

CS 354
Project 3 - motion capture
Due Oct 12, 11:59 pm

Overview

In this assignment, you will need to implement a motion capture viewer using OpenGL.

This project involves playing a character animation that is loaded from a file. The file format is known as BVH. All of the data needed to animate the character is contained in this file. The parsing is provided for this project, but the tasks necessary to complete this project are to

1. create the scene graph data structure for the character,
2. update the key frames in order to animate the character, and
3. apply the necessary transformations in order for the animation to execute properly
4. implement extra camera control

Background

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. Some scene graphs will be provided for you for this assignment. The loader code is provided, but do not worry, because 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 assemble and traverse the scene graph yourself. So, you will need to understand a little bit about the BVH file format. A short description follows, but you might want to read a little more or download some other BVH files. You can go here:

<https://sites.google.com/a/cgspeed.com/cgspeed/motion-capture/cmu-bvh-conversion>
for that information.

It will be necessary to run `dos2unix` on these files before loading them. 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.

Interface

An interface has been created for you to receive the scene graph data. Please read `joint.h` to study this interface. This is realized in a class called `SceneGraph`, which you must implement. You must implement all of the methods in this class and may not change any of the signatures of the methods in this class.

The methods in the class will be called in a reasonable order, e.g a joint will be created before its child, the number of key frames and the size of each key frame will be provided before key frames are submitted, etc..

A few functions might need some clarification

- `setChannelFlags` specifies which channels are active for a given joint,
- `setChannelOrder` specifies the order in which the degrees of freedom should be applied, `order[i]` means that the channel `order[i]` is the i^{th} channel that should be applied. See `bvh_defs.h` to translate that into an actual transformation.
- `setFrameIndex` specifies the offset into the current frame to get the channel information for a joint.

Think carefully about the data structure you will create to represent the data and how this will allow the animation to play as it was meant to.

Caveat: copy all data passed in through this interface - do not assume that it is safe to just save a pointer, since this parser will reuse its own buffers as it loads the file. This parser passes the data into the `SceneGraph` class as it is read.

Camera control

The skeleton project includes 3 waypoint camera views. You switch between the three views by pressing '1', '2', or '3' on the keyboard. You will need to implement two types of camera control:

- 'z' = zoom in

- 'Z' = zoom out
- 'j' = orbit the camera left about the y axis with the origin as the center of rotation
- 'k' = orbit the camera right about the y axis with the origin as the center of rotation

Pressing a waypoint key ('1', '2', or '3') should always return the camera to an original waypoint position.

Rendering

The skeleton project renders an axis and a floor for the character to be drawn on top of. You will need to provide some minimal support for displaying the character and animating it. The character should be standing on the floor in its initial pose when the program is launched. You should also at least accomplish the following

1. display each joint at its correct location,
2. and draw the segments connecting each joint.

This will give you a stick figure character. All of the characters in the provided motion capture files will start in what is known as the T-pose. There will be extra credit for providing a better looking character.

Animation

The end result of this project should be that you are able to play back the motion capture data that is provided. The important things to think about are

1. to be able to understand how local coordinate systems and local frames work,
2. and to be able to traverse the data structure in order to obtain these local frames,

For full credit:

1. The animation should be paused in the initial state, frame zero.
2. It should be possible to play and pause the animation in real time using the space bar.
3. The animation should render in real time. `glutGet(GLUT_ELAPSED_TIME)` may be useful to you.
4. The animation should loop once all of the frames have been played.

Extra features

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

- 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.
- Enable speed up and slow down of the animation.
- 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.

Instructions

1. Compile the skeleton code by typing `make` at the command-line
2. Run the skeleton code by typing `./mocap data/01_01.bvh`.
3. Look for all `TODO` comments in `main.cpp` and follow the instructions. Also add code to `joint.cpp` and `joint.h` to build the scene graph.
4. `./check-code` is run automatically with each build. It is only run on `main.cpp` and `joint.cpp/h` as other files are either automatically generated or are pre-existing. If you choose to add files to your project you should modify the call to `check-code` in `Makefile` so that it picks up these additional files.
5. Add extra features.
6. Take a screenshot of your program using the following command: `import -window "Mocap" mocap.png`.
7. List all additional features in `README.txt`. If you do not list a feature here it will not be graded. Include instructions on how to use the feature (keyboard commands, menu options, etc).
8. Submit your work. Type `turnin --submit edwardsj project3 *` at the command-line. You are responsible to ensure that what is submitted is correct. A helpful tool is `turnin --verify`. Type `man turnin` at the command-line for details.

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 given by report from `check-code` and visual inspection