Vol 16 | Issue 01 | January 2002

Reviews

Karagöz: A Hierarchical Modeling and Animation System for Turkish Shadow Theater
by Ugur Güdükbay & Tolga Abaci

This months Feature
Last month's reviews
Index of all reviews
Review guidelines

In this paper, we describe a hierarchical modeling and animation system for the Turkish Shadow Theater, Karagöz. Using hierarchical modeling, Karagöz is able to animate characters of the Turkish Shadow Theater by using model parameters related to the dierent parts of the body and the jointparameters between these parts. The system consists of two parts: the animation editor that is used as an authoring tool to create keyframe animations, and the model editor, that is used for creating the models used in the animations (e.g. Hacivat and Karagöz). The most important feature of Karagöz is its ability to support characters with an arbitrary structure. This is possible since in Karagöz, hierarchical modeling is implemented in a generic way, and direct manipulation techniques are used in the user interface, which eliminate the need for character-specic control knobs.

The first performances of *Karagöz* (Karagheus), the traditional Turkish Shadow Theatre date back to the 16th century. It was one of the most popular forms of entertainment right up until the cinema replaced it in the late 1950s [1, 2, 4]. The shadow play is performed on a translucent screen by manipulating shadow play characters, like *Karagöz* and *Hacivat,* behind the screen. The characters are manipulated by using sticks attached to different parts of the characters. While the characters are manipulated, the light source behind the screen causes the shadows of the characters to appear on the screen.

In this paper, we describe a hierarchical modeling

and animation system to simulate the Turkish Shadow Theater, Karagöz. The system, called Karagöz, uses hierarchical modeling to construct and animate two-dimensional articulated characters containing body parts and joints between these body parts. Different characters that have different body parts and joints have different hierarchical structures. Texture mapping [5] is used for rendering the characters since different body parts are modeled as simple two-dimensional polygon meshes and have a predefined texture that can be mapped to these polygons as the model animates. To animate the models, the system uses keyframing based on the model parameters of the characters. These model parameters include the positions and orientations of the characters and the joint angles between different body parts of the characters. The Karagöz system consists of two major components:

These model parameters include the positions and orientations of the characters and the joint angles between different body parts of the characters. The Karagöz system consists of two major components:

The animation editor functions as an authoring tool creating keyframe animations involving these characters, by editing the character parameters such as position and orientation for different keyframes. These animations can then be played back by reading the animation parameters for each keyframe from a file and interpolating between the keyframes. The interpolated frames are rendered by using texture mapping on the characters appearing on the frame.

The model editor is the component of the system where new character models can be created. This component actually includes another sub component: The node editor, using which individual nodes of a hierarchical model are created, by specifying the geometry and the texture. The models created by the model editor can be saved and later used in animations to be authored in the animation editor.

The integration of two components outlined above provides a complete and easy-to-use environment, which allows fast creation of animations. The main

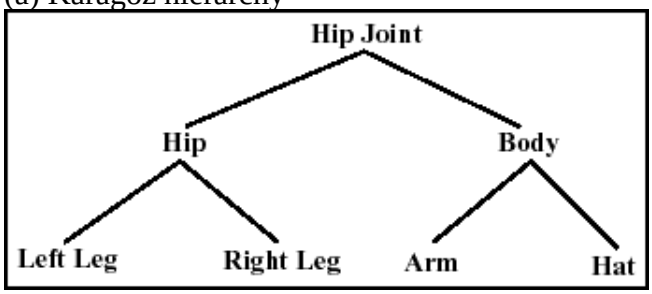
contributions of this paper are solutions to the issues related to the design and implementation of such a system, which are twofold:

A generic implementation of hierarchical modeling The ability to add new characters to the system requires that the system provide a generic implementation of hierarchical modeling, rather than one in which hardwired implementations are provided for each character present in the system.

