Hierarchical Modeling, Manipulation and Animation of Human Body

CS 184: Foundations of Computer Graphics

Final Project

By Johnny Chang and Yitao Duan

Fall 1999

Abstract

In this project we built a hierarchical model of a human body out of a set of unorganized polygons. Necessary information was inferred from the original data to define the relationship among body parts and enable user manipulation. Special efforts were made to ensure correct modeling and rendering of joints. Animation was achieved by recording a sequence of frames and replay them. Although this is basically a modeling project, we did our best to refine the rendering. We implemented a surface normal calculation algorithm proposed by H. Hoppe[2] and precalculated surface normal. Using this information, we did a smooth shading rendering.

Background

As part of an ongoing project which simulates human thermal comfort, it is critical to obtain a good geometric modeling of human body, no only for visualization purpose but also for accurate computing of radiant factors among body parts and the environment. It is desirable if the model can be easily modified so different gestures can be simulated. Currently the geometric model of body is generated outside the simulation program and is composed of a large number of fine, independent polygons. While it is adequate for one configuration, it lacks the flexibility to accommodate other situations. The body parts cannot be moved individually so it is frozen in one gesture. Our project solved this problem by constructing a hierarchical model out of these unorganized polygons. Different body parts are identified by their color tags in the data file(in DXF format).

Implementation

General Structure

The basic structure of hierarchical model in computer graphics is a directed acyclic graph(DAG). In our application, however, since we have both left and right limbs, we do not have to create them from one and copy to the other side. Therefore the structure of our model is simplified to a tree as shown in figure 1.

Transformations

In the hierarchical model tree, each node needs a transformation which transforms it from its local coordinate system to that of its parent's. In our model, the initial transformations were set to identity since all body parts are connected correctly at the beginning. When user identifies a body part and specifies a manipulation, its transformation will be modified accordingly. When applying the standard recursive hierarchical model tree rendering routine, this effected body part will be moved in the way user desires.

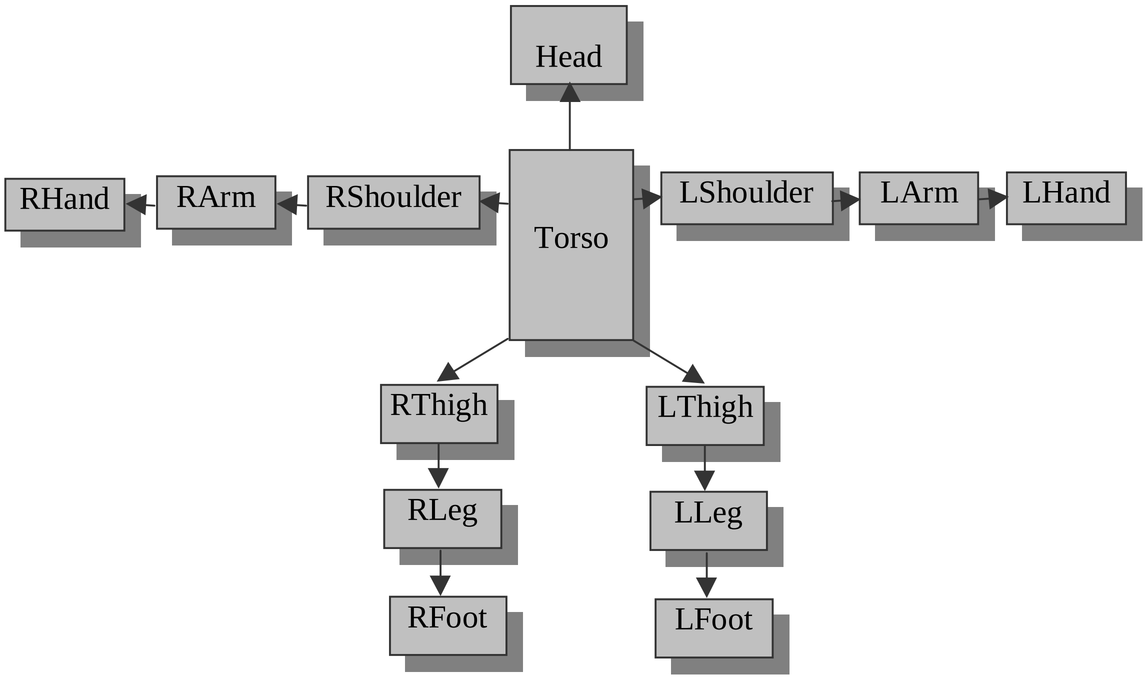


Figure 1 General structure of hierarchical model tree.

