CSCI 520 Assignment 2: Motion Capture Interpolation

Due Wed Mar 12, 2025, by 11:59pm

Overview

In this assignment, you will implement three interpolation schemes to interpolate human motion data obtained from an optical mocap system. The human model (skeleton) is represented using a hierarchy, composed of a root node and several children nodes, corresponding to the various joints. The optical mocap system generates two data files: ASF file, which encodes the skeleton kinematics (lengths of bones, degrees of freedom, etc.) and AMC file, which stores the motion (translation, joint angles) in time (at 120 frames per second). The AMC file contains the translation of the root node (xyz coordinates), rotation of the root node (in Euler angles), and the joint angles (for the various joints, in Euler angles). We provide starter code that can parse ASF and AMC files and store the data into a datastructure. The starter code also includes an OpenGL motion capture player that renders the skeleton (parsed from the ASF file), and plays back the motion from an AMC file. You will be given several ASF files and AMC files corresponding to different motion capture sequences (walking, dancing, running, martial arts). For each motion sequence, you will then form a subsequence, by dropping certain frames (described below). You will use this subsequence to create an interpolated sequence of the original length, and analyze and plot the difference to the original motion.

Source: Wikipedia

Requirements

The AMC files use Euler angles to represent joint rotations. The starter code can load these Euler angles into a datastructure in memory. Your program should take the motion from the input AMC file, and then form the *keyframes* for interpolation. The keyframes are obtained from the original motion by dropping N consecutive frames, where N is a given input parameter. For example, suppose the input motion has 30 frames, labeled 0,1,...,29. Suppose N=6. Then, the keyframes are frames 0,7,14,21,28. Your program must use interpolation to compute new frames 1,2,...,6, 8,9,...,13, 15,16,...,20, 22,23,...,27. Set frame 29 to be identical to the input frame 29. Thus, your program produces a motion sequence of the same length as the original. Depending on the value N, and the choice of the particular interpolation technique, the interpolated motions will more or less closely resemble the input motion. For example, it is expected that quaternion interpolation produces motions that more closely resemble the input than Euler interpolation. You will analyze the different interpolation techniques in your report (more below).

The starter code can perform linear interpolation in Euler space (see interpolator.cpp), which also serves as an example for how to access motion capture joint data and create/set new motion sequences. Your task is to do the following:

- Implement Bezier interpolation for Euler angles.
- Represent the joint angles (including root orientation) using quaternions. A quaternion library is included with the starter code. This library can perform quaternion arithmetics, and also convert from quaternions to rotation matrices and back. You must write a routine that converts from Euler angles to rotations. Starter code already provides a routine that converts from rotations to Euler angles; in XYZ angle format. Then, use these routines to write routines that convert from Euler angles to quaternions, and back (see interpolator.cpp).
- Implement Spherical Linear (SLERP) interpolation for quaternions. For implementation details, see the course slides, Rick Parent's "Computer Animation" book (pages 97-102 (first edition), or pages 110-114 (second edition)), and Ken Shoemake's "Animating Rotation with Quaternion Curves" paper.
- Implement Bezier Spherical Linear (SLERP) interpolation for quaternions (see same references as above for help). Primarily, we recommend that you follow the material in Rick Parent's book.
- The Bezier control points a_n and b_n+1 must be pushed towards points p_n and p_n+1 (1/3 factor; see the notes and Shoemake paper). This applies both to Euler Bezier and Quaternion Bezier.
- With Quaternion interpolation, use Euler interpolation to interpolate the root position, as follows. Linear Quaternion should use Linear Euler for the root, and Bezier Quaternion should use Bezier Euler for the root.

Submission

Please submit the following:

- Your code which implements Bezier interpolation for Euler angles, as well as SLERP and Bezier SLERP interpolations for quaternions.
- A report comparing the four interpolation methods: linear Euler, Bezier Euler, SLERP quaternion, Bezier SLERP quaternion.
- Plot four graphs as part of the report (use any program you want, for example, Matlab, GNU Octave, GNU Plot, etc.). For each of these graphs, the X axis should be the frame number, and Y axis the angle in degrees. When using quaternion interpolation, convert the quaternion results back to Euler angles to plot the graphs. **All graphs should show three curves: the two techniques being compared, and the input motion.**

