CS229 Project 2: 3D Orientations

Assignment Out:	Sept. 22th, 2000
Assignment Due:	Sept. 29nd, 2000 (11:59 pm)

1 The Assignment

The goal of this project is to help build some intuition about Euler angles, quaternions, and quaternion splines. For this assignment, you will create three different types of 3D angle interpolation:

- Linear interpolation of fixed axis angle parameters
- Spherical linear interpolation
- Quaternion splines

To make things easier to reason about, we'll use fixed axis angle parameters instead of Euler angles. Recall that a rotation about fixed axes x, y, and then z, for example:

$$M = R_z(\gamma)\ \ R_y(\beta)\ \ R_x(\alpha) \qquad \text{fixed axes} \tag{1}$$

is exactly equivalent to the same amounts of rotation about z, y, and then x when we have moving axes (or Euler angles):

$$M = R_{x''}(\alpha)\ \ R_{y'}(\beta)\ \ R_z(\gamma) \qquad \text{moving axes} \tag{2}$$

In the Euler angle example, the second axis of rotation would be:

$$y' = R_z(\gamma)\ \ y \tag{3}$$

and the third axis of rotation would be:

$$z'' = R_{y'}(\beta)\ \ R_z(\gamma)\ \ z \tag{4}$$

Keeping track of local axes y' and z'' can be burdensome, and so we will assume fixed axis angle parameters for this assignment. As with Project 1, you can just compose successive rotations about the world coordinate axes to form a composite rotation matrix.

Your program should read in a file of keyframed orientations. Each orientation will occupy a single line, and will consist of the *Rotation* keyword and a rotation about the x-axis, followed by a rotation about the y-axis, followed by a rotation about the z-axis. Each rotation also has a time associated with it, indicating which tick (frame) the rotation should be achieved on. These definitions look like:

```
Time t1
Rotation x_rot_1   y_rot_1   z_rot_1
Time t2
Rotation x_rot_2   y_rot_2   z_rot_2
Time t3
Rotation x_rot_3   y_rot_3   z_rot_3
...
```

Note that each time should be larger than the time before it or it will be ignored. Further, the file starts with header information specifying the number of keyframes, the letter (or word) to display for visualization purposes, and the starting position of that letter. This header information must come first with the format:

```
Number n
Letter l
Position x y z
```

For rotation at a given keyframe n, a point ${}^{L}\mathbf{p}$ in local object space is then converted to point ${}^{W}\mathbf{p}$ in world space as follows:

$$ {}^{W}\mathbf{p} = R_z(z\_rot\_n)\ \ R_y(y\_rot\_n)\ \ R_x(x\_rot\_n)\ \ {}^{L}\mathbf{p} \tag{5} $$

Your job will be to smoothly move the object through the complete set of keyframes. Exactly how to display the results is up to you, but try to provide enough context so that we can accurately visualize the motion. As an example, the demo follows the inspiration of Watt and Watt – it puts up a 3D letter R at some radius from the origin. The R then remains tangent to a sphere of that radius as its rotation matrix is modified based on the input data. The user can select any of the three types of interpolation and play back the results using the selected interpolation technique.

1.1 Linear Interpolation of Parameters

This part is easy! To move from frame m to frame m+1, just interpolate between `x_rot_m` and `x_rot_m+1` to find the new x rotation parameter, interpolate between `y_rot_m` and `y_rot_m+1` to find the new y rotation parameter, and interpolate between `z_rot_m` and `z_rot_m+1` to find the new z rotation parameter.

1.2 Spherical Linear Interpolation

Implement the quaternion equivalent of linear interpolation as discussed in Watt and Watt and in class. You will first need to convert your orientation into quaternion form. The `Quat4D` class should be helpful here. You may notice, in fact, that `Quat4D` has an `interpolate` method. Please **do not** use this method for your assignment. You will essentially be writing your own version. (Feel free to compare the results of this method with your results, however.)

1.3 Quaternion Splines

Create some type of cubic splines through the given keypoints. The only restriction here is that you should use the quaternion representation of orientations, not the fixed axis angle representation. As a bonus, try one of the clever algorithms presented at SIGGRAPH in the last decade. Kim, Kim, and Shin (SIGGRAPH '95 and in the course packet) is probably the most straightforward to tackle. Their quaternion Bezier splines, for example, are fairly straightforward once you work through their notation. Also check out Barr, Currin, Gabriel, and Hughes (SIGGRAPH '92) for a nifty approach involving optimization. (Of course the latter would also make a fine final project!)

2 Questions

In a README file along with your completed assignment, please answer the following:

1. Describe your algorithm for creating a smooth quaternion spline. What are the advantages and disadvantages of your approach? Are your splines C1 continuous? Make an argument for why or why not.

2. How would you modify your algorithm to control initial velocity along this quaternion spline? For example, with a high enough starting velocity, you should be able to make the object flip all the way around a number of times before passing through the first keyframe. (You don't have to actually do it, just explain how you would solve the problem.)

3. Suppose you are controlling a camera located at the origin. Instead of three orientation parameters, the input file contains the world x, y, and z keyframe positions of an object you wish to track. How would you control the orientation of the camera? You should try to minimize camera tilt as much as possible. Ideally, you should also get graceful behavior even when tracking an object that passes directly above the camera.

3 What We Give You

This assignment uses essentially the same user interface as the first one; as such, it is recommended that you adapt your viewer to use here, since you are presumably highly familiar with it. A simple viewer is provided in $/course/cs229/asgn/rotation/Viewer.java$ for those who wish to use it; additionally, it includes a few useful examples, such as the creation of 3D text.

A file parser is provided in $/course/cs229/asgn/rotation/CTNFileReader.java$ which converts the file into an array of KeyFrame objects. These are detailed in KeyFrame.java, and are essentially data structures with a few useful methods.

Example input files are provided in $input*.ctn$. Running these using both quaternions and fixed axis angles will illustrate a few of the differences between these representations.

Finally, a demo version is in $/course/cs229/demos/rotation/$; type *run* in that directory to view it.

4 Handing In

Run "$/course/cs229/bin/cs229.handin\ rotation$" from your project directory.