Animation Toolkit Based on a Database Approach for Reusing Motions and Models

AKANKSHA akanksha@comp.nus.edu.sg
School of Computing, National University of Singapore

ZHIYONG HUANG zyhuang@i2r.a-star.edu.sg
*A*STAR Institute for Infocomm Research (I^2R)*
School of Computing, National University of Singapore

B. PRABHAKARAN praba@utdallas.edu
Department of Computer Science, University of Texas at Dallas, USA

CONRADO R. RUIZ, JR ruizc@dlsu.edu.ph
College of Computer Studies, De La Salle University, Philippines

Editor:

Abstract: As animations become more readily available, simultaneously the complexity of creating animations has also increased. In this paper, we address the issue by describing an animation toolkit based on a database approach for reusing geometric animation models and their motion sequences. The aim of our approach is to create a framework aimed for novice animators. Here, we use an alternative notion of a VRML scene graph to describe a geometric model, specifically intended for reuse. We represent this scene graph model as a relational database. A set of spatial, temporal, and motion operations are then used to manipulate the models and motions in an animation database. Spatial operations help in inserting/deleting geometric models in a new animation scene. Temporal and motion operations help in generating animation sequences in a variety of ways. For instance, motion information of one geometric model can be applied to another model or a motion sequence can be retargeted to meet additional constraints (e.g., wiping action on a table can be retargeted with constraints that reduce the size of the table). We present the design and implementation of this toolkit along with several interesting examples of animation sequences that can be generated using this toolkit.

Keywords: animation, toolkit, database, metadata, multimedia presentations, reuse

1. Introduction

Animation, an important type of media, is increasingly being used in numerous applications, such as games, movies, advertisements, and multimedia presentations. However, users that require animations in their multimedia presentations may find it difficult to produce good animation sequences. Computer animations have two main components, the geometric models and motion sequences. Creating motion of good quality requires considerable effort of skilled animators, actors, and engineers. Reusing animations becomes an appealing problem. Generally, animation is not reusable [12]. Motion for a particular model is very specific, and is unusable for other models and scenarios. For instance, the motion of a person picking up a glass will be precisely that - it will be difficult to reuse the motion for having the person to pick up other objects from the ground or for a different person to pick up the same glass. Thus, making motion reuse as the main hindrance in reusing animations in general.

Computer animation research has developed four general strategies to the problem of producing motion. The first one is to improve the tools used for key frames in animation. Using this method, users can interactively specify and edit the key frames, then, the computer generates the in-between frames automatically using different interpolation techniques. The second one uses procedural methods to generate motions based on descriptions of goals. The third one tracks motion of real world actors or objects. The fourth one attempts to adapt existing motion generated by some other methods. The fourth approach could put animation capabilities in the hands of inexperienced animators and allow the use of motion created by others in new scenarios and with other characters. It can also enable “on-the-fly” adaptation of motions. One technique to motion

adaptation, presented by Bruderlin and Williams [8], treats motions as signals and applies some traditional signal processing to adapt them, while preserving the flavor of the original motion. A variant of their interesting method, motion-displacement mapping, was simultaneously introduced as "motion warping" by Popovic and Witkin [24]. Two other examples of such works are presented by Gleicher [12] and Hodgins [14]. All these techniques addressed the problem of reusing one animation sequence of specific types and can be used for manipulating the low level, i.e., the motion sequence level.

There have been significant strides in computer graphics technology that have already produced numerous 3D models that are readily accessible like those found in the web. There have been some researches that have attempted to store this information in relation databases and object-oriented databases. A similarity based measure is then an evident requirement. There a few researches that have develop similarity metrics based on shape.

1.1 Proposed Approach

In this paper, we address the issue of generating new animation sequences based on existing animation models and motion sequences. For instance, we may have two different animations where a woman is walking and another where man is waving his hands. We can reuse the walking and waving motion, and generate a new animation of a walking man with a waving hand. We propose a simplified scene graph based database approach for this purpose of reusing models and motions. The simplified scene graph focuses on the information relevant for reuse, such as object hierarchies, and disregards easily reconstructed data such as for translation and rotation. For this simplified scene graph based animation database, we provide a set of spatial, temporal, and motion adjustment operations. These operations include reusing existing models in a new scene graph, applying motion of a model to another one, and retargetting motion sequence of a model to meet new constraints. Using the simplified scene graph model and the proposed operations, we have developed an animation toolkit that helps users to generate new animation sequences based on existing models and motion sequences. This toolkit provides a set of Graphic User Interfaces (GUIs) that help users to manipulate scene graph structures, apply motion sequences to models, and to edit motion as well as model features. We use existing solutions in the literature for applying motion sequences to different models and for retargetting motion sequences to meet different constraints. The current implementation of our animation toolkit uses the motion mapping technique proposed in [19] for applying motion of a model to a different model. It (the toolkit) uses the inverse kinematics technique proposed in [31] for retargetting motion sequences to meet new constraints.

Contributions: The novel idea of the paper is a framework for creating animations by investing heavily on reusing models and motion that are stored in databases, rather than on the interactive skills of the user starting modeling and animation from scratch. The contributions of this paper are as follows:

- Simplified scene graphs as a data structure for representing an animation sequence of a specific geometric model.
- A database representation based on the simplified scene graphs, to help in reusing motion information as well as geometric models.
- A set of operations that manipulate models and motion sequences in the animation database of the simplified scene graph models.
- An animation toolkit that uses the proposed database approach. This toolkit provides an exhaustive set of GUIs that allows users to generate new animation sequences from existing ones without having to remember the syntax of operations to reuse animations.

Organization of the paper: We first describe the simplified scene graph model (Section 2). Then, we look at the database structure and similarity measures (Section 3). Next, we present the operations for reusing animations (Section 4). Then, we discuss the development of the

animation toolkit (Section 5) and show some animation examples (Section 6). Finally, we summarize the related work (Section 7) and conclude the paper (Section 8).

Some implementation details are given in the Appendix: database tables and relationships (Appendix A), the pseudo-codes of the scene graph data type declaration and for generating a VRML file from a scene graph (Appendix B).

2. Scene Graph Model

Animation sequences involve an animation object or a geometric model, as it is called. A scene graph is a hierarchical structure to describe a geometric model. A scene graph is a data structure that stores complete information about all objects and behaviors in a scene and all other entities that affect the objects: lights, sounds, background, and views. Scene graph was originally designed for real-time 3D graphics system Performer [29]. The structure of the Performer scene graph is a tree structure which permits shared instancing. Each node in the tree is an object or a motion. Typically, leaves in the scene graph represent geometry objects or motion and interior nodes represent transformation objects. It also contains data about the objects in a scene, such as their shape, size, color, and position. Each piece of information is stored as a node in the scene graph. A subtree in the scene graph may represent an object in the scene (the object itself may be made of subcomponents). If multiple copies of that object are going to populate the scene, then it is efficient to reuse the same subtree in different parts of the graph. Also, any change in that subtree will be reflected in all instances of it.

The Virtual Reality Modeling Language (VRML) scene graph is a directed acyclic graph. In VRML, *Node* statements can contain *SFNode* (the single field node for scalars) or *MFNode* (the multifieldnode for vectors) with field statements that, in turn, contain node (or USE) statements. Each arc in the graph from *A* to *B* means that node *A* has an SFNode or MFNode field whose value directly contains node B. One example of a VRML scene graph is shown in Figure 1.

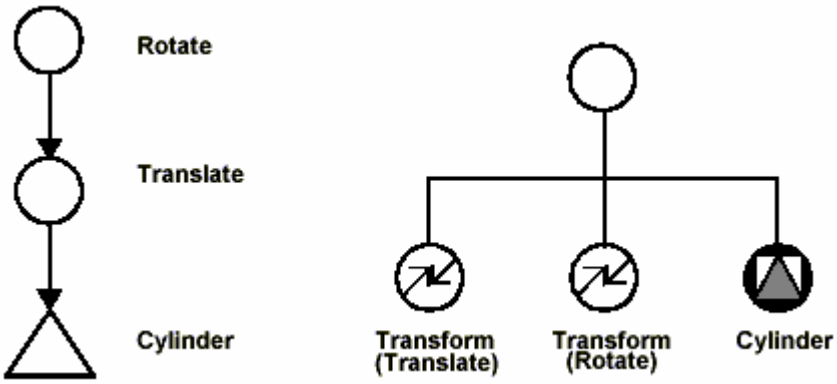

Figure 1. An example of a VRML scene graph sub-tree.

Two scene graph models are both possible and composable. Each matches the abstract scene graph definition in the VRML 97 specification, and each is isomorphic with the other.[9]. Most scene graphs are not actual implementations of the VRML parser and renderer. The scene graphs basically help animators or programmers visualize the hierarchies in a scene. In our case, our main intention is to reuse animation. As such, our scene graph is a simplified version, focusing on the object shapes and hierarchies. In our implementation, rendering information, such as position, orientation, and size, is not stored as separate nodes but part of the object node itself.

In VRML 97, the Interpolator node was introduced, this node specifies the motion. An *Interpolator* node is associated with a geometric model of multiple degrees of freedom (DOFs), e.g., a rigid 3D object has 6 DOFs in 3D space, 3 for translation and 3 for rotation. An articulated figure such as a human body may have hundreds of DOFs. In an Interpolator node, key frames are used to represent the motion of the model. It is defined for all DOFs: key [k_1, k_2, ..., k_i, ..., k_m],

keyvalue [v_1, v_2, ..., v_i,..., v_m], where k_i is the *key frame number* or *key frame time*. The key frame number can be mapped to the time according to the different frame rate standards such as PAL, SECAM, and NTSC. v_i is a vector in the motion configuration space: $v_i=[u_{i,1}, u_{i,2}, ..., u_{i,j}, ..., u_{i,n}]$, where $u_{i,j}$ is a *key value* of DOF *j* (in displacement for translational DOFs and angle value for rotational DOFs) at k_i, and *n* is the number of DOFs of the model. One key and keyvalue together form one animation sequence for a model.

In most VRML scene graphs the Interpolator Node is inserted as a sibling to the geometric model it is connected to. It is even possible that it is not directly connected to the geometric model it is used for if it is used my multiple geometric models. In our scene graph definition we insert the motion as a child of the geometric model that uses it. This model is also intuitive for the user of the toolkit. One example of a scene graph is illustrated in Figure 2.

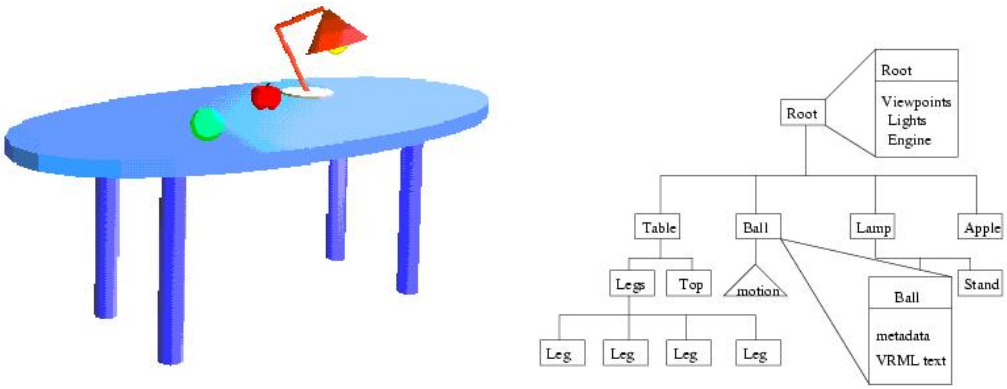

Figure 2. An animation sequence of a ball rolling on a table. The scene graph for this example decomposes the scene into a tree of its atomic objects.