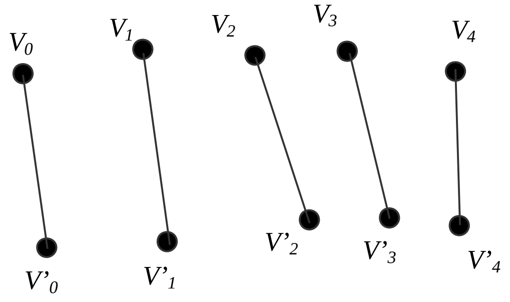
Blending of two neighboring segments:

Body parts are initially connected seamlessly. When one segment is moved, e.g. being rotated about an axis passing through its joint, since all body parts are treated as rigid body except in the twisting case which will be described later, a gap will appear between them. The following method is used to blend these two parts:

1. Processing the vertices of these two parts and identify those shared by both. They will be referred to by common vertices.
2. Determine joint by finding the centroid of the common vertices between these two neighboring segments.
3. Initially these common vertices are really "common" in the sense that both segments share them. However, when one is moved, common vertices on its side must reflect the same move so they are not really common anymore. We keep a record of the initial common vertices in a node. The moved common vertices can be found by applying its transformation on the initial record.
4. Once we identify the moved and unmoved "common" vertices, we can connect them and construct a surface.

The 4th step requires special care because we do not know the connectivity of these common vertices. If we did not find the connectivity information, all we can do is connecting the corresponding vertices and get a set of lines. The situation is shown in figure 2.

Figure 2. Without information of connectivity among *Vi* and *V'i*



What we desire, however, is a tessellated surface, e.g., a triangularized patch shown in figure 3.

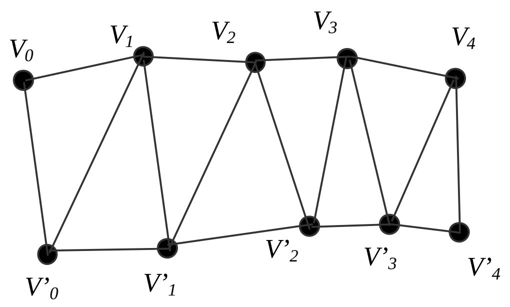


Figure 3. Tessellation with information of connectivity among *Vi* and *V'i*

To do this we must find the connectivity among the common vertices. Inferring topological information from a set of arbitrary data points is quite a hard task itself. Fortunately, in our application, we can utilize the safe assumption that our data points(common vertices) form a somewhat regular circle-like polygon. Base on this, we use the follow algorithm to find connectivity:

1. Find the controid *O* of all these vertices. Construct vectors from *O* to each vertex.
2. Project all vectors onto a plane *P*, e.g., the plane formed by the first two vectors. Call the projected vectors u_i.
3. On the plane *P* sort these u_i's according to the angles they form with a fixed reference vector, e.g., u_0.
4. Arrange common vertices in the same order as their corresponding projections.

Figure 4 shows the idea.

