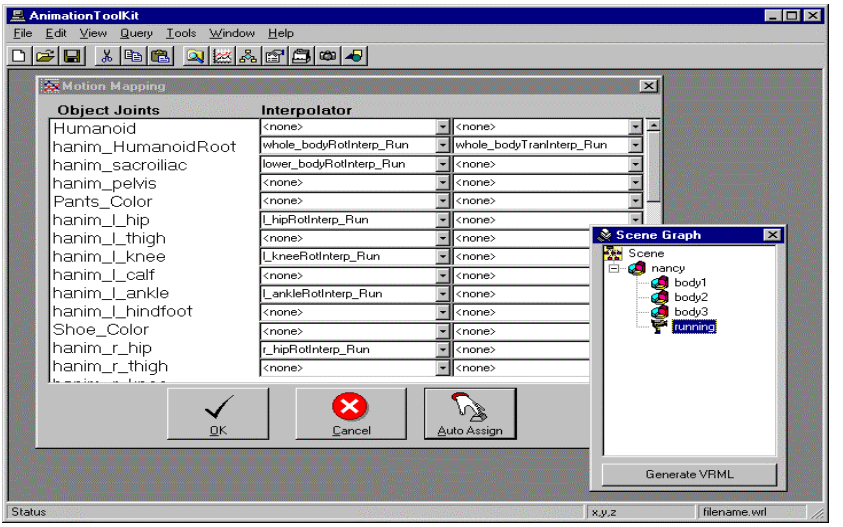


Fig. 19. Motion mapping GUI.

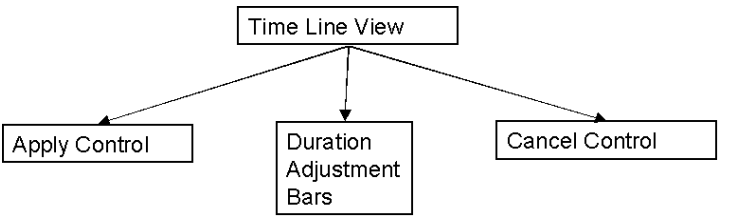


Fig. 20. Time line view.

Time Line GUI: Users can specify the temporal restrictions using this interface. For creating complex motions, more than one motion can be used for the same model. The time frame during which motions are applied can be changed using the *Time Line* GUI. The JOIN statement will be generated in the case when two motions are applied to the same model and the motions are overlapping in time. Figure 21 shows the *Time Line* GUI and depicts a scenario in which the following

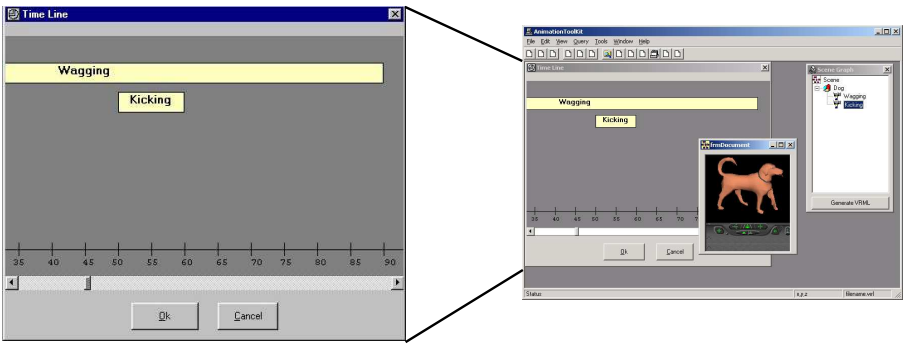


Fig. 21. Time line GUI.