3. Database Structure

There has been much effort in the VRML community to link databases and VRML [34]. Although there are ways of connecting databases to VRML and numerous groups in the Web3D consortium developing interfaces for VRML, there has only been a few research projects that store the VRML itself in databases. In this work, we do not explore visualizing databases though VRML, but how to store VRML in databases.

3.1 Existing Database Schemes

One research that stores VRML files in databases is that of [7]. In this method, the VRML files are parsed and saved into an object-oriented database, including all the links and node information. The database scheme is shown in Figure 3.

Another method was proposed by [13], which focused on motion. Their database is used for the management of a large number of motion data for humanoid virtual actors. The metadata of motion are manually annotated though automatic categorization is discussed. The E-R diagram of the database is shown in Figure 4.

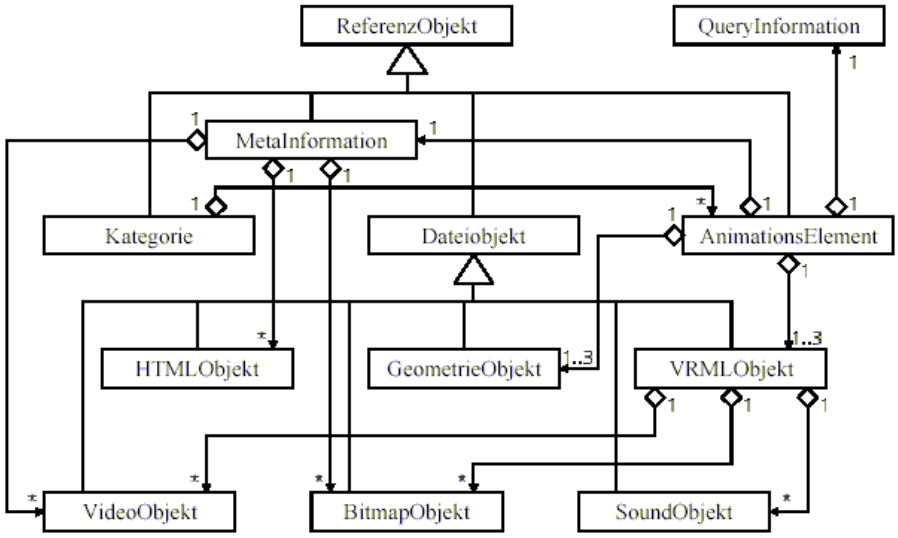

Figure 3. The object-oriented database scheme of Norbert et al. [7].

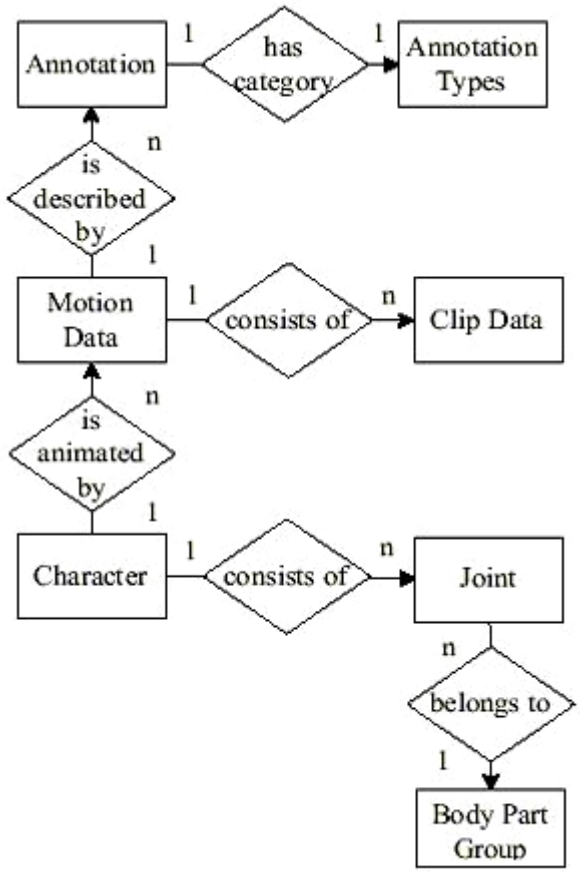

Figure 4. A Simplified Entity-Relation Diagram of Amoba et al. [13].

3.2 Scene Graph Database

In our case, we adopt a relational database approach for representing and storing simplified scene graphs. We do not include all the information like what was used in [7]. The fields of the database for the geometric model and the motion are shown in Table A-2 and Table A-3 respectively (in Appendix A). The geometric models and motions of each object are stored in two separate tables of the database and are linked by the Object ID being the primary key. The object table and the motion table are not the only tables in the database.

The scene also has its table in the database, which is linked to the object table by a Has-A relationship. The objects and scene database tables have an entry for metadata that provide descriptions about objects and scenes. These descriptions include the type of object (man, woman, cow, etc.), nature of object (color, size, shape), as well as descriptions of motions in scenes such as walking, running, etc.

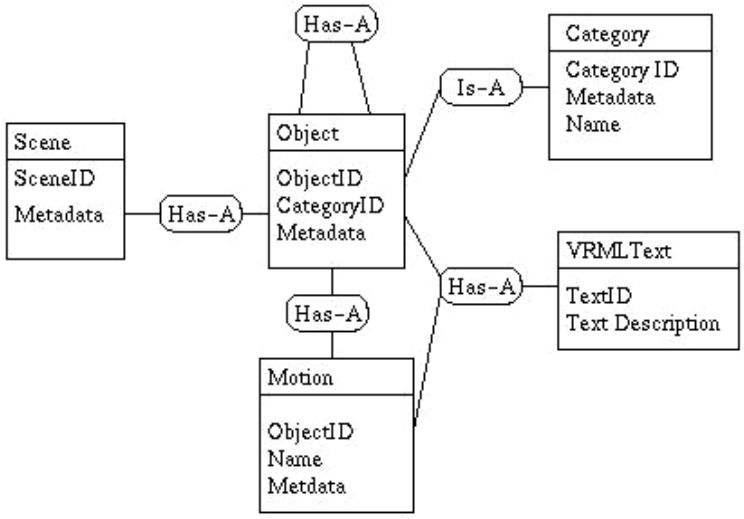

Figure 5. The ER diagram of RDBS for the scene graph

In the database, objects are also linked to an Object_Component table that links records on the object database to other objects if it is a complex object. A complex object is made up of simple objects (or atomic objects) with Has-A relationship. We also have a category class and each object belongs to a certain category. In our database the object table is linked by an Is-A relationship to the category table. Generally, relational databases are designed based on an Entity-Relation (ER) diagram, which is given in Figure 5.

The rendering information, such as position, orientation, and size, is included in the database. The information is regained or extrapolated to recreate an existing scene. In most cases, when an extracted model is used in another scene, the information usually needs to be adapted to fit the new scene, such as the example in section 6.2. In this case, it may be easier to disregard the existing position, orientation, and size and start from a predefined neutral and default position, orientation and size

3.3 Similarity Measure

The toolkit uses similarity measures to search for models and motions in the database. Similarity measures are used to quantify the resemblance between the query and the database objects. These measures are widely used multimedia systems, to search for media such as for images, audio and video. In our case, we treat animation as our media and implore a basic similarity measure. Similarity between animation A_i and query Q can be computed as the inner vector product:

$$sim(A_i, Q) = \sum_k (a_{ik} \bullet q_k)$$

where $\boldsymbol{a_{ik}} = 1$ if keyword k occurs in animation ***I;*** otherwise $\boldsymbol{a_{ik}} = 0$. $q_k = 1$ if keyword *k is* present in the keyword entered by the user; otherwise $q_k = 1$, otherwise. Since the vectors are binary the inner product represents the number of matched keywords in the animation. In the system we implore the Cosine similarity:

$$CosSim(D_i, Q) = \frac{\sum_{k=1}^{t} (d_{ik} \bullet q_k)}{\sqrt{\sum_{k=1}^{t} d_{ik}^{\,2} \bullet \sum_{k=1}^{t} q_k^{\,2}}}$$

since the vectors are binary we can then write it as:

$$\frac{|\, d_i \cap q_k \,|}{\sqrt{|\, di \,|} \times \sqrt{|\, qk \,|}}$$

There are works that explored creating a similarity based measure for 3D models and categorizing motion. In [10] a search engine was proposed for 3D models. In this method, Shape Histograms, 3D harmonics with a sketch based query are applied for 3D models. In [1], a 3D Shape Similarity Metric was proposed based on the Correspondence of Points. In [35], a content-based information retrieval was proposed for VRML 3D objects based on color and shape. Our system is general and all these approaches can be integrated.

The database is indexed by propagating the metadata of the objects upward, thus the scene would have all the metadata of its children objects and motion. The searching would first be done in the scene level. It will only go down to the object level if a match is found. Another alternative would be using categories as indices, where objects or scenes are also assigned to the categories. Currently, metadata of models and motion sequences in VRML files are manually annotated in the database.

4 Operations for Reusing Animations

After gathering all the models required for creating a new scene, the next step is to integrate them into the new scene and adjust their properties accordingly. To facilitate this step the animation toolkit supports a set of operations to manipulate models and their associated motion characteristics in a simplified scene graph model.

For instance, we can consider an operation that can *use* the walking motion of a man model to a woman model or *join* a walking motion and a (hand) waving motion to animate a walking man with a waving hand. These operations basically manipulate either the spatial characteristics of the simplified scene graph (by inserting/deleting models) or their motion characteristics. The operations can be broadly classified into three categories: spatial operations, temporal operations, and motion adjustment operations. The simplified scene graph is used as an intermediate structure providing a consistent framework for the approach.

4.1 Spatial Operations

Spatial operations on animations involve changing the position, size, and orientation of objects. The operations used to query and manipulate the spatial attributes of an animation, are as follows:

1. *Insert Operation*: adds an object into a scene under specified conditions. The user can include the new x, y, z coordinates of the object, and/or the time frame when the object will be shown. The object can be an atomic simple object or a complex object. The insert operation is accomplished by simply inserting the 3D model into the scene graph. The specific location of the node is determined by the user, by indicating the parent.

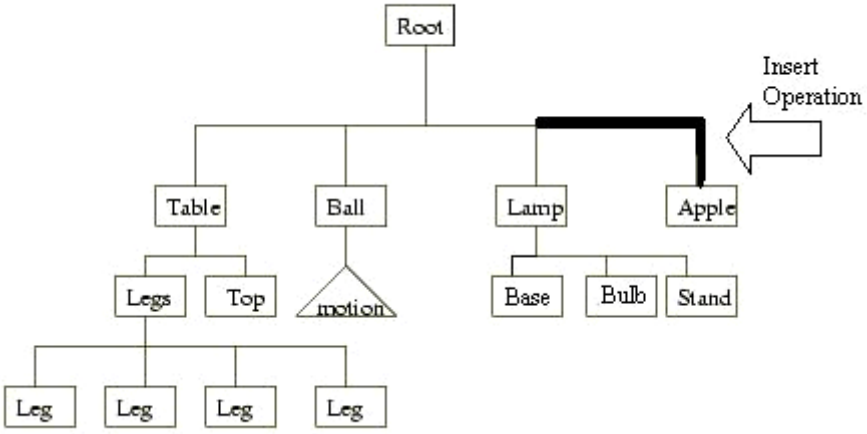

Figure 6. Insert operation on the scene graph.

2. *Delete Operation*: removes an object from a scene as shown in Figure 7. This can help the user eliminate unnecessary objects from a retrieved scene. For example, a user can query for a particular scene and remove the objects that the user may want to replace or completely take out of the animation.