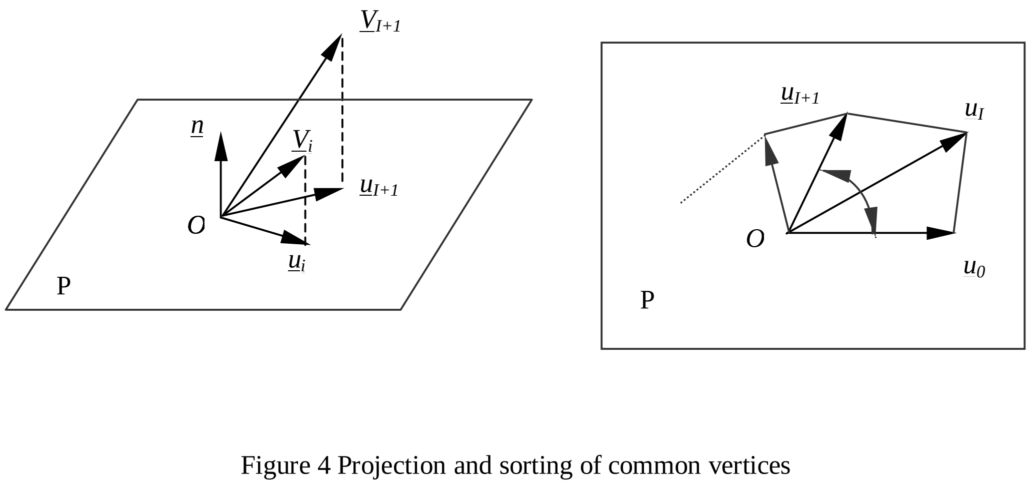


Figure 4 Projection and sorting of common vertices

Picking

Picking is an input operation that enables the user to identify an object on the display. Picking is considerably difficult to implement in OpenGL, as well as other modern systems, due to their nature of rendering pipelines. Primitives are defined in an application program and pass through a sequence of transformations and clippings until they are rasterized in frame buffer. Although much of this process is reversible mathematically, the hardware is not. Hence converting from a location on the display to corresponding primitive is not direct calculation. There are also potential uniqueness problems.

We use OpenGL selection mode to deal this problem. The idea is to define a small clipping region near the cursor and render the scene in GL_SELECT mode. Objects generating a “hit” with the clipping region will be returned. Process this information and we can identify the object being selected.

Smooth Rendering

Smooth shading needs surface normal for each vertex. These information were calculated using an algorithm described in H. Hoppe[2]. Since we are manipulating the body, the normal should be applied same transformation as that of its associated vertex. This is especially important for joint surfaces since they are newly generated and keep changing.

Manipulation

Due to the time constrain and the complexity of the problem, we are not dealing with inversed kinetics issues. Making a natural posture is up to the user.

The manipulations supported include rotation, translation and twisting.

Rotation

Each body part can be rotated about an axis passing through its joint with its parent. The direction of this axis depends on the desire of the user. User specifies rotating direction by different keys. Three degrees of freedom are provided for each joint.

Translation

Our model has the capability of translating each body part individually. Although this does not make sense in real life(You are taking an arm off the guy!), it provides a useful way to check that the joint surfaces are constructed correctly. This will be disabled for end users. The whole body, however, can be translated by picking up torso and press appropriate keys(Please refer to instruction section).

Twisting

Twisting is defined as rotation of body parts about its bone which, in our model, is the line connecting its joint and that of its child. One can only twist a body segment which has one and only one child, e.g., arms and legs. Hand and torsu, however, cannot be twisted since the former has no child and the later has 5. This conforms with the natural structure of body and its physical constraints. To model twist, the following are observed:

1. While twisting a body part, the points on the part rotate about bone by an angle proportional to its distance to the joint. That means the part has to be deformed.

2. The child of twisted part must rotate about the same axis, by the same angle as the end of the twisted part. This is to ensure the continuity between the two parts.

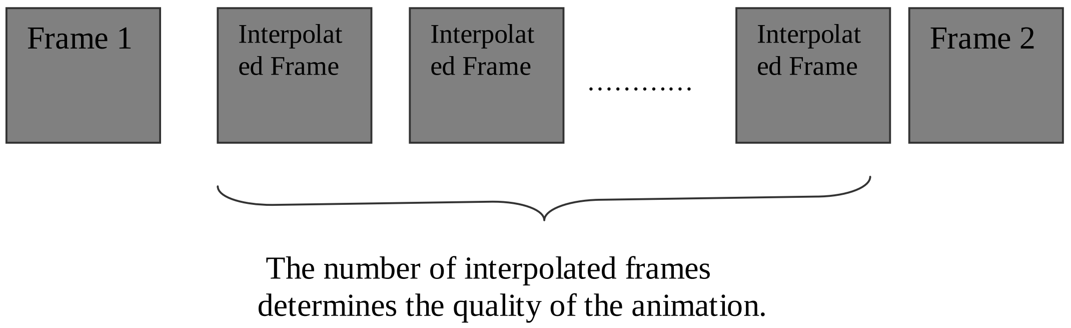
3. Vertices on joints must be connected correctly.