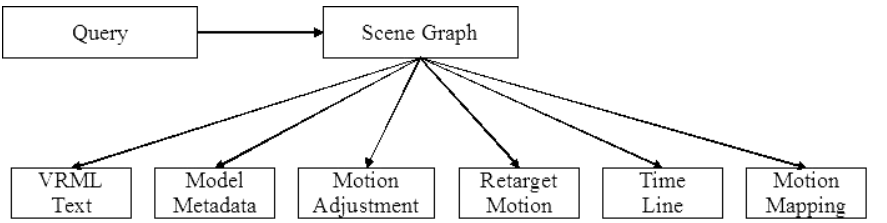


Fig. 22. Block diagram of GUI invocation sequence.

JOIN statement is generated:

JOIN *kicking* WITH *dog_wagging.wagging* WHEN (50, 60)

A block diagram to show the order in which a user can invoke these GUIs is given in Fig. 22.

3.3.8. *Other GUIs*

VRML Text GUI: This GUI displays the generated VRML Text to the user. It is activated when the Generate VRML button on the *Scene Graph* GUI is used. The user can only view the generated VRML but cannot interact physically with this interface. The user also has the menus and GUIs to save the new animations. These animations can be saved as VRML files from the *Files* menu as well as an element in the database using the *Database Management* GUI of the *Tools* menu. The Tools menu also provides a GUI in which the weights of the various metadata can be changed.

The Database Management GUI: This GUI provides the user with the functions for managing the database. The user can browse the database and add, delete and update animation objects in the database.

3.3.9. *Example*

Now, we will describe an example of using GUIs to produce an animation of a woman sitting on the bed and a dog kicking a ball. Human computer interaction and related GUIs are listed as follow.

Table 2. Summary of human computer interaction and related GUIs.

The Query GUI: The 3D models, i.e., a woman, a dog, a bed, a sofa, and a ball, etc. are retrieved together with their original motion sequences.
Model Metadata GUI: The position and orientation of 3D models are adjusted according to the new requirements.
Motion Adjustment GUI: The motion of the woman is adjusted from the original wiping to sitting on the bed. The motion of the dog is adjusted according to the new path specification.
Motion Retarget GUI: The running motion of the dog is retargeted to kick the ball at the end of the running.
Motion Adjustment GUI: The motion of the ball is generated and adjusted reacting to the kicking.
Time Line GUI: Time of each motion is adjusted for the consistency.

The snapshots of the scene along the human computer interaction are shown in Fig. 23, where the total number of GUIs invoked were 15, the user clicked about 46 times and changed 18 parameters.

The pseudo-code of the above example is listed in Table 3. It is clear that by human computer interaction using the visualization and GUIs greatly reduces the complexity for producing animation scene by avoiding to write the command list directly.

4. Implementation and Software Structure

For the animation reuse toolkit the system was designed in Visual Basic along with a database in MS-Access. The software structure of the Animation toolkit consists of five components. These are the Animation Database, VRML Text Generator, Scene Graph Generator, Operation Generator and the user interface system. Figure 25 gives an overview of the communications between the various elements in the system, followed by a brief description of these elements.

The Animation Database: This component is the database of animations in MS-Access 97. It contains the animations in the form of scene graphs along with the metadata to describe the content of the animations. By using GUI a user interacts with this component to retrieve the matching animation objects. The user can also save the new animations that are generated.