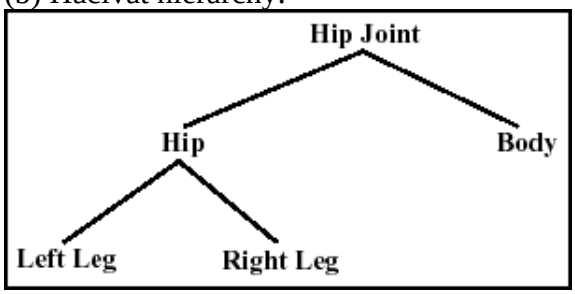
A powerful and easy-to-use interface is very important for allowing an artist to leverage his/her creative capabilities. However, there is a technically important side to the user interface of Karagöz as well. The importance of the user interface is that by employing direct manipulation techniques, it becomes the prime means of integration between the two components of Karagöz [7]. Such a well-designed interface helps tie the components together in a stronger fashion.

Figure 1

(a) Karagöz hierarchy



(b) Hacivat hierarchy.



In the rest of this paper, we describe how Karagöz, in its current state, solves the above issues. First, the

generic hierarchical modeling in Karagöz is examined. In the next section, we have a look at the user interface components of Karagöz. Then, some important points and experiences regarding the implementation of Karagöz are mentioned. The last section concludes our discussion.

Hierarchical Modeling in Karagöz

Hierarchical modeling is a technique to model and animate articulated structures consisting of links connected by joints. Hierarchical modeling accomplishes this as follows: once simple parts of a hierarchy are defined as primitives, we combine these simple parts into more complex hierarchical (articulated) objects. The main theme of hierarchical modeling is that a group of parts (aggregate objects) can be treated just like a primitive object. Any operation that can be performed on a primitive object can be applied to aggregate objects too [3].

Since Karagöz requires the ability to support arbitrary hierarchical models, it needs a data structure that can represent any hierarchical model. It can be said that the most suitable data structure for the representation of a generic hierarchy is an n-way tree, so Karagöz uses this structure for the representation of hierarchical models. Since the main theme of hierarchical modeling is the treatment of aggregate objects just as in the same way as the simple objects, a tree suits in very well, due to its recursive nature.

Each node in the model tree has a primitive object associated with it. The leaf nodes in the model tree represent the simplest objects such as arms or legs, while subtrees of the model tree represent aggregate objects such as the body, the hip, or even the whole model. It is important to emphasize the difference between the simple and the primitive objects. For example, a close examination of Figure 1 reveals that a leg is a simple object consisting of a single primitive object, while the lower body is an aggregate object that is a combination of the two legs (that are the child nodes of the lower body node), and the hip, which is the primitive object associated with the lower body node. As in this example, the primitive objects associated with the intermediate nodes are used as connectives that bring together various other objects, whether aggregate or simple. As a result, moving from the bottom of the tree to the

top, the leaf objects and the connectives reduce to the whole model, following the hierarchical pattern.

The main operation defined on the hierarchical model tree is drawing. The model is drawn by traversing the tree from the top to the bottom. During the traversal, the course of action described in Figure 2 is taken at each node n. Regarding the tree traversal operation defined in Figure 2, there are two points that are of interest:

> First, as is made explicit in the course of action given above, the order in which the child nodes are traversed and the primitive objects are drawn is important. If we return to the example of -the lower body- given above, it is obvious that if the model is facing right, the left leg should be rendered first, then the hip should be rendered, and at last the right leg should be rendered. As a result, the order of the child nodes specified within the tree should not be ignored during the traversal for drawing, if we want the resulting image to be the same as the intended one.
>
> Second, while transformations that are related to an object represented by a subtree t rooted at node n are being applied, the formerly applied transformations must not be lost. In other words, the state of the graphics renderer at the second step of the traversal of the subtree t must be a combination of the transformations applied at the nodes on the path from node n up to the root node of the model. This property is essential for being able to treat aggregate objects in the same way as simple ones, since it guarantees that the transformations applied on a node are also in effect during the rendering of the child nodes of that node. A similar constraint applies to the last step, that the undoing of the transformations results just in the combination of the previously applied transformations.