    - Graph #1 compares linear Euler to Bezier Euler interpolation (and input).
      Graph #2 compares SLERP quaternion to Bezier SLERP quaternion interpolation (and input).
      Plot graphs #1, #2 for **lfemur** joint, rotation around **X** axis, frames **600-800**, for **N=20**, for **131_04-dance.amc** input file.
    - Graph #3 compares linear Euler to SLERP quaternion (and input).
      Graph #4 compares Bezier Euler to Bezier SLERP quaternion (and input).
      Plot graphs #3, #4 for **root** joint, rotation around **Z** axis, frames **200-500**, for **N=20**, for **131_04-dance.amc** input file.
    - You can also include other graphs to demonstrate the points you made in your report.
- Three videos demonstrating your results (all three for **135_06-martialArts.amc**, **N=40**):
    - Input motion and Bezier Euler (superimposed on top of each other)
    - Input motion and SLERP quaternion (superimposed on top of each other)
    - Input motion and Bezier SLERP quaternion (superimposed on top of each other)

  All videos should use the **default camera settings** and should show the input motion in **red**, and the interpolated motion in **green**. Use the functionality to display two skeletons to create these videos (see "Displaying two or more motions simultaneously" below). The two motions should be shown on top of each other (tx=ty=tz=rx=ry=rz=0 on the UI). You can also include other videos to demonstrate the points you made in your report. You can submit the videos as a sequence of PNG or JPG images, or as an MP4 video. Please do not submit PPM images as those are big and difficult to store and transmit to the Blackboard. In Windows, you can convert using "batch convert" in the free program Irfanview. For Linux and Mac, use command "convert" in the terminal. You may need to write a script to do "batch convert".
- In addition to the graphs and the videos, your report should contain a narrative giving your findings and observations. Which techniques did you successfully implement? How well does each of the techniques work? What are the strengths and weaknesses of each technique? Include any additional findings, interesting observations or insights gained during this homework. Also, describe any extra credit implemented.

Please submit the assignment using the USC Blackboard. In order to do so, create a single zip file containing your entire submission. Submit this file to the blackboard using the "Attach File" field. After submission, please verify that your zip file has been successfully uploaded. You may submit as many times as you like. If you submit the assignment multiple times, we will grade your LAST submission only. Your submission time is the time of your LAST submission; if this time is after the deadline, late policy will apply to it.

Note: The instructors will read your code. Quality of your code and comments might affect your grade. This is an individual assignment. Your code might be compared to code of other students using a code comparison tool.

Starter code and data

You can download the starter code and data here.

Code setup instructions

```
=====================================================================================
How to build fltk in Windows, Linux, MAC OS X
=====================================================================================
Please follow the corresponding readme files in the folder: fltk-1.3.8.
Note: On Linux, you may need to install the "autoconf" tool.

Note: There are many projects in FLTK library. This homework
is only dependent on fltk and fltkgl projects. Students can choose
to only compile those two projects in the Visual Studio project
of the FLTK library.

Note: If you want to compile the homework under the release
configuration, you should first compile the FLTK projects in release.
Same goes for the debug configuration.

Note: Both the main homework project and FLTK projects work only
on the Win32 platform. Please do not use other platforms (e.g. X64).

=====================================================================================
How to build starter code in Linux, MAC OS X (Assuming fltk has been compiled)
=====================================================================================
1) Enter the %homeworkFolder%/mocapPlayer-starter
2) make

=====================================================================================
How to build starter code using Visual Studio 2017 (Assuming fltk has been compiled)
=====================================================================================
1) Open the project file in homework folder: IDE-starter/VS2017/mocapPlayer.sln
2) Choose Debug/Release mode
3) Compile project: mocapPlayer
4) Compile project: interpolate

Because there are many versions of Windows 10 (different build versions),
you may get an error when compiling: ... selecting "Retarget solution".
This can be solved, as the message says, by right-clicking the solution
and selecting "Retarget solution".
```

Source code structure and description

The code consists of several classes. We only provide a brief description here. For more detailed description of each class, see the corresponding .h files. When the "Load Skeleton" button is pressed, the selected ASF file is parsed, and skeleton loaded is into memory and saved into an instance of the skeleton class. When the "Load Motion" button is pressed, the AMC file is read and saved into an instance of the motion class.

