CS4458
Project 3 - motion capture
Due Oct 10, 5:00 pm

Overview

In this assignment, you will implement a motion capture viewer using WebGL. This involves playing a character animation that is loaded from a file. The file format is known as BVH, and it contains all data needed to animate the character. The parsing and building of the scene graph are provided.

The purposes of this assignment are:

1. Continue familiarity with WebGL, including using multiple programs (shaders) and model-view and projection matrices.
2. Understand hierarchical models, including local and world frames and how they relate to each other in the hierarchy.
3. Gain familiarity with camera control.

BVH File Format Loader

The file format used in this assignment is called BVH. It describes a character model using a simple utility language that describes a scene graph and the key frames needed to animate the model. Each scene graph description specifies the offset and degrees of freedom for each joint. The parser and in-memory scene graph are provided for you. But don't worry – there is plenty of fun left to be had.

Scene Graph Description

The parser for this is already written for you, but you will need to understand the BVH file format in order to know how to interpret the scene graph. A short description follows, but you might want to read a little more or download some other BVH files. You can download additional bvh files here:
`https://sites.google.com/a/cgspeed.com/cgspeed/motion-capture/3dsmax-friendly-release-of-cmu-motion-database`

An example of a scene graph description in a BVH file would be

Listing 1: BVH scene graph

```
ROOT root
{
        OFFSET 0 0 0
        CHANNELS 6 Xposition Yposition Zposition Xrotation Yrotation Zrotation
        JOINT lfemur
        {
                OFFSET 3.80421 −3.76868 0.00000
                CHANNELS 3 Xrotation Yrotation Zrotation
                JOINT ltibia
                {
                        OFFSET 0.00000 −17.76572 0.00000
                        CHANNELS 3 Xrotation Yrotation Zrotation
                        JOINT lfoot
                        {
                                OFFSET 0.00000 −16.34445 0.00000
                                CHANNELS 3 Xrotation Yrotation Zrotation
                                JOINT ltoes
                                {
                                        OFFSET 0.00000 −1.60934 6.00613
                                        CHANNELS 3 Xrotation Yrotation Zrotation
                                        End Site
                                        {
                                                OFFSET 0.00000 0.00000 2.66486
                                        }
                                }
                        }
                }
        }
}
```

The basic things that you need to know are:

1. There is one ROOT per file
2. Each `OFFSET X Y Z` corresponds to a translation by X, Y, and Z.
3. Each `CHANNELS` statement includes a number N, which indicates the number of channels, also known as the *degrees of freedom*. N is at most 6. The degrees of freedom are described by `(X|Y|Z)(rotation|position)` where `(X|Y|Z)position` corresponds to a translation, and `(X|Y|Z)rotation` corresponds to a rotation in degrees about the corresponding axis.
4. There will be anywhere from 1 to 6 degrees of freedom in a CHANNEL statement, but the position statements are only allowed in a root.
5. Degrees of freedom in a CHANNEL statement can appear in any order
6. `CHANNELS` and `OFFSET` are required for each `JOINT` and `ROOT`.
7. Each `End Site` only requires an `OFFSET`.
8. Joints are specified by `ROOT`, `JOINT`, or `End Site`.
9. The format is meant to be interpreted in the order that it is read. This means that transformations are performed in exactly the order they appear in the file

Key Frame Description

After the scene graph description, the key frame description is specified. This looks like the following

```
MOTION
Frames: 36075
Frame Time: 0.008333
<frame>
```

```
<frame>
<frame>
...
```

The first meaningful line tells you how many key frames are in the file. The frame time is specified in seconds. Each *frame* is a string of floating point numbers that specify the degrees of freedom for each joint read in the scene graph description. These numbers are in the order that the degrees of freedom were read in the scene graph description. There is one *frame* per line.

Scene graph

The scene graph is created for you in the parse() function of mocap.js and is stored in the `bvh` variable. You can look at BVH.toString() in BVH.js to see the structure. Look carefully at the comments inside of BVH.js as well.

Camera control

You will implement simple camera control:

- up arrow = zoom in
- down arrow = zoom out
- right arrow = orbit the camera left about the y axis with the origin as the center of rotation
- left arrow = orbit the camera right about the y axis with the origin as the center of rotation

Approach

Consider developing your code as follows:

1. Render a floor.
2. Render two spheres with a line between them.
3. Implement the camera controls.
4. Render balls at the joints of the character in `testData2`. See the `parse(testData2)` call near the end of mocap.js – `testData2` is defined in `test2.bvh.js`. Don't implement the channels yet – only deal with the offsets.
5. Render lines between the balls.
6. Now load testData1 (i.e. `parse(testData1)`) and implement channels. Render only the first frame.
7. Implement animation with all frames.
8. Implement extra awesome features. Document these features in README.txt.

For full credit:

- The animation is paused in the initial state, frame zero.
- Space bar plays and pauses the animation.
- The animation runs in realtime at the framerate specified in the bvh file.
- The animation loops once all of the frames have been played.
- Any bvh file loaded using the "Choose file" button renders correctly.
- Arrow keys control the camera as described above.

Extra features

Some ideas for extra features to pick up the last 10 points are:

- Enable speed up and slow down of the animation.
- Better character rendering: draw boxes for each limb instead of just drawing a stick figure,
- Shadows: some of the animations could use shadows, so it is easier to tell where the character is with respect to its surroundings.
- Multiple characters: if you go to the provided website, there are animations that include multiple characters, however the animation for each character is stored in a separate BVH file. You should load multiple BVH files and play them simultaneously. This will let you play scenes which depict two characters boxing or dancing.
- Key frame editor: provide a key frame editor. Allow the user to adjust the characters pose in order to change the individual key frames and then enter those key frames and then replay them.
- Character creator: allow a user to create joints and connect them in order to create a character and then be able to read that file back in and render and animate the character using your key frame editor.
- Texture and shading: if you have rendered the character using more interesting primitives than lines and points, provide some texturing and shading to make the character look more interesting.

Scoring

1. 30% - Render initial pose
2. 30% - Animation
3. 20% - Zoom and orbit camera control
4. 10% - Extra features
5. 10% - Coding quality and style (note that you are no longer required to use closure linter, although it is strongly encouraged)

Notes

1. For the basic features, you may not use any third-party libraries other than what is bundled in project3.zip. For the extra features, you may use third-party libraries and code, but you must document it in README.txt.