Figure 2:
Pseudocode of the algorithm for traversing the hierarchical model tree.

```
/* preList(n): The list associated with node n
   which includes nodes that should be rendered
   before the primitive object of n.
   postList(n): The list associated with node n
   which includes nodes that should be rendered
   after the primitive object of n.
*/

Traverse(n)
begin
  ApplyTransformations(n)
  for each Node t in preList(n) do
    Traverse(t)
  Render(n)
  for each Node t in postList(n) do
    Traverse(t)
  UndoTransformations(n)
end
```

The hierarchical model renderer of Karagöz provides operations that can be used to create, manipulate, and destroy model trees, along with operations that are used for rendering using OpenGL and general user interaction. These operations are used by the help of the user interface components of Karagöz.

Karagöz User Interface

Figure 3:
Karagöz user interface in animation editing mode

As mentioned before, Karagöz consists of the following two components: The Animation Editor and The Hierarchy (Model) Editor. In the following

sections, the features provided by these components are examined in detail.

Hierarchy (Model) Editor
The hierarchy editor facilitates quick creation of reusable models for animations. When Karagöz operates in this mode, the main work area behaves as a model canvas. The construction of an hierarchical model using the Karagöz Model Editor comprises of the following actions:

Definition of the primitive objects (parts of the models) through the node editor.

Declaration of the hierarchical relationships between the simple / aggregate objects.

Specification of the relative transformations between hierarchically-related simple / aggregate objects.

Each of these actions is accomplished through the graphical user interface of the Karagöz Model Editor, which means that no textual representation is required. Although due to obvious reasons, the actions must be logically performed in the order of the list given above, there is no need to perform the actions in strictly-separated steps. For example, after defining a number of primitive objects and declaring the hierarchical relationships between them, the user can define other primitive objects and declare other relationships, and specify the relative transformations in the very end. However, it is clear that you cannot specify relative transformations between objects before they even exist.

Definition of primitive objects is accomplished through the Node Editor, as mentioned above. The node editor allows the user to load an image to be used as the source for the textures. This image should contain all textures to be used for the model. The user can then specify on the image the boundary points for each primitive object. This operation can be considered analogous to cutting figures from a piece of paper. As the node definitions are completed, they appear on the model canvas, as individual, independent, simple (leaf node) objects. Nodes that have already been defined can be edited (without altering the hierarchical relationships

already defined on them) as well, by selecting them on the model canvas (cf. Figure 4).

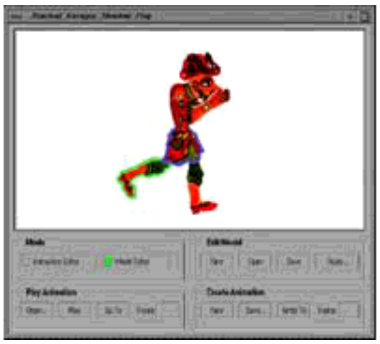
Declaration of hierarchical relationships is performed on the model canvas, on the simple or aggregate objects that have been defined (cf. Figure 5). After selecting a primitive object a, with the specific key and mouse button combination, clicking on a different primitive object b, makes the node associated with object a, a direct child of the node associated with primitive object a. Since only primitive objects can be selected, the following should be elaborated:

* The case of selecting a simple object is trivial, since they contain only one primitive object.

Figure 4: Node Designer Tool of the Hierarchy Editor; this tool is used to create the parts of an articulated character by using textures and polygons defined by using mouse.

* The case of selecting an aggregate object is accomplished through selecting the primitive (connective) object associated with the root node of the subtree defining that aggregate object.

Figure 5: Karagöz user interface in hierarchy editing mode; this part constructs the hierarchical structure by defining the parent child relationships between the parts using mouse and key combinations.



Specification of the relative transformations is also performed on the model canvas. Actually, the model

editor deduces the relative transformations between hierarchically-related objects automatically, from the state of the objects on the model canvas. The user interacts with the model, as it appears on the canvas, by applying transformations such as translation and rotation on the objects. For each transformation that can be applied, a key and mouse button combination exists. For example, for specifying a relative angle between two hierarchically-related objects, the user first selects the child object on the model canvas, and while holding down the appropriate mouse-button and key combination, rotates the chosen object by dragging it with the mouse. When the mouse button is released, the relative angle between the hierarchically-related objects is determined, from the state of the objects on the canvas.