It turned out the third is the trickiest. The twisted body part must update its child's record of common vertices since they have been rotated, by different angles, while it is twisted. Its child, however, still needs the information of these vertices' original coordinates because it has to transform, by applying on them its own transformation, from there to their final position. This is necessary to close the gap and form a reasonable joint surface. To solve this dilemma, we keep too copies of "common vertices" in each node. One is a list of vertices on its parent's side, the other contains vertices on its own side. Initially they are identical. Twisted part updates only its child's vertices on its side. The sorting algorithm described above, on the other hand, must be applied on both sets of lists.

At this point twisting is no supported in animation. Twist will be shown in flat shading to illustrate the deformation effect.

Animation

We added the animation feature by allowing the user to create a sequence of frames. The program will do its best to interpolate the animation with a sequence of frames. The program records the difference in rotations and translations of the body parts. Then, it interpolates the animation by creating new frames with small changes.


Frame 1
Interpolated Frame
Interpolated Frame
…………
Interpolated Frame
Frame 2
The number of interpolated frames determines the quality of the animation.

To save a frame, we need to record all the transformation matrices in the tree. To do that, we simply store the matrices in a *.amt file. To display a frame, we simply load the transformation matrices from the *.amt file back to the model tree. Since it is very hard to interpolate the animation when the adjacent frames are not very similar, the animation may be choppy in some case. However, since our project is mainly on modeling, we feel the animation feature we have now is sufficient.

Results

The results produced by this project are quite satisfactory. The geometric data are very accurate and they are being used in the real project mentioned in Background section. The results verified their accuracy.

The implementation of the algorithms described above is successful. The following figures show the progress we made at each step.

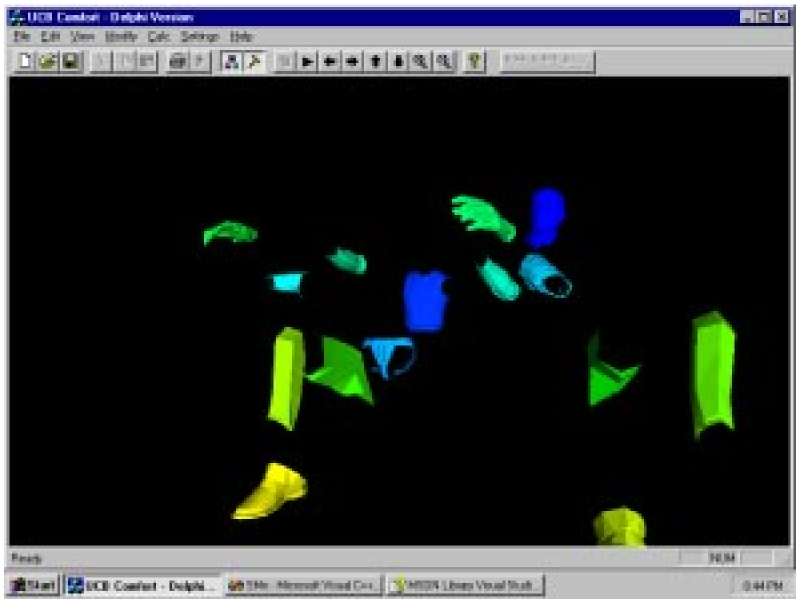

UCB Comfort - Delphi Version
File Edit View Modify Calc Settings Help
Ready
NUM
Start

