COSC422 Advanced Computer Graphics
Programming Exercise 12
Skeletal Animation

This programming exercise provides an overview of the implementation of fundamental skeletal animation techniques using Assimp. After completing this exercise, you will be able to animate skeletal structures using motion capture data stored in Bounding Volume Hierarchy (BVH) format. The Carnegie Mellon University's online motion capture database (http://mocap.cs.cmu.edu/) contains bvh files for various action sequences. Two bvh files, "Dance.bvh" and "Walk.bvh" are provided with this exercise. Another file, "Test.bvh" containing a very simple 3-link skeletal structure and a 10 frame motion sequence is also included (Slides [8]-14,15).

bvhLoader.cpp:

Please create a copy of the program `ModelLoader.cpp` (Ex. 11) and rename it as `bvhLoader.cpp`. We will not require any texture mapping operations, and therefore you may comment out or delete statements and functions used for texturing.

Please change the Assimp import post-processing preset to:

```
aiProcessPreset_TargetRealtime_MaxQuality | aiProcess_Debone
```

When Assimp loads a BVH file, it will create a mesh structure, the contents of which can be printed out by un-commenting the lines, "printSceneInfo()" and "printTreeInfo()" in the loadModel() function. Assimp generates a node hierarchy for the mesh objects with transformation matrices corresponding to joint offsets as shown in Slide [18]-21.

We will need the following three global variables for creating an animation sequence:

```
int tDuration;       //Animation duration in ticks.
int currTick = 0;    //current tick
float timeStep = 50; //Animation time step = 50 m.sec
```

The animation sequence is controlled by a timer function `glutTimerFunc(timeStep, update, 0);` where "update()" is the timer call-back function defined as follows:

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

}
```

The update function repeatedly generates a timer event by calling glutTimerFunc(), each time incrementing "currTick" until it reaches the value "tDuration". The variable "tDuration" must be assigned the value
`scene->mAnimations[0]->mDuration;`

The `updateNodeMatrices()` function updates the transformation matrices as shown in Slide [18]-22. The code for this function is given below:

```
void updateNodeMatrices(int tick)
{
	int index;
	aiAnimation* anim = scene->mAnimations[0];
	aiMatrix4x4 matPos, matRot, matProd;
	aiMatrix3x3 matRot3;
	aiNode* nd;

	for (int i = 0; i < anim->mNumChannels; i++)
	{
		matPos = aiMatrix4x4();   //Identity
		matRot = aiMatrix4x4();
		aiNodeAnim* ndAnim = anim->mChannels[i]; //Channel

		if (ndAnim->mNumPositionKeys > 1) index = tick;
		else index = 0;
		aiVector3D posn = (ndAnim->mPositionKeys[index]).mValue;
		matPos.Translation(posn, matPos);

		if (ndAnim->mNumRotationKeys > 1) index = tick;
		else index = 0;
		aiQuaternion rotn = (ndAnim->mRotationKeys[index]).mValue;
		matRot3 = rotn.GetMatrix();
		matRot = aiMatrix4x4(matRot3);

		matProd = matPos * matRot;
		nd = scene->mRootNode->FindNode(ndAnim->mNodeName);
		nd->mTransformation = matProd;
	}
}
```

You will now be able to load a file in BVH format and generate the animations using the timer function.

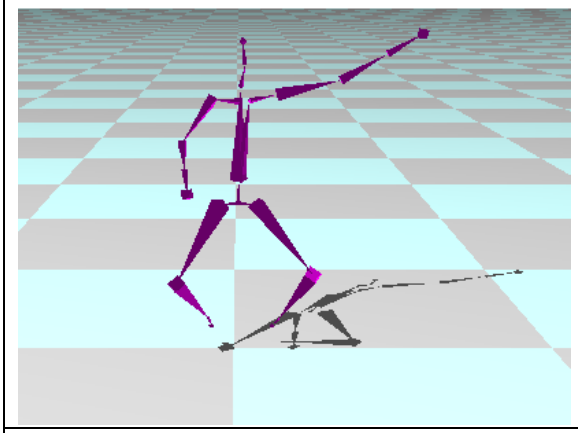
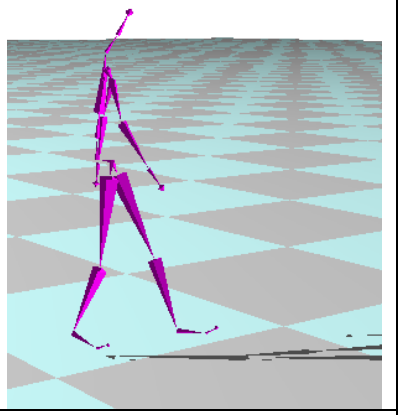
Programming considerations:

The model loader program centers the model within the view frame using the min, max values computed using the function "get_bounding_box". Motion capture data usually contains a global position offset with the keyframes (the position offset being associated with the root node). This offset will change the position of the skeleton when the animation sequence is initiated. Therefore, the model should be centered using the bounding box data obtained *after* updating the node matrices with the first keyframe (tick=0).

Please add a floor plane and planar shadows for the skeleton object. A simple method to generate planar shadows on the floor plane was discussed in COSC363.

Several animation sequences will involve translational motion of the skeleton (eg. walk, run sequences). For such animation sequences, the camera should follow the skeleton. The position of the skeleton can be easily tracked (how?)

Two sample outputs are given below:

	
Dance.bvh	Walk.bvh

[8]: COSC422 Lecture Slides: "Lec08 Motion Data and Skeletal Animation"