Animation Editor

The animation editor has been designed specifically for very fast creation of professional animations with little effort. The editor actually functions as an authoring tool to create keyframe animations. The keyframes are produced by the user adjusting the various model parameters, such as orientation and position. Once the keyframe specification is completed, the system may play back the animation by interpolating between the keyframes. It should be noted that it is possible to incorporate any model that has been defined through the Model Editor in any one of the keyframes of an animation.

When Karagöz operates in the animation editor mode, the main work area functions as the animation canvas. For creating a new keyframe, the user first specifies the sequence number for the keyframe. Then, the models that will appear in the keyframe will be selected, and the selected models will appear immediately on the canvas. After this, the user can directly manipulate the positions, orientations and postures of the models on the canvas. For example, for the Hacivat model in Figure 3, the user can click on one of the legs, and dragging with the mouse, rotate that leg. It is also possible to rotate or move the whole object. Since the models are hierarchical, when a parameter of an aggregate object is changed, the simpler objects that constitute it will also change parameters accordingly. For example, when the hip of the Hacivat model in Figure 3 is rotated, the legs will also rotate with it. It is important to note that the animation editor does not allow the user to manipulate a model in such a way that its integrity is

broken. For example, in the Hacivat model of Figure 3, the user can easily rotate the legs, but he cannot tear them apart from the rest of the model. If such actions are intended, the model editor must be used. However, the changes made in the model editor are applied globally, so all keyframes will be modified, which also means that interpolation cannot be applied in such cases.

It should be noted that all properties of the model that can be manipulated by the user will be subject to the interpolation operation. Therefore, it is possible for a model to simultaneously rotate its parts, change position, and rotate around itself. The employment of direct manipulation techniques in the user interface is the main source of the power of Karagöz. Karagöz supports virtually any number of different models, each with virtually an unlimited number of primitive objects (thus, drawing parameters). If direct manipulation were not used, we would have to supply the user with a number of user interface objects (such as sliders or input-fields) that could provide the user with a means of indirect manipulation. For example, for rotating the leg of the Hacivat model in Figure 3, the user would have to enter the desired angle (by moving a slider or entering the angle value in an input-field) till he could obtain the posture he intended to produce. This would be a very time-consuming process, and contradict with the design goals of the Karagöz animation editor. It is also very difficult to nicely tie the user interface components and the on-screen representations of the models. Such an attempt would very likely result in a limitation in the range of models that could feasibly be supported, and it would also decrease the degree of integration between the two components of Karagöz.

Implementation

Most features of the Karagöz animation system described in the previous sections have been implemented using OpenGL [6] and XForms [8] library, on X workstations.

This section will provide some information on the fine-points of the implementation and on the experience obtained. In the current Karagöz implementation, the OpenGL matrix stack is employed in the first and the last steps of the tree-traversal operation described previously. The

current implementation compiles and runs on SGI workstations and PCs running Linux. Recall that while the transformations are being done or undone, the effects of the transformations previously applied must be preserved. Using the matrix stack, the first step (applying transformations) of the traversal of a subtree t rooted at a node n reduces into pushing a transformation matrix m into the matrix stack. The pushed matrix stays in the stack until the traversal of the subtree t is completed, that is, until the last step of the processing of node n. This ensures that all subtrees of subtree t (all simpler objects that constitute the aggregate object described by subtree t) have the transformations described by matrix m applied. This is the prime requisite for implementing hierarchical modeling. When the traversal of subtree t is completed, the matrix m is popped into the stack.