Figure 5 Model with each body part identified and separated

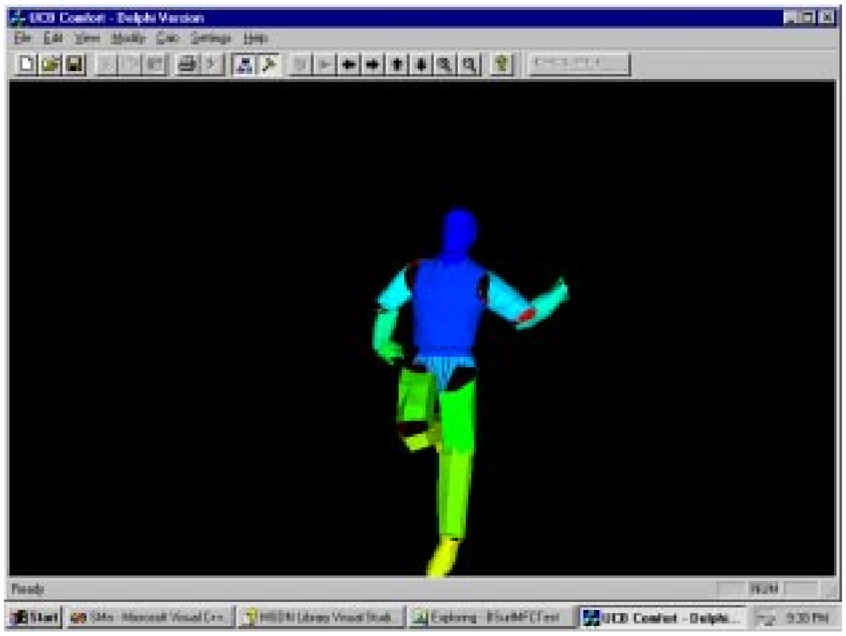

UCB Comfort - Delphi Version
File Edit View Modify Calc Settings Help
Ready
NUM
Start