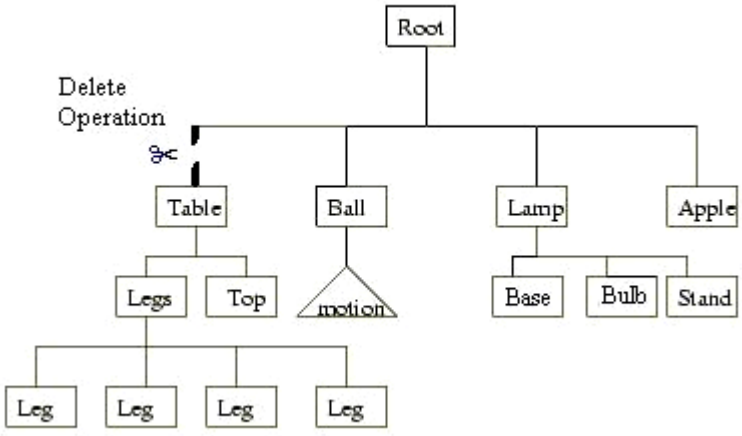

Figure 7. Delete operation on the scene graph.

3. *Extract Operation*: extracts an object from a scene. This operation is used when a scene or a complex object is the query result and the user only wishes to use a part of the result.

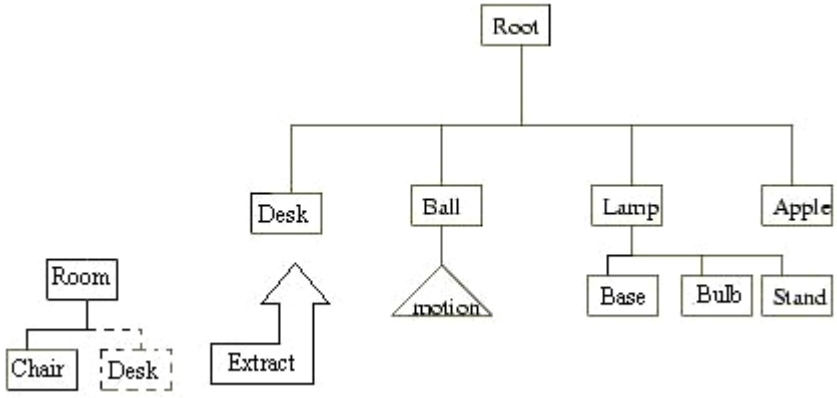

Figure 8. An object extracted and inserted into a new scene graph.

4. *Edit Operation*: modifies the metadata of the object. The user can edit the position, size, and orientation of the object. This operation directly manipulates the attributes of a user specified node in the scene graph.

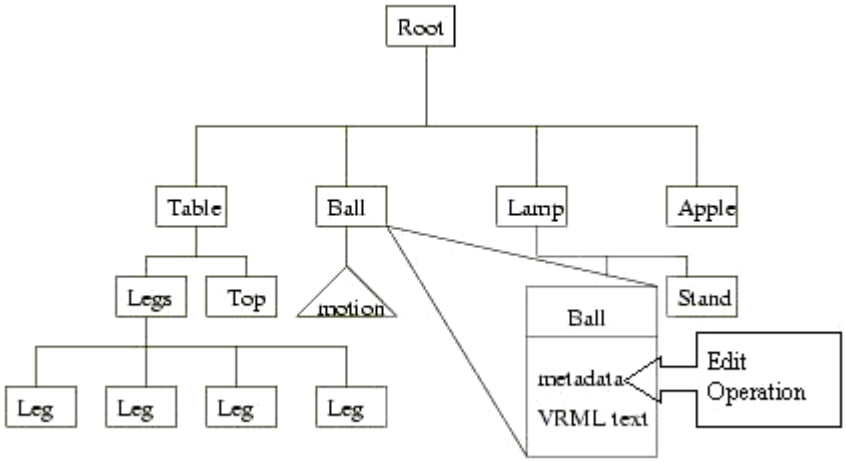

Figure 9. Edit operation on a node in the scene graph

4.2. Temporal Operations

Motion is an integral part of any animation. Temporal operations help in retargeting motion for new characters and scenes. The operations for manipulating motion are use, get, and disregard.

1. *Use Operation*: This operation utilizes a motion to a character. This paper has adapted the motion-mapping approach to re-use existing motion to other models. The user maps the interpolator points of the motion to the joints of the geometric model to be applied to. The use operation operates similar to the insert operation of the spatial operations. However, it requires that the parent is neither the root node nor another motion node. In cases of articulated figures and complex motion, reusing motion requires the mapping of the Interpolator nodes of the motion. Examples of how motion of one object is reused on other 3D models are given in Section 5.1.

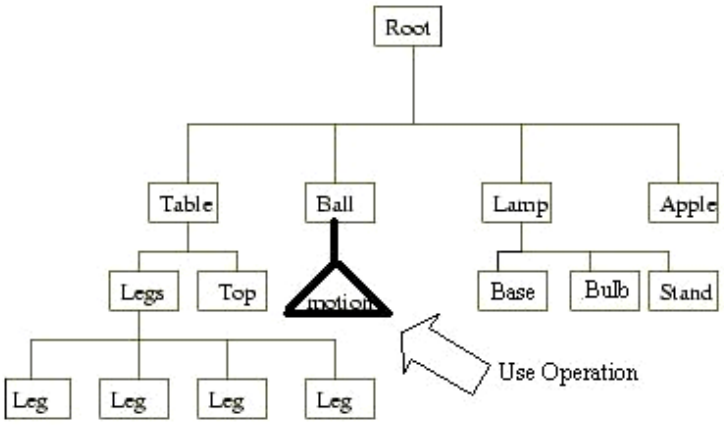

Figure 10. Use operation for inserting motion into animations

2. *Get Operation*: extracts motion from a character. In a scene graph, motion is defined by keyframe animation and by the nodes: OrientationInterpolator and PositionInterpolator, among others. The proposed approach searches for the interpolators that are linked to the specified model and copy the keyframes. This operation is similar to the extract operation. The get operation can be invoked on a complex object with a motion attached to it, and only the motion is returned to (as result of the operation) for the authoring of the new animation.

3. *Disregard Motion*: deletes the motion of an object. This is done by searching for the keyframes that are used on the object and by deleting them, including the timer node that was used.

4. *Project Operation*: projects a specific time interval from a specified animation based on the user's specified conditions. For example, if the user found an animation matching all the requirements, except for time constraints, s/he may opt to project a portion of the animation. The approach parses through the entire scene graph tree and updates all the start and end times of the motion nodes.

5. *Join Operation*: the operation allows the user to join two key frame sequences together. This provides the user with the ability to generate more complex motion from standard motion. The time base will be automatically adjusted. Figure 11 shows how two motion nodes are linked to one parent object.

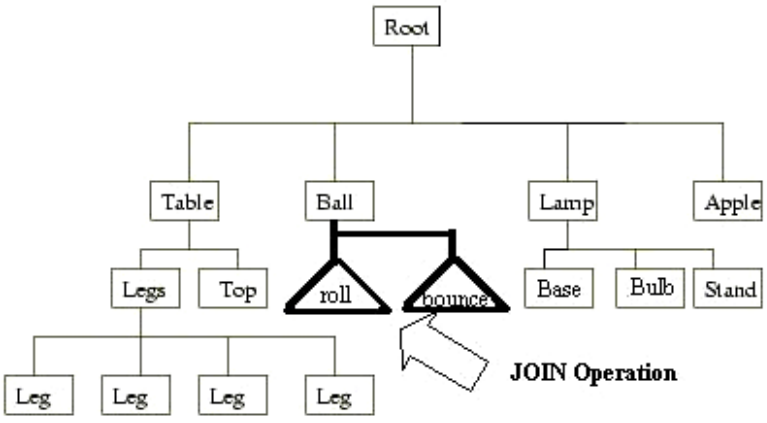

Figure 11. Join operation for combining two motions to one object.

4.3 Motion Adjustment Operations

In most cases, when a motion of an object is reused by another object, there is a need for adjusting the motion to the new scene. Motion is usually very specific and hence it is desirable to propose a set of operations to help the user fit the motion to the new scene.

1. *Crop Motion*: operation cuts the motion into a smaller motion sequence by deleting the keyframes and calculating the appropriate substitutions. The crop operation edits all the Interpolator nodes (described in Section 2.1) for a particular motion, based on the crop value α. For example, consider the motion node in a scene graph, *PositionInterpolator*, whose key frames are: key [k_1, k_2, ..., k_i, ..., k_m], keyvalue [v_1, v_2, ..., v_i, ..., v_m]. They will be changed to new key frames: key [k_1, k_2, ...,k_i, ..., α], keyvalue [v_1, v_2, ..., v_i, ..., v_m'], where the key value v_m' can be computed using the key frame interpolation if it is not an existing one. If the crop value α falls between two consequent keyframes k_i and k_{i+1}, the key frames from k_{i+1} to k_m will be deleted.

2. *Duplicate Motion*: operation duplicates the key frames to expand the duration of the motion. This helps if the user requires a longer animation than what is retrieved from the animation database. The duplicate operation adjusts all the Interpolator nodes for a particular motion, based on the duplicate value α. For example, given an Interpolator node in the scene graph whose key frames are key [k_1, ..., k_m], keyvalue [v_1, ..., v_m]. They will be changed to new key frames key [k_1, ..., k_m, k_{m+1}, ..., k_{2m}, ..., $k_{(\alpha-1)m+1}$, ..., $k_{\alpha m}$], keyvalue [v_1, ..., v_m, v_1, ..., v_m, ..., v_1, ..., v_m], i.e., the old key frames are duplicated α times.

3. *Change Speed*: operation changes the speed of the motion. It is implemented by scaling the key time base, i.e., the period represented in key [k_1, ..., k_m] of a PositionInterpolator node. So the duration of the motion will be changed. In order to keep the same duration, the new motion can be used in conjunction with the previous two operations.

4. *Retarget*: operation allows the user to adapt an existing motion sequence to meet the new constraints. Our current implementation uses inverse kinematics [31] for articulated 3D models. The position and orientation of the end effectors are specified automatically by the new constraints or interactively by users. The inverse kinematics algorithm can compute the new position and orientation values of other internal joints automatically in real time.

5. Animation Toolkit

An animation toolkit has been developed based on the proposed database approach using Visual Basic 6.0 (VB). Access 2000 was used as the primary database for the storage of the animations represented in scene graph. In our implementation, the scene graph is represented in Virtual Reality Markup Language (VRML) format. VRML is basically a text-based language used to model virtual reality systems and animations [33]. A VRML file consists of the following components:

1. **The File Header**: All VRML files must start with the following line: #VRML V2.0 utf8.
2. **The Scene Graph**: This part contains the nodes that describe the geometry and their properties, as well as nodes that participate in the event generation and routing mechanism.
3. Other nodes such as Prototypes and event routing.

The VRML file is decomposed into smaller VRML Text Descriptions, which represent individual objects and motions. The decomposition is based on the scene graph of the animation. These *Text Descriptions* are then stored in the VRMLText table of the database. Objects are linked to the table by a Has-A relationship which stored the prototypes and nodes used to represent the object. Motion is also linked to the table by a Has-A relation that contains the Interpolator nodes and Event Routing. One simple example of a scene graph representing an animated blue cube is shown in Figure 12, whose VRML description is shown in Figure 13.

