COSC422 Advanced Computer Graphics
Programming Exercise 16
Character Animation

This programming exercise provides an overview of the important methods used for animating a rigged character model using its skeleton (joint hierarchy), bones and animation data.

CharAnimation.cpp:

The program displays a rigged character model "Dwarf.x" (Fig. 1). The model uses two textures "dwarf.jpg", "axe.jpg", and contains one short (2 secs) animation sequence. The left and right arrow keys can be used to change the camera's view direction.

Fig. 1.

As part of the previous exercise (Exercise 15), we developed a program (`SkeletalAnimation.cpp`), to display the animation sequence stored in a BVH file. A character animation algorithm uses a similar skeletal structure and animation parameters to generate joint angle transformations. For each "tick" (frame) of the animation sequence, we compute each joint's transformation matrix using position and rotation keys, and update that node's transformation matrix. This program also uses a similar method (please refer to the `updateNodeMatrices` function), with a few differences:

- A character model may, in general, contain more than one animation sequence
- There may not exist an animation keyframe for every tick, and the keyframes may not be uniformly distributed among ticks. See slides [11]-25, 26. The `updateNodeMatrices()` function performs a linear interpolation between two closest key values to compute the transformation matrix.

A sketch of the node hierarchy of the model is given in Fig. 2.

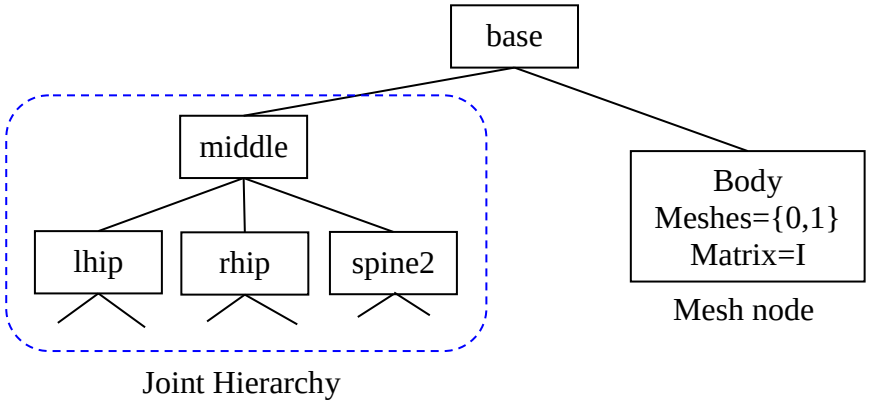

Fig. 2.

None of the nodes in the joint hierarchy contains a mesh index. The bones of a mesh specify a partitioning of the mesh corresponding to joints in the hierarchy. In the above example, Mesh[0] represents the axe model and contains a single bone named "weapon". Mesh[1] represents the character's body and contains 39 bones.

A simple structure `meshInit` is used to store the initial vertex coordinates and normal vectors of both mesh objects in the array `initData[]`. The data is stored in this structure when the mesh is loaded, in the `loadModel()` function (see also slide [11]-21).

Create a function `transformVertices()` as outlined in slides [11]-22, 23. This function should construct the transformation matrices required to transform the vertex set represented by each bone. The corresponding vertices and normal vectors from the initial mesh data stored in `initData` must be transformed and assigned to the character's mesh.

Include a timer call-back function that increments the tick value, calls the above functions, and refreshes the display:

```
void update(int value)
{
	if (currTick < tDuration)
	{
		updateNodeMatrices(currTick);
		transformVertices();
		glutTimerFunc(timeStep, update, 0);
		currTick++;
	}
	glutPostRedisplay();
}
```

Fig. 3 shows the initial and transformed configurations of the character model.

Fig. 3.

The model “wuson.x” (Fig. 4) contains two animation sequences, “walk” and “run”. To select an animation, set the value of the variable `animIndex` to either 0 or 1. Since the model is designed with the y-axis as the primary axis, please include a rotation about the x-axis and a translation along the y-axis in the display() function to get the model displayed in a proper orientation.

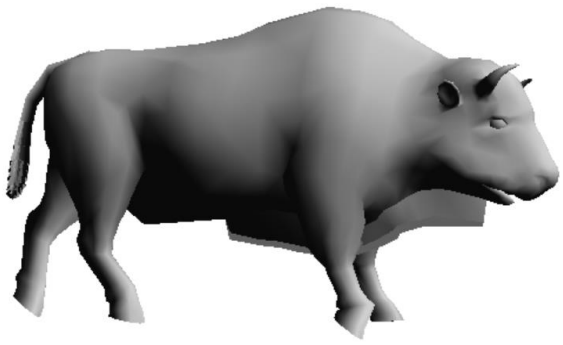

Fig. 4.

[11]: COSC422 Lecture Slides: "Lec11 Character Animation"