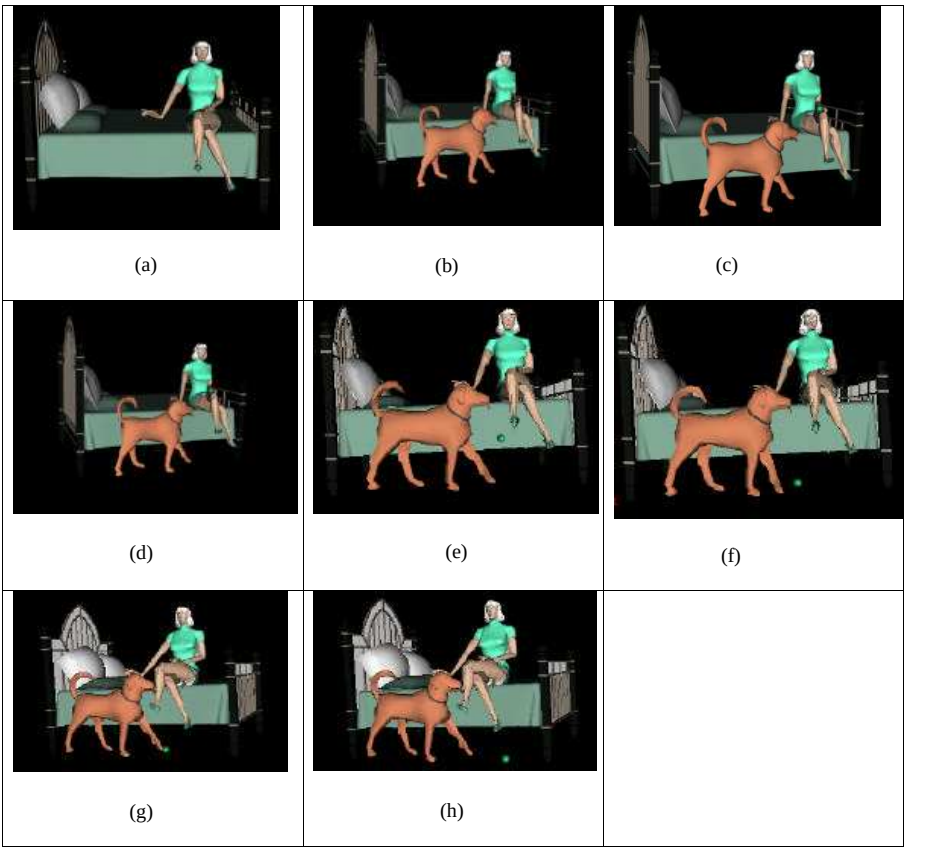


Fig. 23. The snapshots from a human computer interaction example to produce an animation scene.

The Operation Generator: This component generates the SQL-like operations according to the parameters which the user specifies through the various interfaces. Each time the user makes changes through the user interfaces of the *Modification process*, this component generates the corresponding operation sequence. The operation sequence generated is passed on to the *Scene Graph Generator*. Inverse kinematics is implemented in this component.

The Scene Graph Generator: This component generates the scene graph from the original scene and the operations generated. This process is executed in real-time with the modifications made to the animation. The scene graph can be saved to the database.

The VRML Text Generator: This component generates the VRML text from the scene graph. The text representing animation sequences, can be viewed through the browser and provides the user with continuous visual feedback. Such feedback is necessary to see the effect of the changes made and to let the user decide whether

Table 3. The SQL syntax commands automatically generated from human computer interactions.