- *Skeleton class (skeleton.h)*: read an ASF file and build the skeleton hierarchy. See the description of member functions and variables in "skeleton.h" file. When the skeleton is read, a hierarchy of bones is constructed. Each node in the hierarchy represents a bone as shown in Figure 1. Each node has exactly one child. If in the skeleton file (ASF) a particular bone has more than one child (for example, the root), the node corresponding to the second child is set as a sibling of the node of the first child, the next child as a sibling of a previous child and so on.
- *Motion class (motion.h)*: read/write an AMC file and store joint angles for each frame in a memory array (m_pPostures). Each posture defines the root position and orientations for all the bones (including root orientation) at a particular frame. For more details, see "motion.h" file.
- *DisplaySkeleton class:* renders the skeleton at a given posture.
- *motionPlayer.cpp*: the player itself, with the user interface and OpenGL initialization and render settings.
- *transform.cpp*: some math functions for transformation
- *Vector class*: 3d vector representation and operations (addition, subtraction, ...). See "vector.h" file for the full description.
- *Interpolator class*: currently supports linear interpolation using the Euler angle representation. This is the class where you need to do the vast majority (if not all) of the programming in this assignment. You first need to initialize the interpolator by setting the interpolation type (LINEAR or BEZIER), and angle representation (EULER or QUATERNION). Then the "Interpolate" function can be called to create interpolated motion. See "interpolator.h" file for more information. You will need to add functions to this class to do the kinds of interpolation you were asked in the assignment.

Description of motion capture ASF and AMC file formats

The starter code reads and displays the data from the optical motion capture system (ASF and AMC). The world coordinate system is Y up.

Skeleton file

The *skeleton file* (ASF) describes how the bones in the skeleton are connected, and their degrees of freedom (see Figure 1 for a representative skeleton). All bones are described in the ":bonedata" section of this file. Each bone has the following fields:

```
begin
id bone_id              //Unique id for each bone
name bone_name           //Unique name for each bone
direction dX dY dZ   //Vector describing direction of the bone in world coordinate system
length 7.01722          //Length of the bone
axis 0 0 20 XYZ          //Rotation of local coordinate system for
                        //this bone relative to the world coordinate
                        //system. In AMC file the rotation angles
                         //for this bone for each time frame will be
                        //defined relative to this local coordinate
                         //system (see Figure 2)
dof rx ry rz           //Degrees of freedom for this bone.
limits (-160.0 20.0)
         (-70.0 70.0)
         (-60.0 70.0)
end
```

The parent/child relationship among all bones in skeleton hierarchy is defined in the ":hierarchy " section of the ASF file. Each line list the parent first, then its children.
For example:
"*root lhipjoint rhipjoint lowerback*"
Root is a parent and lhipjoint, rhipjoint, lowerback are root's children.

The following picture shows the skeleton hierarchy as defined in a representative ASF file (length of the bones is not up to scale).