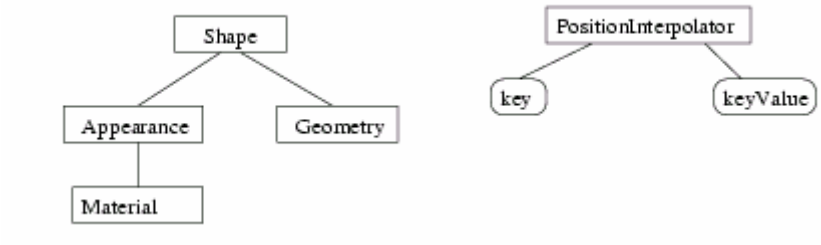

Figure 12. An example of scene graph of an animated blue cube. Three keyframes are defined in the PositionInterpolator node.

```
Shape {
   geometry Box {size 1 2 3}
   appearance Appearance (material Material (diffuseColor 0 0 1}}
}
PostionInterpolator {
   key [0.0 0.5 1.0]
   keyvalue [2.0 0.5 2.0, 2.0 0.5 3.0, 2.0 0.5 3.0]
}
```

Figure 13. The VRML description of the animated blue cube.

The scene graph can be rendered by Open GL Optimizer, IRIS Performer, and VRML browser. To parse and browse the animation the ActiveX object Web Browser has been used, which invokes the default web browser of the system with the installed VRML browser, such as Cosmo Player or MS VRML viewer. The abstract view of the system is shown in Figure 14.

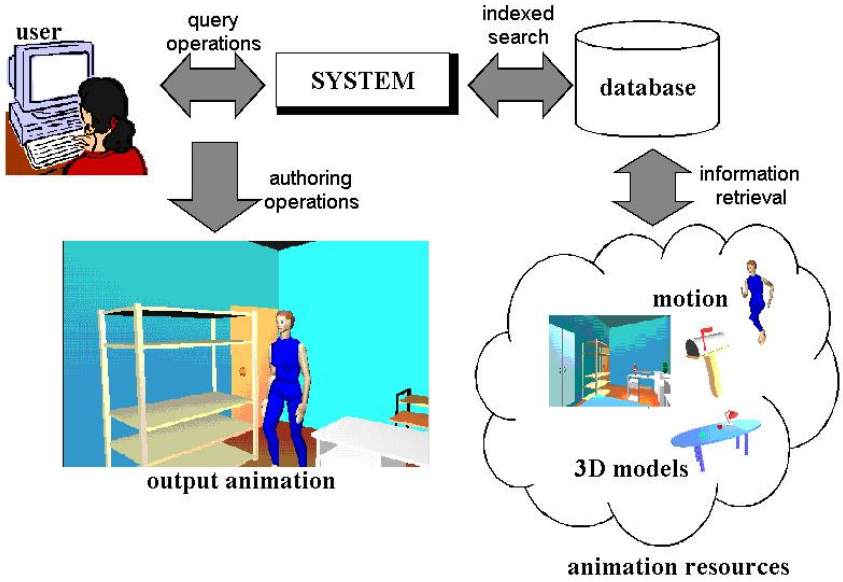

Figure 14. Abstracted view of the system. Animation is produced by using the operations to reuse existing models and motion sequences.

5.1 Toolkit Design and Implementation

The overall organization of the toolkit is shown in Figure 15. Animations are decomposed and stored in the database based on the scene graph structure. The scene graph is also used as an intermediate structure for managing the objects and motion that will be used for the new animation.

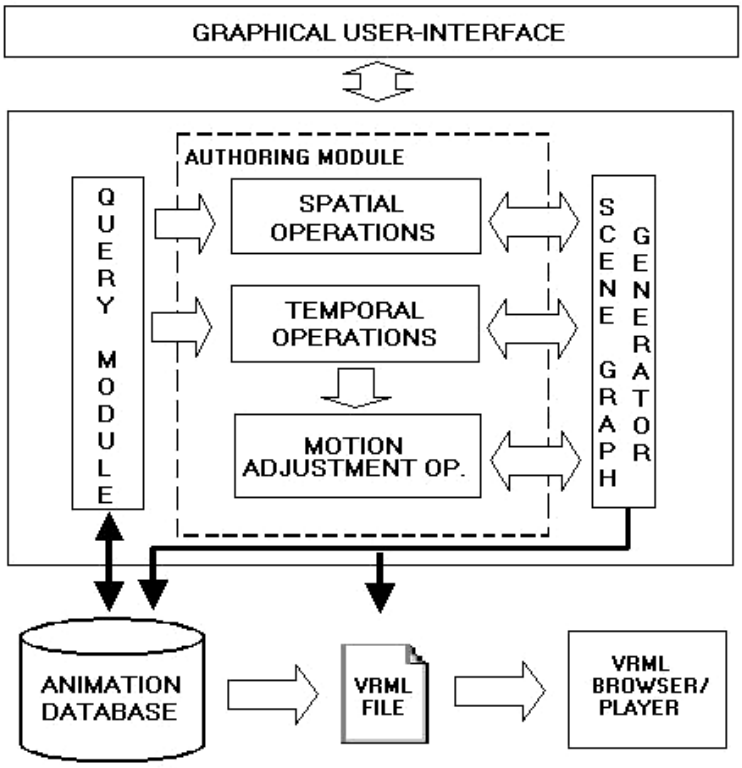

Figure 15. Toolkit organization.

The authoring of a new animation starts by collecting the necessary objects and motion in the scene. This is done by invoking the Query Module. The Query Module directly interacts with the database to retrieve the animation, object or motion the user has requested. The user interacts with the system through a Graphical User Interface (GUI). The user can manipulate the query results using the authoring operations. If the result is an object, it can be edited by the spatial operations before it is added to the new scene graph. If the result is a motion, it could be passed to the temporal operations to change time constraints. It can then be either sent to the motion adjustment operations for further editing or added directly to the scene graph. The user can at any time manipulate the nodes of the scene graph by using the authoring operations again.

Most of the motion adjustment operations are primarily text and string manipulations of the VRML files associated with the simplified scene graph models. The toolkit uses Visual Basic's Rich Text box component and its methods to search, cut, copy, and paste portions of the VRML text. Other operations manipulate the scene graph structure. The toolkit uses VB's TreeView Control Component to implement the scene graph structure, catering to functions such as delete, add and search. After editing the objects and the motion, the scene graph now reflects the desired animation of the user. The scene graph can either be stored in the database again or saved as a file. Viewing the animation also requires that the scene graph be first saved to a temporary file.

5.1.1 Data Structures

The scene graph is the most important data structure in the animation toolkit. Since the toolkit is implemented in visual basic, it is defined as a user-defined type shown in Figure B-1 (see Appendix B). The main structure that represents the hierarchical structure of the scene graph is VB's TreeView control object. The control object manages the insertion, deletion and editing of the nodes. The user-defined data structure extends the TreeView control object by incorporating more information needed by the nodes of the simplified scene graph. The data structure is a dynamic array of variants to accommodate either object nodes or motion nodes. The index of the array is cross-referenced with the index of the nodes in the TreeView Control Object.

The object and motion nodes have the following attributes: the *Name* which corresponds to the database Object Name or Motion Name also shown in the TreeView control; *ID* that is indicative of the unique ID assigned in the database; *Text* which is the actual VRML Description of the object or of the motion; and the is*Motion* attribute which is a Boolean data type to distinguish the type of a scene graph node. The Object type has additional attributes. These attributes are *Position* and *Size* which store the x, y, z coordinates or scaling factor of the object, they are both of Point Type. The Object Type also has the attribute Rotation that is of Rotation Type similar to Point Type except with the inclusion of the field theta to store the degree of rotation in radians. The Object Type also contains the Proto attribute that stores the actual VRML text description of the prototype used by the object. The Motion Node also has its own additional attributes. These are the Start Time and End Time represented as a double data type in seconds. The Speed is also another data type, which is also declared as a double. The Text attribute contains the *Interpolator Nodes* and the Routes attribute, also declared as a string, contains the routing information for motion signals.

5.2 Modules

The animation toolkit has the following modules:

1. VRML to Simplified Scene graph (and vice-versa) Converter: The conversion of a scene graph to a VRML v2.0 text file is an integral part of the system. The animation in the file format allows the user to browse the animation on a VRML player or manipulate it using other VRML authoring software. The pseudo code for changing the scene graph into a VRML file is given in Figure B-2. Alternatively, the scene graph can be stored again in the database as a new animation. The stored animation and its parts can then be used by other users to create new animations or saved to a file. The description of the module for saving the scene graph into the database is given in Figure B-3.

2. Deletion/insertion from/into a simplified scene graph: To create animations, objects must be inserted into the scene graph from the database based on the user's query. The algorithm for inserting objects from the database tables into the scene graph is described in Figure B-4. The delete operation module is a simple node delete and the scene graph array is also updated.

3. Motion reuse module: Motion reuse is another important aspect of our approach. Motion reuse is carried out through the *use* operation that is based on a motion mapping technique [18]. Our approach of translating the user's selected mapping of the Interpolator nodes into VRML format is given in Figure B-5. The algorithm generates the appropriate routing information for the Interpolator nodes based on the user input.

4. Animation engine module: Motion can be either overlapping or in sequence with each other. The motion sequencing is based on the motion times, and is managed by an animation engine in the root node. The animation engine is a Javascript node in the VRML file that triggers the appropriate Interpolator nodes in the animation. An example of an animation with multiple motion nodes in animation is shown Section 5.3 and a more complex animation in Section 5.4.

The Animation engine takes into account all of the start and end times of all the motion nodes and the algorithm to generate the Java script is given in Figure B-6.

4.3 User Interface

The GUI of the system uses a Multiple Document Interface with menus and toolbars. Figure 13 shows the GUI, with some of its children interfaces, which are the scene Graph Window, the VRML Text Document Window, and the VRML Browser Window. The user can create a new animation by choosing the new operation through the menu or the toolbar. This operation creates a new scene graph with only one root node and a blank VRML Text Document, except for the file header.

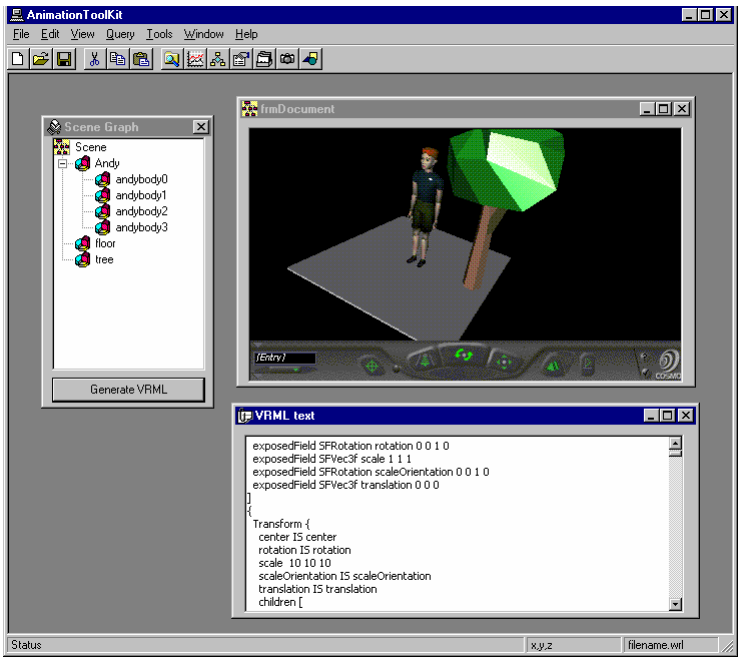

Figure 16. Multi-document interface and toolbars.