1.	DELETE *sofa* FROM *Maggie_on_sofa* SAVE AS *Maggie*
2.	INSERT *bed* TO *Maggie* PARENT *ROOT* SAVE AS *Maggie_Bed*
3.	EDIT *bed* POSITION *(0, 0, 0)* SIZE *(1, 1, 1)* ORIENTATION *(1, 0, 0, 1)* OF *Maggie_Bed* SAVE AS *Maggie_on_Bed*
4.	INSERT *Dog_Wagging* TO *Maggie_on_Bed* PARENT *ROOT* SAVE AS *Maggie_on_Bed_dog*
5.	EDIT *Dog_Wagging* POSITION *(0, 0.1, 0)* SIZE *(0.6, 0.6, 0.6)* ORIENTATION *(1, 0, 0, 1)]* OF *Maggie_on_Bed_dog*
6.	EXTRACT *ball1* FROM *Croquet* SAVE AS *green_ball*
7.	INSERT *green_ball* TO *Maggie_on_Bed_dog* PARENT *ROOT* SAVE AS *Maggie_on_Bed_with_dog_ball*
8.	EDIT *green_ball* POSITION *(0, 0, 0.2)* OF *Maggie_on_Bed_with_dog_ball*
9.	GET *wiping* FROM *Barmaid_wiping_table.Barmaid* SAVE AS *petting*
10.	CROP *petting* BY *10* SAVE AS *petting*
11.	USE *petting* TO *Maggie_on_Bed_with_dog_ball.Maggie* MAP *Barmaid*`.r_shoulderXOI.OrientationInterpolator` TO *Maggie*`.r_shoulderXOI.OrientationInterpolator` SAVE AS *Maggie_pettingDog*
12.	RETARGET *Maggie_pettingDog.petting* TO *dog (0.1, 0.3, 0)* SAVE AS *Maggie_pettingDog.petting*
13.	GET *falling* FROM *Tennis.ball* SAVE AS *falling* USE *falling* TO *Maggie_ pettingDog.green_ball* SAVE AS *Maggie_pettingDog_fallingBall*
14.	GET *kicking* FROM *Dog_kicking.dog* SAVE AS *kicking*
15.	JOIN *kicking* WITH *Maggie_ pettingDog_fallingBall.dog.wagging* WHEN *(50, 60)* SAVE AS *Maggie_kickingDog.kicking_wagging*
16.	RETARGET *Maggie_kickingDog.dog.kicking_wagging* TO *green_ball (0.1, 0.3, 0)*
17.	GET *rolling* FROM *Bowling_Alley.ball* SAVE AS *rolling*
18.	USE *rolling* TO *Maggie_kickingDog.green_ball* WHEN *(60, 90)* SAVE AS *Maggie_kickingDog_rollingBall*
19.	JOIN *Maggie_kickingDog_rollingBall.green_ball.rolling* WITH *translation (0.6, 0.4, 0)*
20.	CHANGE SPEED *Maggie_kickingDog_rollingBall. green_ball.rolling* BY *2*
21.	DUPLICATE *Maggie_kickingDog_rollingBall. green_ball.rolling* BY *2*

the changes should be kept or not.

5. Related Work

Databases have been used in computer animation. In [1, 2], techniques and tools are developed to generate new animation sequences by reusing existing geometric

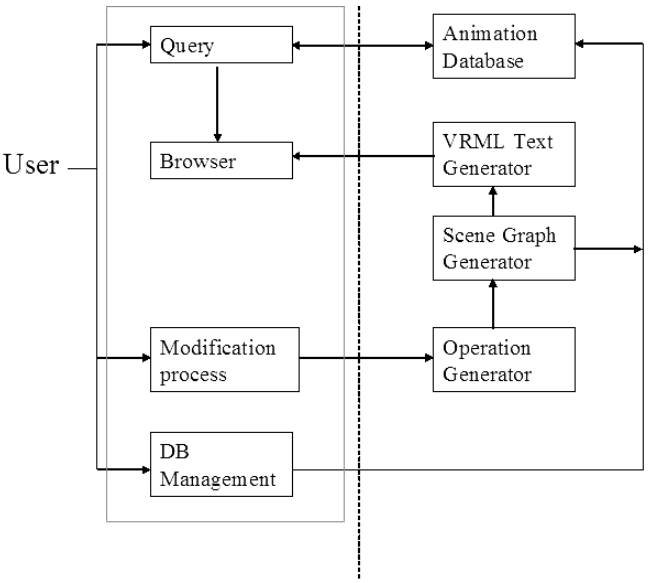


Fig. 24. Software structure of the animation reuse toolkit.

models and their motion sequences stored in animation databases. New animation sequences can be created through a series of operations that facilitate the searching of databases and manipulation of the properties of geometric models and motion information. Though these operations have syntax similar to that of the database query language SQL, they are still not intuitive to use. Other database approaches do not address the problem of motion reuse. They include [21], where an Informed Environment is proposed that creates a database dedicated to urban life simulation. Using a set of manipulation tools, the database permits integration of the urban knowledge in order to simulate more realistic behaviors. Another related work that makes use of databases in animation is presented in [14]. It uses a scene graph and an animated agent in multimedia presentations. Similarly, Ayadin *et al.* [5] have used databases based on divisions of the reachable space of a virtual actor to guide the grasping movement of virtual actors.