Figure 6 Model with joint identified and limbs moved

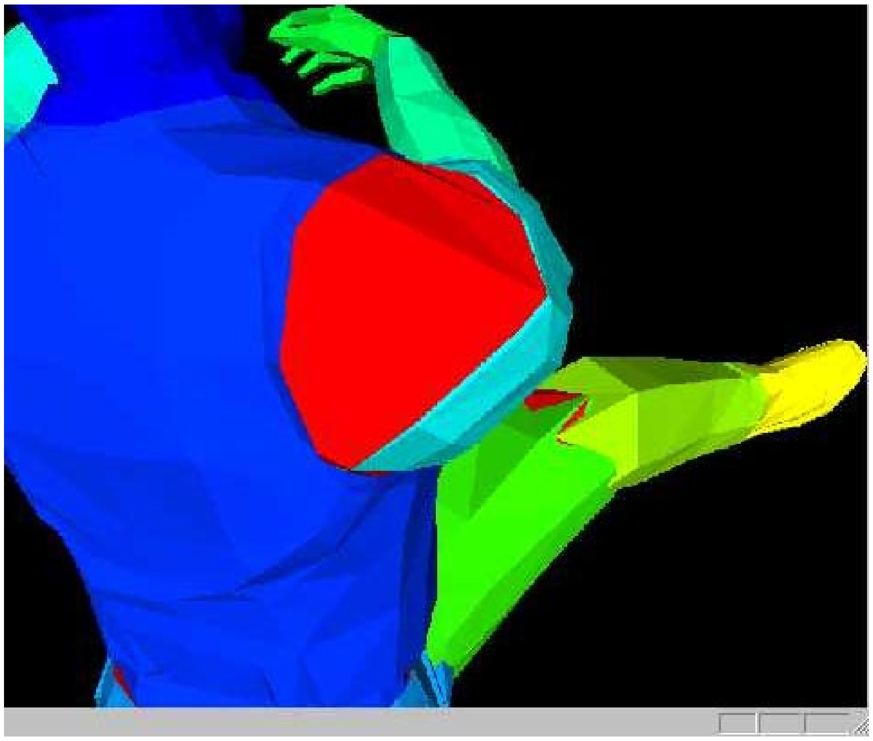


Figure 6 Joint surface constructed

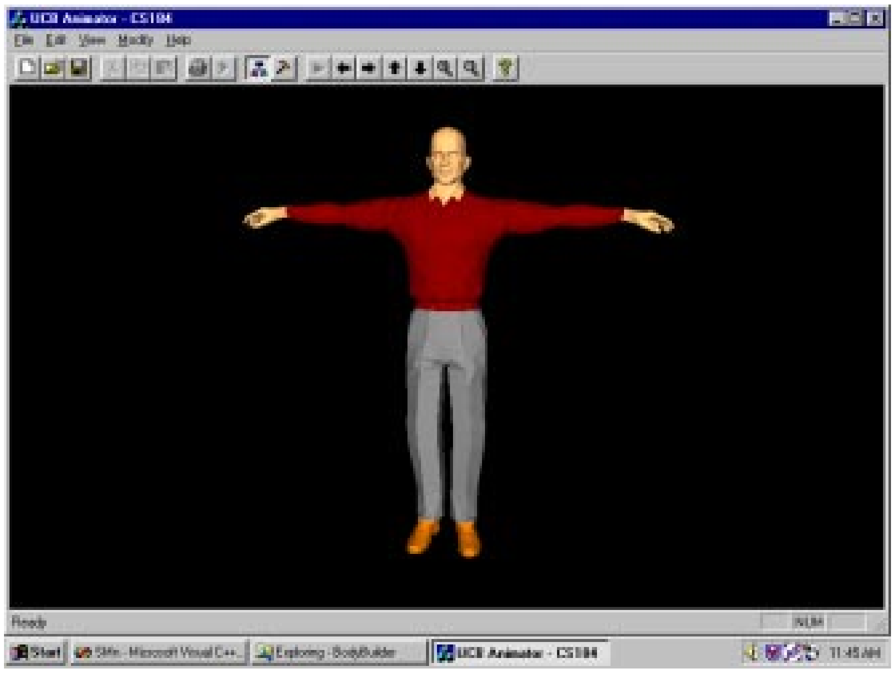


Figure 7 Dressed up

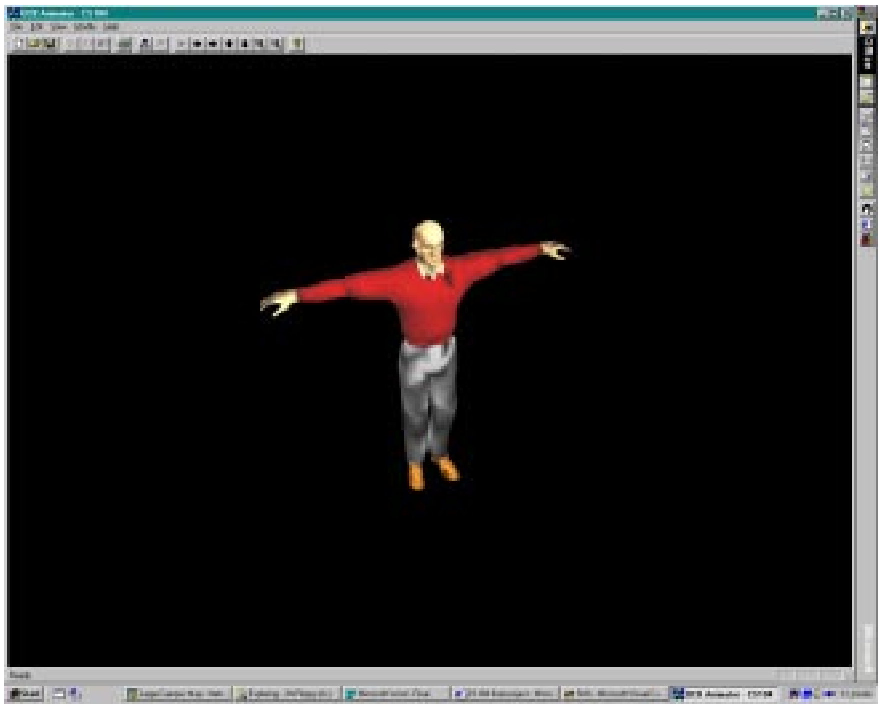


Figure 8 Smooth shading

Conclusion and Future Work

This project achieved its goal in that, as a modeling project, it produced correct data with adequate accuracy. It provides an interactive interface for manipulating the model to user-desired posture. The data it produced are being used in the simulation project and, conceivably, can be used in other applications too, such as rendering programs.

At this point the model is not too sophisticated. For example, we are modeling the hands as rigid body and the fingers cannot be manipulated. One big improvement would be building hierarchical model for these fingers and enabling fingers manipulation.

Currently manipulations are best recorded by keyboard because mouse stroke is sometimes inefficient in specifying movement in 3D. A better defined mouse manipulation can be developed in the future.

Reference:

[1] "Computer Graphics", Second edition, Donald Hearn and Pauline Baker, Prentice Hall, 1997
[2] "Surface reconstruction from unorganized points. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. *Computer Graphics (SIGGRAPH 92 Proceedings)*, pages 71-78.
[3] "Interactive Computer Graphics, A Top-Down approach with OpenGL", Edward Angle, Addison-Wesley, 1997
[4] "OpenGL Programming for the X Windows System", Mark J. Kilgard, Addison-Wesley, 1996