The user then utilizes the Query GUI, through which the user can specify the query for required objects and motions and perform the search. The result is displayed to the user in the form of a list. The search may return zero or more objects from the database. When multiple objects match the query, the objects are ranked according to the degree of match in their metadata. When the user selects an object it is inserted to the current scene graph. The user can alter its attributes by double-clicking a specific object node in the scene graph. This calls another window, where the user can specify the new values for the position, size and/or orientation of the object. The animation can be reviewed by converting the scene graph into a VRML file and browsing it. If the query result is a motion on the other hand, the user is asked to select from the scene graph the node on which the motion will be used for. The user then is presented with the motion mapping interface, shown in Figure 17. The user then specifies which joint/segment of the model is to be assigned a particular keyframe. As this may be too tedious, if the original model of the motion and the new model have the same prototype, the user can use the Auto-Assign function. This function parses through the event routing part of the motion and assigns the segments accordingly.

To alter the motion, the user can activate the Motion Adjustment GUI, shown in Figure 18 double clicking the specific motion node in the scene graph. This window allows the user to indicate the parameters for motion adjustment. The user can then apply the operations described in Section 3 on the VRML Text of the motion node.

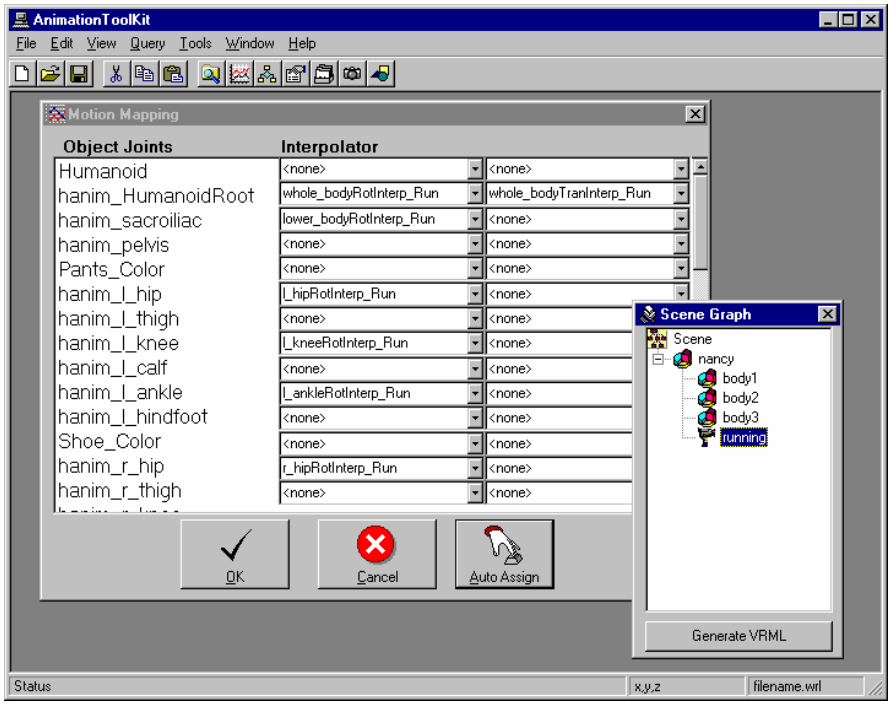

Figure 17. Motion mapping interface.

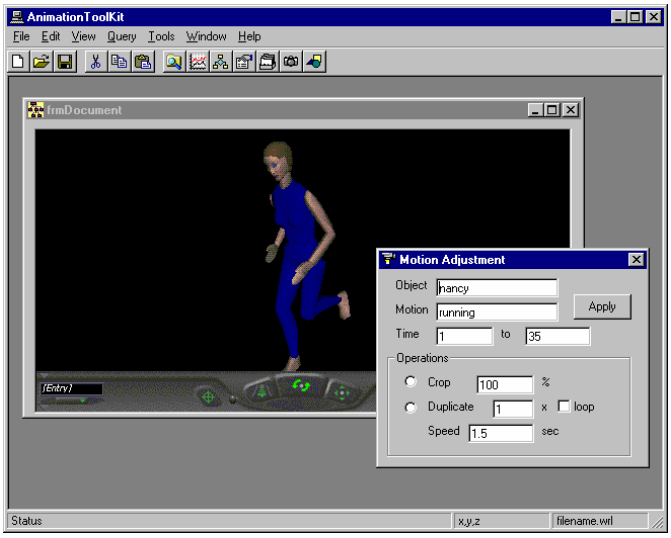

Figure 18. A screen snapshot of the motion adjustment GUI.

To generate more complex motion the user can retarget motions. Retargeting means changing the actual keyframe values of the motion. This can be done by left-clicking a motion node to display all the keyframes of the motion. The user can then edit particular points to fit the new scene. The user can combine motions by selecting the join operation. It is automatically invoked when two or more motions are used to one object. The user then enters the start time and stop time of every motion, permitting overlaps. Alternatively, the user can use the Visualized Time Chart, Figure 19, to visually change the time intervals and start times of the motion.

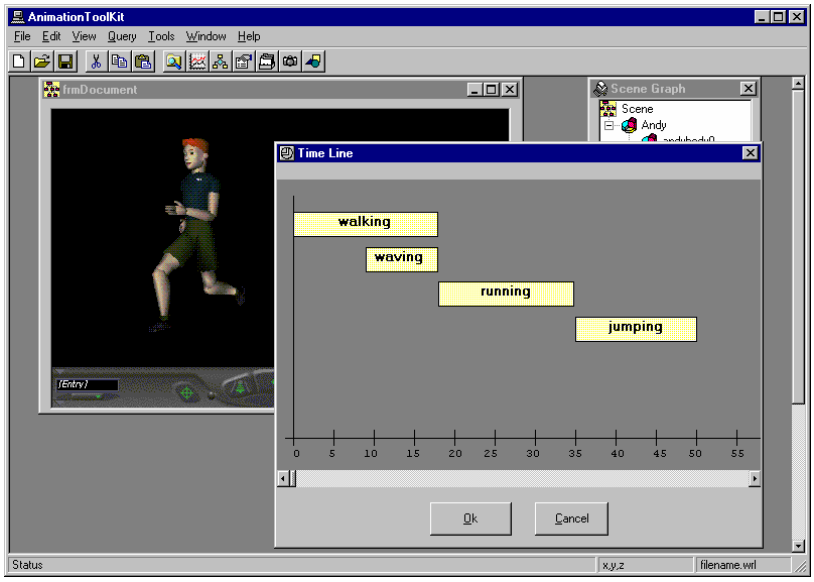

Figure 19. Visualized time chart for motion.

The user can now generate the VRML file, and view the animation. The user can also opt to save the animation in the database or as a file through the menus. It should be observed here that the GUI helps the users to generate new animation sequences without having to remember the syntax of animation database operations. The animation toolkit also provides a set of database management tools under the tools menus for updating the database. The tools provide functions for adding, deleting, and editing of the tables in the relational database. The VRMLText table is linked to the forms for scene, object and motion.

6. Animation Examples

This section illustrates the different possible animations that the animation toolkit can generate. It also shows how the animations were created using the proposed operations written in Section 3. Users can create animations by simply adjusting the motion of existing animation scenes (Section 6.1), reusing motion of other characters to a new one (Section 6.2), and by generating relatively complex animation sequences (Section 6.3).

6.1 Adjusting Motion of a Model

The animation toolkit allows the user to modify the motion in several ways. For example, the user can change the speed of motion and the duration. The modified motion is applied to the same geometric model that the motion was initially applied. In a more general manner, the user can adjust motion to meet the new constraints.

For example, if the user wants to create an animation of a barmaid wiping a table. The user could first try search for scenes that would fit his description of the animation he wanted. For instance, the user finds an animation, shown in Figure 17, similar to his needs. Let us assume that the table is too high to fit the user's requirements of his intended animation. The user can employ the *EDIT* operation on the table to translate the table downward. The next problem is that the hand of the barmaid no longer touches the table. This requires the user to use the *RETARGET* operation, where the user indicates the exact interpolator node to be adjusted, in this case the hand. The user then selects the other interpolator nodes that are to be automatically adjusted based on the new interpolator for the arm. This is done by the system through inverse kinematics [31].

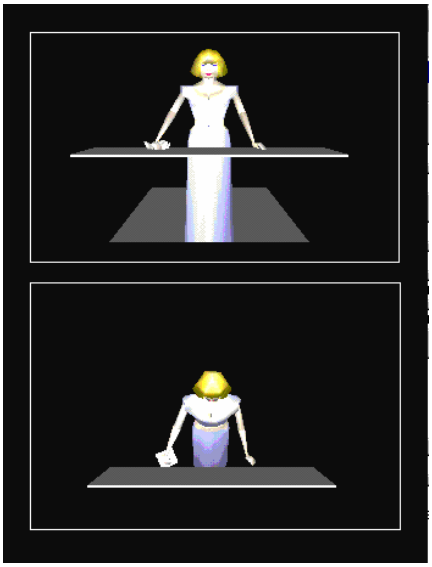

Figure 20. Retargeting an existing motion of the barmaid[26] to the new object in the scene.

6.2 Applying Motion to a Different Model

In some cases, the motion that the user requires may not be existing in the scene that the user has queried. In this case, s/he can query the database for motion of similar objects to obtain one. The motion can be remapped on the model on the current scene graph/animation the user is working on and the motion can also be adjusted to conform to the new scene. For example, the user intends to create an animation of a man walking in a room. The user queries for the animation but the closest match to his scene is just a room with no human models. The user then queries for a 3D model of a man that is walking but he only finds a 3D model of a man without motion. The user then queries for a walking motion, and finds one linked to a woman model. The query results are shown in Figure 21 and the generated animation is shown in Figure 22.

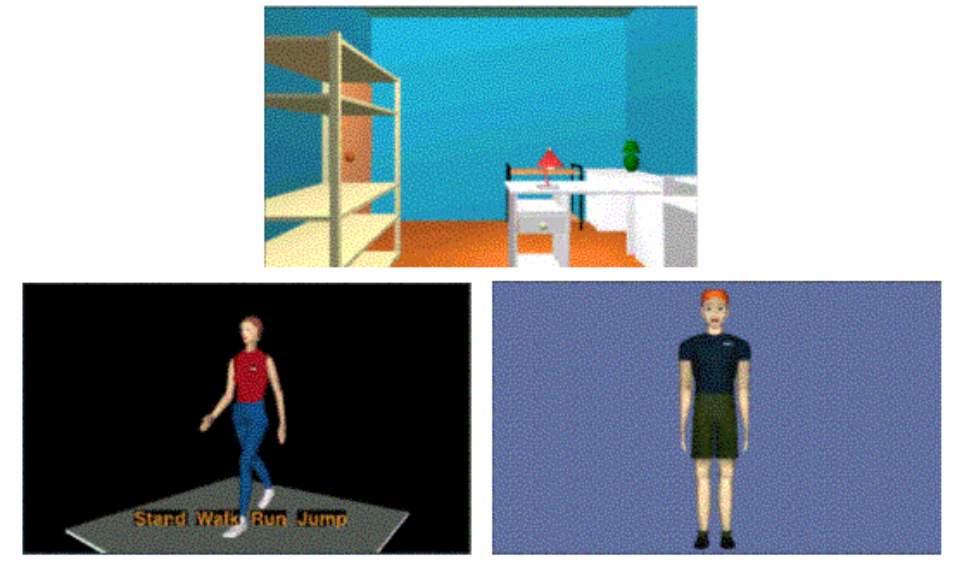

Figure 21. Query Results of a room, an animated woman model [4], and 3D model of a man [32].

Figure 22a. Result after the *INSERT* operation of the model of a man[32].

Figure22b. Result after the *EDIT* operation.

Figure 22c. Result after using the *USE* operation on the walking motion of a woman[4] to a model of a man[32].