Motion editing refers to any method that adjusts the existing motion sequences constrained by new requirements. Among all motion editing method, *Inverse kinematics*, originally developed in robotics control [9], is the most popular method and it is also used in our implementation. Note that the motion editor is an independent module in our system. Thus, any motion editing methods [16, 11, 12, 8] can be adapted for our purposes.

Visualization has been attracting the interest of researchers and substantial amount of work has been reported in recent years to make data and processes visual in nature. One major reason for an increase in visualization is that it is easier to comprehend things as pictures rather than in the form of texts. Visualization has

been applied to several fields such as data analysis, training and learning systems. In [3], the inductive bias is derived for multi-dimensional clustering from visualization in which the user selects parameters that are difficult to specify automatically. Geroimenko *et al.* [10] developed an immersive collaborative environment in VRML to teach and learn VRML. Watson *et al.* [25] designed a generic computer-based Training Information System (TIS) and visualization of the data that it generates. Schroeder *et al.* [19] describe the object-oriented toolkit they developed for 3D graphics and visualization. They have defined a graphics model that has several types of objects, which capture the essential features of 3D models and a visualization model, based on the data-flow paradigm.

Graphical user interfaces (GUIs) are important in visualization that allows users to effectively perceive and express information. The advantages of GUIs are [20]:

(1) They are relatively easy to learn and use. Users with no computing experience can start to use it after a brief training session.
(2) The user has multiple screens (windows) for system interaction. Switching from one task to another is possible without losing sight of information generated during the first task.
(3) Fast, full-screen interaction is possible with immediate access to anywhere on the screen.

GUIs have been used in several domains including that of generating graphics themselves. One such work is that of Igarashi *et al.* [13], in which the gesture interfaces are extended for more general use. The proposed GUI allows users to give hints about the operation desired by selecting the related geometric components. The system then infers and returns the possible operations and returns them as thumbnails. The users perform the operation by clicking on the desired thumbnail. Arvo *et al.* [4] present a user interface for sketching which couples shape recognition and morphing. The input strokes are continuously morphed into pre-defined geometric shapes. Thus the users are apprised of the shape recognition process while being in control of the process.

6. Conclusions

Animations are an interesting data type that can help in creating interesting multimedia presentations. The main hurdle in using animations for multimedia presentations is that it requires a lot of skill and effort to produce good quality animations. A database approach helps in reusing models and motions to generate new animation sequences. However, this approach is quite complex in terms of using the various querying, model and motion manipulation operations. In this paper, we proposed a visually interactive method that can help users to handle the various animation database operations and generate new animation sequences. This method involves a set of GUIs that enable users to carry out complex animation generations in a simple manner while allowing them to visualize the changes being made.