The fact that OpenGL is solely a "graphics library" (i.e. very little support is provided for user interaction) has affected the implementation of direct manipulation severely. For a quality implementation, clicks on the primitive objects on the screen must be captured correctly, that is, boundaries of the primitive objects should be checked. For a general implementation of this, the feedback mode of OpenGL is used. When this mode is used, OpenGL does not render the objects actually on the screen, but it returns information on how they would have been rendered. Using this information, the Karagöz user interface code determines whether a mouse click is inside the boundaries of one of the primitive objects. After this phase, the OpenGL renderer is invoked once more, this time in rendering mode, for displaying the objects on the screen.

Actually, the method of capturing the logical mouse click locations may seem ugly at first, since for each frame that is to be displayed, the rendering process is executed twice. However, a closer inspection of the situation at hand clearly indicates that this is not so unpleasant, since the objects that are to be rendered on the screen are fairly simple and two dimensional. Also, the availability of hardware acceleration for the rendering process is another relief for us. Figure 6 gives still frames from two different Karagöz animations. Sample animations can be found at http://www.cs.bilkent.edu.tr/~gudukbay/karagoz.html.

Conclusions and Future Work

This paper introduced an animation system for modeling and animating Karagöz, the Turkish shadow theatre. The system uses hierarchical modeling to construct and animate two-dimensional articulated characters containing body parts and joints between these body parts. Texture mapping is used for rendering the characters. To animate the models, we use keyframing based on the model parameters of the characters. These model parameters include the positions of some body parts and the joint angles between different body parts.After the user defines the keyframes, the system interpolates these keyframes and displays the characters by rendering them using texture mapping to produce animation. The animation system works as an authoring tool to create keyframe animations involving these characters by editing the character parameters for different keyframes.

Figure 6: Still frames from two different shadow plays.

(a) One of the unending conversations Karagöz and Hacivat.

(b) Celebi is wooing a Zenne.

The animations can be played back by reading the animation parameters for each keyframe from disk and then interpolating between the keyframes. The interpolated frames are rendered using texture mapping. The most important feature of the system is its ability to support arbitrary characters. This is achieved by a combination of a generic hierarchical model renderer, and a user interface that employs direct manipulation techniques.

There are possible future extensions to this work.

The real shadow theater is performed by using sticks attached to different parts of the characters and these sticks are used to move the parts of the models. These sticks could be simulated by binding them to different keyboard/mouse buttons to interactively animate the models as in the real shadow theater. This may enable an operator/artist to give live performances.

Sound-track: Dialogue, sound effects and music are components that are crucial to Karagöz. Currently, the software does not include these features, so a sound-track must be added in the post-production stage, using a standard sound editor. A music composer, effects library and dialogue program could be integrated into the software.

Acknowledgments

The characters used in the animations are scanned from Hayali Kücükali Shadow Play Collection of the Turkish National Library and from the book *Dünkü Karagöz* by Ugur Göktas, Akademi Kitabevi, 1992. http://www.cs.bilkent.edu.tr/~gudukbay/home.html

We are grateful to Fatih Erol for implementing an earlier version of the animation system for shadow theater and to Nezih Erdogan for valuable comments.

References

[1] And, M., *Karagöz - Turkish Shadow Theatre*, Dost Yayinlari, 1975.

[2] Diker, V.G., "The Stock Characters in Karagöz", http://www.ie.boun.edu.tr/assist/diker/stockchar

[3] Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F., *Computer Graphics: Principles and Practice*, Second Edition in C, Addison-Wesley, 1996.

[4] Göktas, U., *Dünkü Karagöz*, Akademi Kitabevi, 1992 (in Turkish).

[5] Heckbert, P., "Survey of Texture Mapping", *IEEE Computer Graphics and Applications*, Vol. 6, No. 11, pp. 56-67, Nov. 1986.

[6] Neider, J., Davis, T. and Woo, M., *OpenGL Programming Guide*, Second edition,

Addison-Wesley, 1997.

[7] Shneiderman, B., "Direct Manipulation: A Step Beyond Programming Languages", *IEEE Computer*, Vol. 16, No. 8, pp. 57-63, 1983.

[8] Zhao, T.C. and Overmars, M., "Forms Library: A Graphical User Interface Toolkit for X", http://world.std.com/ xforms/.