The room and the 3D man model are inserted to the scene graph through the *INSERT* operation. The model of the man is scaled to match the room's size using the *EDIT* operation. After which the walking motion is extracted using the *EXTRACT* operations and remapped to the man model using the *USE* operation. The motion mapping technique we use [18] is very flexible. It is similar to the motion reuse work done by Gleicher [12], without the restriction that the two models should have the same structure. Since the interpolator nodes of one model could be connected by the user, using the GUI, to another model it offers much more versatility. For example, the walking model of the woman described in previous example could be remapped to an animal such as a cow. Since the scene graph provides a hierarchical structure of the human and the cow the user can easily connect the respective segments of the cow with the woman's interpolator nodes. The example is shown in Figure 23.

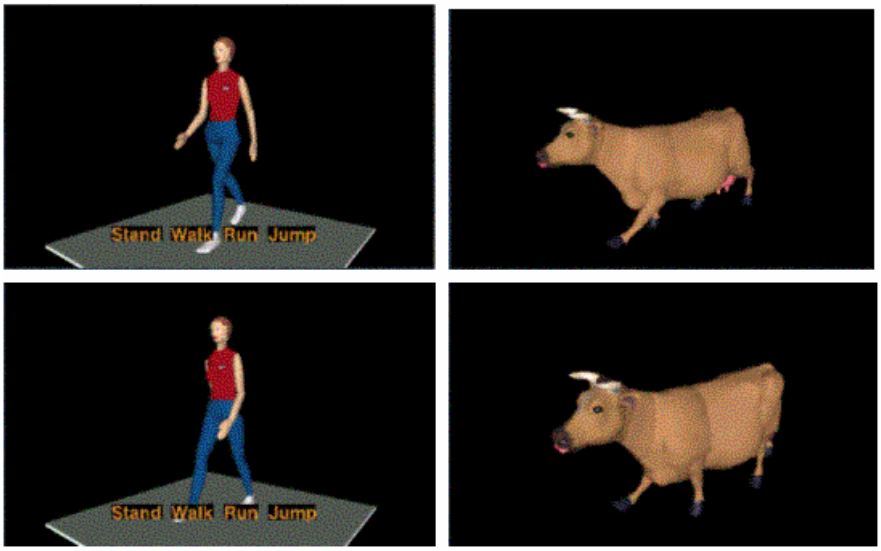

Figure 23 Mapping the walking motion of a woman[4] to a cow[3].

6.3 Combining Motion

Complex motion can be derived by combining two or more motion sequences into a single sequence. The motion sequences may be from a single model or multiple models. The resulting complex motion may be applied to one of the original models or to a different model. An example of complex motion could be a man "walking" (lower body) and "waving" his hand (upper body) at the same time. In Figure 24, a snap shot of the individual motion sequences and the new motion sequence created by utilizing the USE to reuse the walking motion and using the JOIN operation to use for the waving motion.

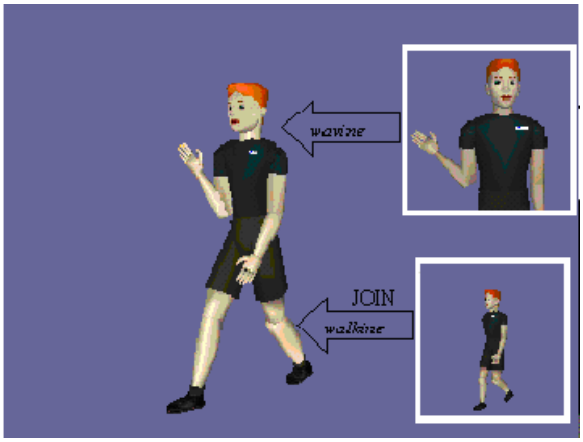

Figure 24 An animation reuse example combining two motions into one object [32].

6.4 Generating Complex Motion Sequences

By combining different operations, users can create more elaborate sequences. The animations are limited only by the resources in the database and the user's creativity. The proposed operations are classical computer animation transformations for both 3D models and objects. However, given a good set of animations in the database the user can create relatively good animations. An example of a complex animation is one that requires some interaction with two objects and the combination of numerous objects and motion in one scene. For instance, if the user wishes to create an animation of a man walking in a park and crosses path with a woman who is jogging. The man then stops walking then turns his head and waves. The woman ignores him and leaves the scene.

To create this elaborate scene, all the necessary objects must first be queried, inserted and edited using the *INSERT* and *EDIT* operation, respectively. Figure 25 shows the query for a particular tree that will be used in the scene, and inserted into a blank scene graph. Figure 26 shows the edit operation for tree, invoked by double clicking the tree node in the scene graph. The user tweaks all the 3D models he has inserted into the scene graph to the desired position, size and rotation. The required motions are then queried and used on the two human models. Figure 27 shows the user querying for walking motions. The timing information is adjusted using the *JOIN* operation to properly sequence the animation shown in Figure 28. The resulting animation is shown in Figure 29.

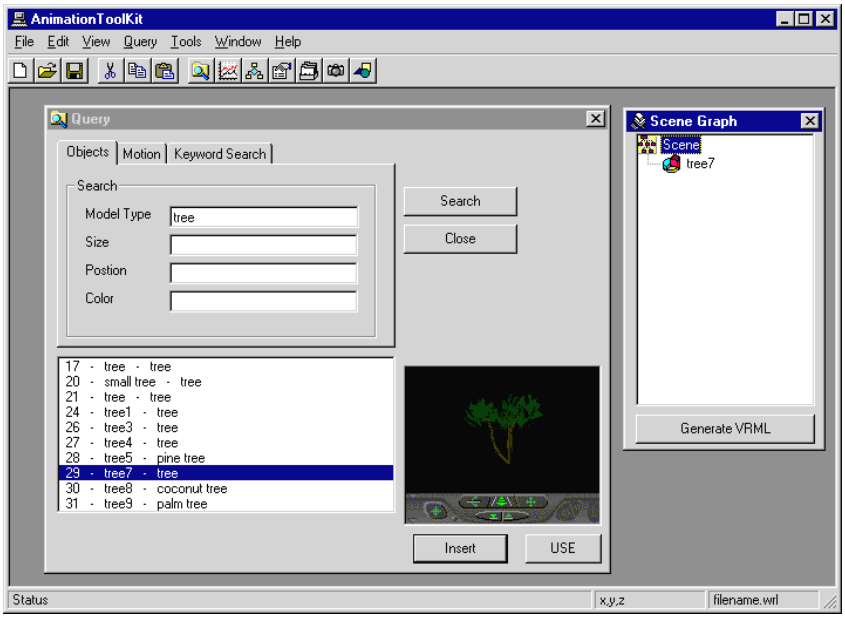

Figure 25. Querying for 3D models to be used for the scene, inserting the results to the scene graph.

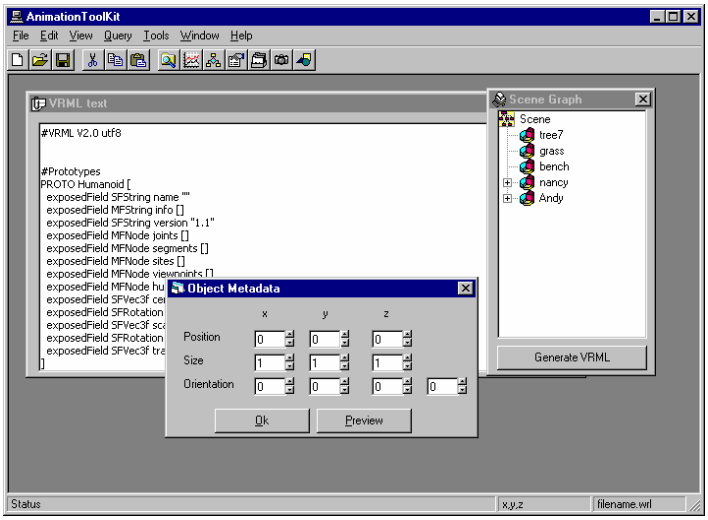

Figure 26 Adjusting the individual object's metadata to fit the new scene.

Figure 27. Querying and reusing motion. The user selects the desired motion and invokes the USE operation to assign the motion to a 3D model.

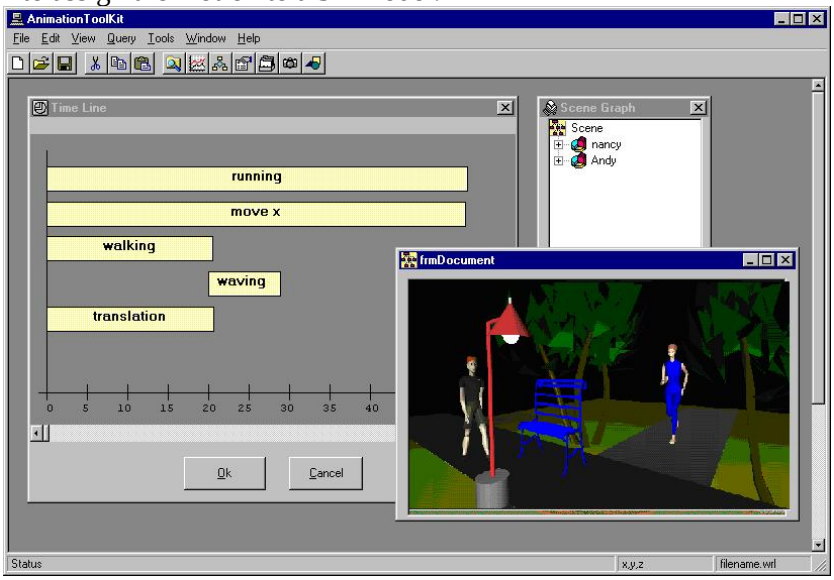

Figure 28. Timing the animation. The start and stop times of each motion are adjusted, this can be directly manipulated through the Visualized Time Line GUI.