References

1. Akanksha, Z. Huang, B. Prabhakaran, and C. R. Ruiz, Jr., "Reusing motions and models in animations", in *Proc. of EGMM 2001*, pp. 11–22. Also appeared in J. A. Jorge, N. M. Correia, H. Jones and M. B. Kamegai (eds.), *Multimedia 2001*, Springer-Verlag/Wien, 2002, pp. 21–32.
2. Akanksha, B. Prabhakaran, Z. Huang, C. R. Ruiz, Jr., "Animation toolkit based on database approach for reusing motions and models", preparing for submission.
3. M. Ankerst, M. M. Breunig, H. P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure", in *Proc. 1999 ACM SIGMOD Int. Conf. on Management of Data*, Philadelphia, PA, USA, 1999, pp. 49–60.
4. J. Arvo and K. Novins, "Fluid sketches: continuous recognition and morphing of simple hand-drawn shapes", in *Proc. ACM UIST '00*, 2000, pp. 73–80.
5. Y. Ayadin, H. Takahashi, and M. Nakajima, "Database guided animation of grasp movement for virtual actors", in *Proc. Multimedia Modeling '97*, 1997, pp. 213–225.
6. N. C. Ballreich, "3D Model. 3Name3D", http://www.ballreich.net/vrml/h-anim/nancy_h-anim.wrl. (1997).
7. R. Boulic, Z. Huang, N. Magnenat-Thalmann, and D. Thalmann, "Goal-oriented design and correction of articulated figure motion with the track system", *Journal of Computers & Graphics* **18**(4) (1994) 443–452.
8. A. Bruderlin and L. Williams, "Motion signal processing", in *Proc. ACM SIGGRAPH '95*, 1995, pp. 97–104.
9. K. S. Fu, Gonzalez, and C. S. G. Lee, *Robotics, Control, Sensing, Vision, and Intelligence*, McGraw-Hill, 1987, pp. 52–76, 84–102, and 111–112.
10. V. Geroimenko and M. Phillips, "Multi-user VRML environment for teaching VRML: Immersive collaborative learning", in *Proc. Information Visualization*, 1999.
11. M. Gleicher, "Retargeting motion for new characters", in *Proc. ACM SIGGRAPH '98*, 1998, pp. 33–42.
12. J. Hodgins and N. Pollard, "Adapting simulated behaviors for new characters", in *Proc. ACM SIGGRAPH '97*, Los Angeles, CA, 1997, pp. 153–162.
13. T. Igarashi and J. F. Hughes, "A suggestive interface for 3D drawing", in *Proc. ACM UIST '01*, 2001, pp. 173–181.
14. K. Kakizaki, "Generating the animation of a 3D agent from explanatory text", in *Proc. ACM MM '98*, 1998, pp. 139–144.
15. W. M. Lee and M. G. Lee, "An animation toolkit based on motion mapping", *IEEE Computer Graphics International*, 2000, pp. 11–17.
16. Z. Popovic and A. Witkin, "Physically based motion transformation", in *Proc. ACM SIGGRAPH '99*, 1999, pp. 11–19.
17. A. Reitemeyer, *Barmaid Bot*, http://www.geometrek.com/web3d/objects.html.
18. D. Roberts, D. Berry, S. Isensee, and J. Mullaly, *Designing for the user with Ovid: Bridging User Interface Design and Software Engineering*, Macmillan Technical Publishing, London, 1998.
19. W. J. Schroeder, K. M. Martin and W. E. Lorenson, "The design and implementation of an object-oriented toolkit for 3d graphics and visualization", *IEEE Visualization '96*, 1996, pp. 93–100.
20. I. Sommerville, *Software Engineering*, Addison-Wesley, Fifth Edition, 1996, pp. 319–344.
21. D. Thalmann, N. Farenc and R. Boulic, "Virtual human life simulation and database: Why and how", in *Proc. Int. Symp. on Database Applications in Non-Traditional Environments* (*DANTE'99*), IEEE CS Press, 1999.

22. D. Tolani, A. Goswami, and N. Badler, "Real-time inverse kinematics techniques for anthropomorphic limbs", *Graphical Models* **62**(5) (2000) 353–388.
23. The VRML Consortium Incorporated, *The Virtual Reality Modeling Language*, http://www.vrml.org/Specifications/VRML97/. International Standard ISO/IEC 14772-1 (1997).
24. Vcom3D, Inc., *Seamless Solutions*, Andy-H-Anim Working Group, 1998. http://www.seamless-solutions.com/html/ animation/humanoid_animation.htm.
25. J. Watson, D. Taylor and S. Lockwood, "Visualization of data from an intranet training systems using virtual reality modeling language (VRML)", in *Proc. Information Visualization*, 1999.