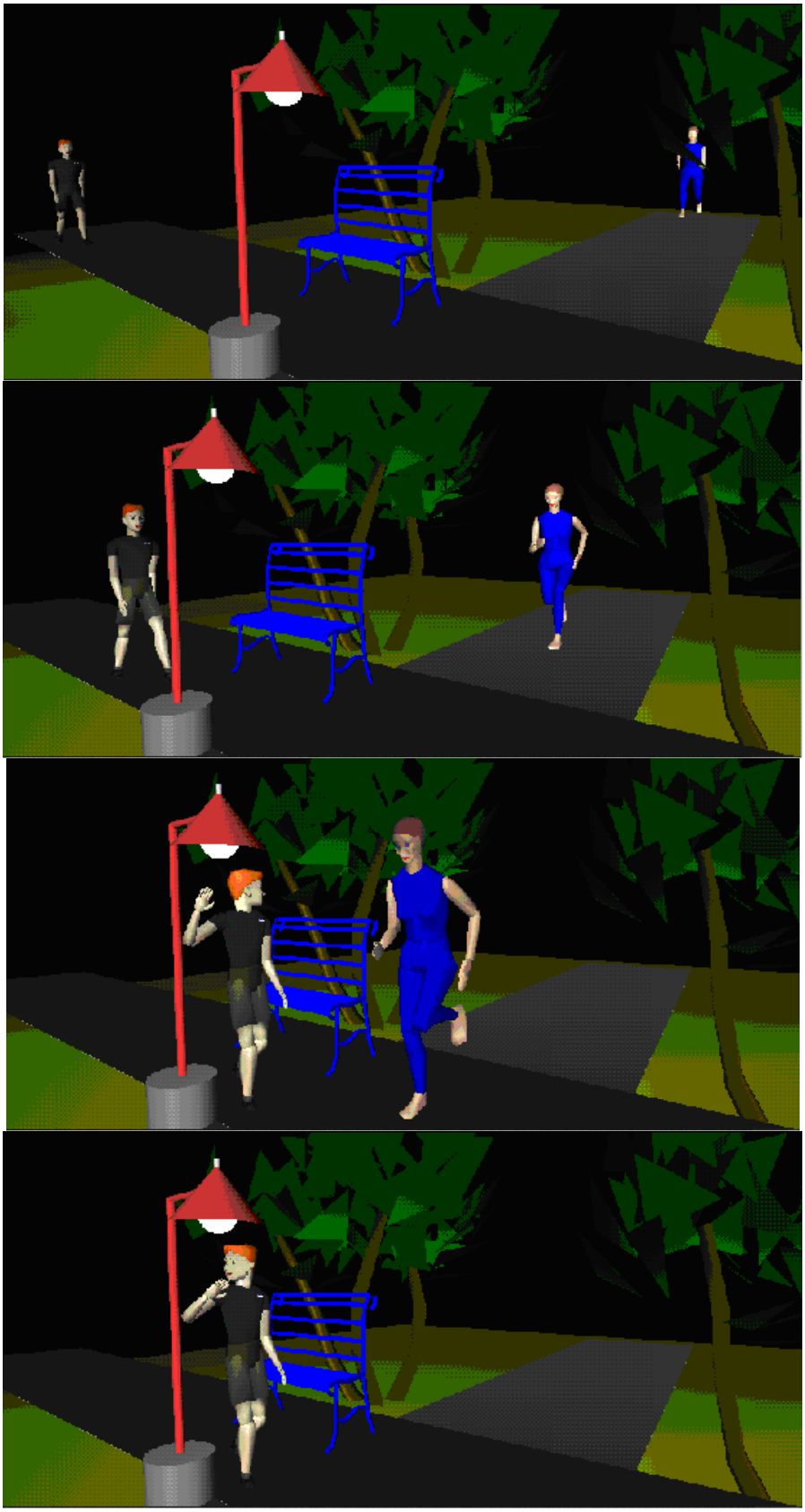

Figure 29. Complex animation of a man[32] and a woman[4] in a park.

7. Related Work

Most animation tools are intended for skilled animators to create motion sequences from scratch. Only a few software tools cater to novice users by providing the function of reuse. Moreover, most of the research work in computer graphics has been geared towards creating highly realistic models and animations.

7.1 Software Tools

With the advent of faster and more powerful machines, there has been an increase in the number of computer animation tools available in the market. Some of the popular tools are 3D studio Max [10], Maya and Lightwave 3D to name a few. However, these tools are very complicated, requiring hours of training and advance knowledge in computer graphics. This is certainly beyond the reach of novice animators. There are some tools that intend to simplify the animation authoring process. For example, Simply 3D by Metacreations [21] provides a catalog of 3D models that could be dragged into a scene. This means that the user does not need to create 3D models from scratch. However, they do not have a search engine built on a database to query for geometric models. They require the user to browse through a categorized catalog of 3D models. It does not support reusing animations of complex articulated figures. It does support dragging simple animations to be applied to the 3D models. These animations include basic translation, rotation and scaling, with some predefined path movement. Lee and Lee [18] proposed an animation toolkit to create animations interactively. Rather than reusing 3D models they have a mesh generator, which automatically creates the 3D model based on 2D images. The generation of the motion is based on a motion mapping technique to apply the motion of one object to another interactively. The motion mapping technique is based on a hierarchical data structure similar to that of a scene graph.

7.2 Animations Databases

Thalmann et al. [30] presented an Informed Environment 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 [15]. It uses a scene graph and an animated agent in multimedia presentations. The animated agent performs a presentation based on a text description. In order to generate the agent's motion automatically, the motion of the agent is categorized into three classes, namely pointing, moving, and gesturing. To determine what the target object is and to extract detailed information about the object, the system accesses a scene graph that contains all the information about the virtual environments. This information, stored in a database, is used for determining details to generate the agent's motion. Similarly, Ayadin et al. [2] have used databases to guide the grasp movement of virtual actors. Their approach creates a database based on divisions of the reachable space of a virtual actor. Objects are classified into a set of primitives such as blocks, cylinders, and spheres. Both attempts of storing objects into databases have been fairly successful. However, the problem of motion reuse is not addressed.

6.3 Animation Reuse

The major idea of 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. One method proposed by Bruderlin and Williams [8] considers motions as signals and applies techniques of signal processing to adapt them. They introduced a multi-resolution filtering, multi-target motion interpolation with dynamic time warping, wave shifting and motion displacement mapping. The pre-existing motions can be manipulated interactively. The approach was meant as a complement to keyframing, motion capture and procedural animation. A variant of the above method called the *motion-displacement mapping*, was introduced by Popovic and Witkin [24]. Their approach uses space-time constraints dynamic formulation for transforming character animation sequences

that maintains the integral physical properties of the motion. In addition, they have described a new methodology for motion mapping between models with drastically different number of freedoms.

Gleicher's work [12] is concerned with retargeting motion of one articulated figure to another with the same structure but different lengths. He also uses a space-time constraint solver to compute an adapted motion and preserve the frequency characteristics of the original signal. Hodgins [14], similarly worked on adapting motion for other articulated figures with different segment lengths. Their algorithm has two stages, first the control system parameters are scaled based on sizes, masses and moments of inertia. Then some of the parameters are fine-tuned using a search process on simulated annealing. Monzani et al. [22], in their work, present a method for solving the Motion Retargeting Problem, by using an intermediate skeleton. This allows us to convert movements between hierarchically and geometrically different characters. An Inverse Kinematics engine is then used to enforce Cartesian constraints while staying as close as possible to the captured motion. All these methods addressed the reuse for a specific type of animation and manipulated on the low level kinematics or dynamic structures of geometric models.

7.4 Animation Asset Management Systems

Shatkin [28] surveyed the computer animation companies on what type of asset management system they were using by collating a series of interviews with lead personnel of the animation firms. Kaufma's [16] article on Pixar's Asset Management System, revealed that the company had just started using such a system after Toy Story, the first full length computer animated film. Before that they kept track of all of their animations manually. They do not use third party asset management tools but have created their own. The asset management system basically keeps track of the metadata associated with animations. These include the versions, dates the animations where created, who created them and the actual location of the digital material. Currently most large computer animation companies have adapted their own asset management system.

8. Conclusion

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. This paper, addresses the issue of generating animations by reusing existing motion information and geometric models. This paper has proposed a relational database approach based on simplified scene graph model as a solution for reusing motions. The simplified scene graph model allows motion sequences to be stored along with the models. The paper then proposes a set of operations to manipulate the simplified scene graphs as databases. These operations help in manipulating models and their motion characteristics, and create new animation sequences. The novel idea of the paper is a framework for creating animations by the procedures on reusing models and motion that are stored in databases, rather than the common interactive way starting modeling and animation from scratch.

Currently, the metadata in the animation database have been manually annotated and represented as string. Similarity measures for query resolutions were also simple. We are working on incorporating features for semi-automatic metadata generation from VRML files. We are also looking for a more effective similarity measures for comparison of models and motion sequences in the database. Future work could adapt the current metadata into standards accepted by organizations, such Resource Description Format (RDF). Also, the paper does not try to solve transition problems or resolve object collision problems. Similar to other animation tools the user has to manually tweak the animations for proper object interaction.

Acknowledgement

Conrado R. Ruiz, Jr. and Akanksha were recipients of the Research Scholarship from National University of Singapore while working on this project. This material is based upon the work supported by the National Science Foundation under Grant No. 0237954 for the project CAREER: Animation Databases. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. Akihiro, K. 3D Shape Similarity Metric based on the Correspondence of Points. Graduation Thesis, Kyoto University. (1997).
2. Ayadin, Y., Takahashi, H. and Nakajima, M. Database Guided Animation of Grasp Movement for Virtual Actors. Proc. Multimedia Modeling '97. (1997) 213-225
3. Christian Babski (LIG-EPFL) . http://ligwww.epfl.ch/~babski/StandardBody/.
4. Ballreich, C. Nancy - 3D Model. 3Name3D.http://www.ballreich.net/vrml/h-anim/nancy_h-anim.wrl. (1997).
5. Bekker,H., Belgers, I., Valkenburg P. Inventory of Metadata for Multimedia, http://www.surfnet.nl/innovatie/surfworks/doc/mmmetadata/, (2000)
6. Böhm, K. and Rawok. T. C. Metadata for Multimedia Documents. SIGMOD-Record Special Issue on Metadata for Digital Media. Vol. 23, No. 4, Dec. 1994.
7. Braun, Norbert; Dörner, Ralf; Soetebier, Ingo, A VRML and Java-Based Interface for Retrieving VRML Content in Object-Oriented Databases. In: Bra, Paul de (Hrsg.) u.a.; Association for the Advancement of Computing in Education: WebNet 99 - World Conference of the WWW and Internet. Proceedings. Charlottesville, VA, USA : Association for the Advancement of Computing in Education, AACE, (1999) S. 987-992
8. Bruderlin, A. and Williams, L. Motion Signal Processing. Proc. ACM SIGGRAPH '95. (1995) 97-104.
9. Brutzman, D. Composing Scene Graph Alternatives. www.web3D.org/TaskGroups/x3d/translation/ComposingAlternateSceneGraphs.html (2000).
10. Discreet, Autodesk, Inc. 3D Studio Max. www.discreet.com/products/3dsmax/ (2003)
11. Funkhouser, T., Min, P., Kazhdan, M., Chen, J., Halderman, A., Dobkin, D. and Jacobs, D. A Search Engine for 3D Models. ACM Transactions on Graphics Vol. V. (2003).
12. Gleicher, M. Retargeting Motion for New Characters. Proc. ACM SIGGRAPH '98.(1998) 33-42.
13. Grünvogel, S., Piesk, J., Schwichtenberg, S. & Büchel, G. (2002). AMOBA: A Database System for Annotating Captured Human Movements. In: Proceedings of Computer Animation 2002 (CA2002), 19-21 June 2002, Genevea, Switzerland IEEE Computer Society, Los Alamitos, pp. 98 – 102
14. Hodgins, J. and Pollard, N. Adapting Simulated Behaviors For New Characters. Proc. ACM SIGGRAPH '97. Los Angeles, CA. (1997) 153-162.
15. Kakizaki, K. Generating the Animation of a 3D Agent from Explanatory Text. Proc. ACM MM '98. (1998) 139-144.
16. Kuafman, Debra. Interview with Darwyn Peachey and Greg Brandeau on Pixar's Asset Management System. Creative Planet . http://www.designinmotion.com/article/mainv/0,7220,113467,00.html, 1999.
17. Leahy Jr, M. B., Nugent, L. M., Saridis, G. N., and Valavanis, K. P. Efficient Puma Manipulator Jacobian Calculation And Inversion. Journal of Robotic Systems, (1987), 185-197.
18. Lee, G. C. S. Tutorial on Robotics, chapter 2, pages 47-65. IEEE Computer Society (1983)..
19. Lee, W.M. and Lee M.G. An Animation Toolkit Based on Motion Mapping. IEEE Computer Graphics International. (2000) 11-17.

20. Martinez, J. Overview of the MPEG-7 Standard. http://www.cselt.it/mpeg/standards/mpeg-7/mpeg-7.htm, 2001.
21. Micrografx, Inc. Products: Micrografx Simply 3D: Product Info. http://www.micrografx.com/mgxproducts.
22. Monzani J. S., Baerlocher P., Boulic R., and Thalmann D. Using an Intermediate Skeleton and Inverse Kinematics for Motion Retargeting. Proc. Eurographics 2000.
23. Paul, R. P., Shimano, B., and Mayer, G. E. Kinematic control equations for simple manipulators. IEEE Transactions On Systems, Man, And Cybernetics, 1981, SMC-11(6), 66-72.
24. Popovic, Z. and Witkin, A. Physically Based Motion Transformation. Proc. ACM SIGGRAPH '99.(1999) 11-19 .
25. Prabhakaran, B., Jiao, B., Ruiz, C., Huang, Z., "Reusing Animations in Databases for Multimedia Presentations", Proc. Asian Computing Conference, 2000.
26. Reitemeyer, A. Barmaid Bot http://www.geometrek.com/web3d/objects.html.
27. Samet, H. The Design and Analysis of Spatial Data Structures. Addison-Wesley Publishing Company, INC. (1989).
28. Elina Shatkin. Spotlight on Asset Management. Creative Planet Inc. http://www.designinmotion.com/article/mainv/0,7220,103709,00.html, 1999.
29. Rohlf J. and Helman J. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. Proc. ACM SIGGRAPH '95. 550-557.
30. Thalmann D., Farenc N., and Boulic R. Virtual Human Life Simulation and Database: Why and How. Proc. International Symposium on Database Applications in Non-Traditional Environments (DANTE'99). IEEE CS Press, 1999.
31. D. Tolani, A. Goswami, and N. Badler. Real-time inverse kinematics Techniques for anthropomorphic limbs. Graphical Models 62(5), Sept. 2000, 353-388.
32. Vcom3D, Inc. - Seamless Solutions, Andy - H-Anim Working Group. http://www.seamless-solutions.com/html/ animation/humanoid_animation.htm. 1998.
33. The VRML Consortium Incorporated. The Virtual Reality Modeling Language. http://www.vrml.org/Specifications/VRML97/. International Standard ISO/IEC 14772-1: 1997.
34. Walczak, K. Integration of Virtual Reality and Multimedia Data in Databases, Multimedia DatabaseManagement Systems, Proceedings of International Workshop on , pp.: (1996). 80 - 84
35. Wang, C.S., Shih, T., Huang C., and Chen J. Content-Based Information Retrieval For VRML 3D Objects. Proc. 17th International Conference on Advanced Information Networking and Application (AINA'03) 2003.
36. Watt, A. and Watt, M. . Advanced Animation and Rendering Techniques. Addison-Wesley. 1992

Appendix A

Database Tables and Relationships

Scene : Table

Field Name	Data Type	Description
SceneID	Number	A unique ID given to each scene
SceneName	Text	The name of the Scene
Metadata	Text	Keywords to describe th e scene
STextID	Number	The unique ID used to link the scne to its VRML Text

Table A-1 Scene table in the animation database

Object : Table

Field Name	Data Type	Description
ObjectID	Number	A unique ID given to a record
CategoryID	Number	A unique ID to link to link the record to a specific category
ObjectName	Text	The name of the object
ModelType	Text	The model type of the object, e.g. human, chair
Size_x	Number	The scaling factor in the x-axis, double
Size_y	Number	The scaling factor in the y-axis, double
Size_z	Number	The scaling factor in the z-axis, double
Position_x	Number	The x coordinate of the object's position
Position_y	Number	The y coordinate of the object's position
Position_z	Number	The z coordinate of the object's position
Rotation_x	Number	The x coordinate of the rotation's axis
Rotation_y	Number	The y coordinate of the rotation's axis
Rotation_z	Number	The z coordinate of the rotation's axis
Rotation_theta	Number	The angle around the axis, in radians
Color_R	Number	The red value of the color
Color_G	Number	The green value of the color
Color_B	Number	The blue value of the color
OTextID	Number	The unique Id used to link to the VRML Text Description
MotionID	Number	The unique ID of the motion
SceneID	Number	The unique ID of the scene where it belong to
ParentID	Number	The Parent, if any
ProtoID	Number	The unique ID of the VRML Text for the Prototype
StandAlone	Yes/No	Whether the object can stand on its own for if it needs its siblings
SiblingOrder	Number	The specific order of the object among its siblings

Table A-2 Object table in the animation database

Motion : Table

Field Name	Data Type	Description
MotionID	Number	A unique ID given to a motion record, long integer
MotionName	Text	The name of the motion
MotionType	Text	The type of motion
MotionSpeed	Number	The speed of the motion in secs for one interval cyle
MotionStyle	Text	The motion style, e.g. slowly, happy, etc.
StartTime	Number	The start time in seconds
EndTime	Number	The time the animtion will terminate
MTextID	Number	A unique ID that links the record to its Text Description
ObjectID	Number	A unique ID that links the record to the object it is attached to

Table A-3 Motion table in the animation database

Category : Table

Field Name	Data Type	Description
CategoryID	Number	A unique ID given to a category
CategoryName	Text	The category name
Metadata	Text	Keywords to describe the category

Table A-4 Category table in the animation database

VRMLText : Table

Field Name	Data Type	Description
TextID	Number	A unique ID given to a VRML Text Description
VRMLText	Memo	The Text Description, max 64K

Table A-5 VRML Text table in the animation database

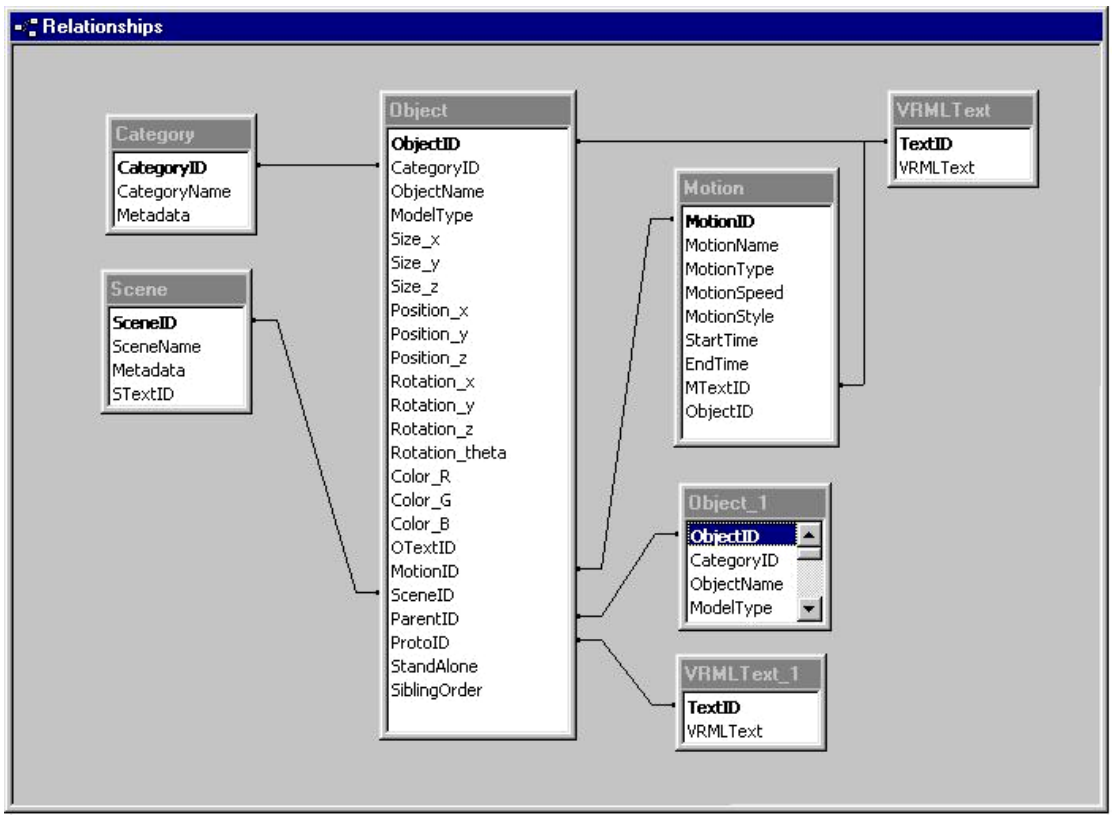

Table A-6 Access database relationships. Note that VRMLText and VRMLText_1 are the same tables shown in two instances for clarity. Likewise, Object and Object_1 are the same tables.

Appendix B

Pseudo-codes

```
Public Type RotationType
X As Double
Y As Double
z As Double
theta As Double
End Type

Public Type PointType
X As Double
Y As Double
z As Double
End Type

Public Type ObjectNode
Name As String
ID as String
Text As String
Position As PointType
Size As PointType
Rotation As RotationType
Proto As String
isMotion as Boolean

End Type

Public Type MotionNode
Name As String
ID As String
Text As String
isMotion As Boolean
Routes As String
StartTime As Double
EndTime As Double
Speed As Double

End Type

Global SceneNodes()
```

Figure B-1. The scene graph data type declaration.

```
Procedure SceneGraph2File(Root as Node) {


   Declare PrototypeText, NodesText
           RoutesText 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 RoutesText
                Get temporal information
                    and update AnimationEngine
     End if
   }
     Open File
     Save to File ( "#VRML V2.0 utf8" )
     Save to File ( Protypetext + NodesText +
                    RoutesText + AnimationEngine )
     Close File
```

Figure B-2. Pseudo code for generating a VRML file from a scene graph.

```
Procedure SceneGraph2DB(Root as Node) {

   Open Database for writing and lock exclusive
   Create a new record in the Scene table
   Traverse all the nodes of the scene graph by DFS

   For each node in the scene graph except the root node {

      If node.isMOtion = TRUE Then
           Create a new record in the Motion Table
           Store all attributes into the appropriate fields
           Store the record field ObjectID as the parent node
      Else
           Create a new record in the Object table
           Store all attributes into the appropriate fields
           If Object is not a level 2 node Then
              'object has a parent which is not the root node
               Store the record field ParentID as the parent Node
           End

    }

     Release exclusive lock
     Close Database
 }
```

Figure B-3 Pseudo code for storing a scene graph into a database.

```
Procedure INSERT(ObjectID as Interger) {
   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
       Insert in scene graph array
   Else
       Change the unique key of the temporary node and declare
           As a new instance
       Insert the modified temporary node.
       Insert in scene graph array
   End if
   Close Database
 }
```

Figure B-2. Pseudo code for inserting an object from the database into the scene graph.

```
 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 list to the user the joints and interpolator
      nodes and allow the user to assign them respectively

      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
}
```

Figure B-5. Pseudo code for the motion mapping in VRML

```
Procedure Generate_Animation_Engine(RootNode as SceneGraphNode) {

Define JavaScript as String

   Parse through the VRML Text Description of all the Motion Nodes in
   the scene graph and extract the start and end times of each.

   Append to JavaScript:
     “DEF TimerControl Script {
           eventIn SFTime enterTime
           eventOut SFBool loop”

   For every Motion node {
       Extract the Timer Name of the Motion node
       Append to JavaScript:
           “eventOut SFTime” + Timer Name + “_startTime”
           “eventOut SFTime” + Timer_Name + “_endTime”

   }
   Append to JavaScript:
   “url [
    "javascript:
        function enterTime(time) {”

   For every Motion node {
       Extract the Timer Name of the Motion node
       Append to JavaScript:
           Timer Name + “_startTime” = time + Start Time
           Timer Name + “_startTime” = time + Start Time

   }
   Append to JavaScript:
     “loop = true;         }"
      ] } ”

     Create a Visibility Sensor to Trigger the start of the Javascript
     node including routing information

     Create Routes for triggering the Time Sensor nodes of all the
     motion from the JavaScript node

}
```

Figure B-6. Psuedo code for generating the javascript for the animation engine.