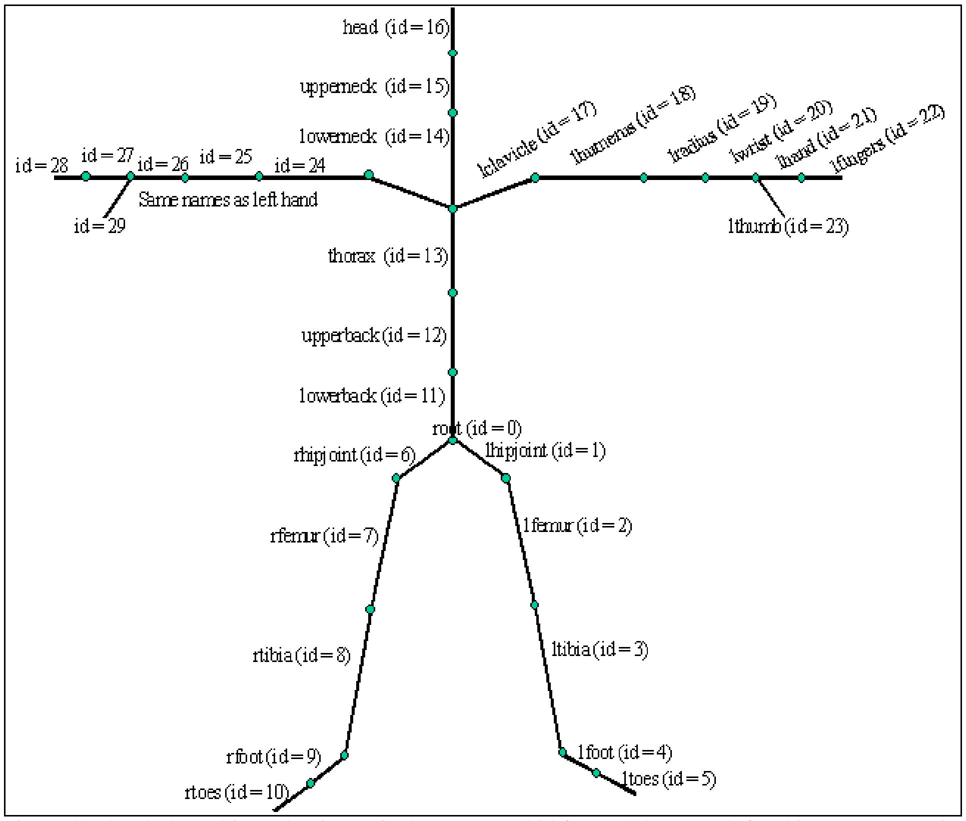


Figure 1. The skeleton hierarchy, in particular name and id for each bone as defined in a representative ASF file.

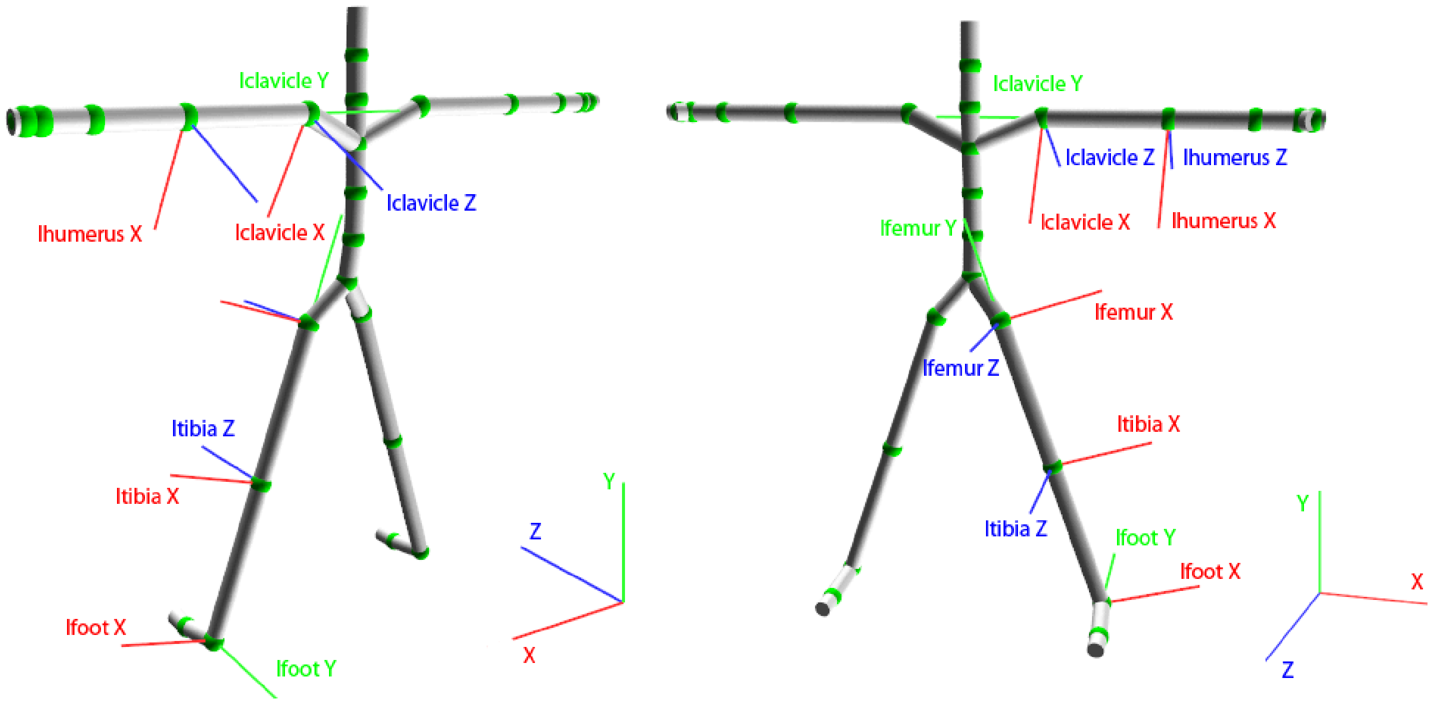


Figure 2. The names and local coordinate system for some of the bones.

For a longer introduction on the mocap file format, you can look at http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/ASF-AMC.html.

Motion file

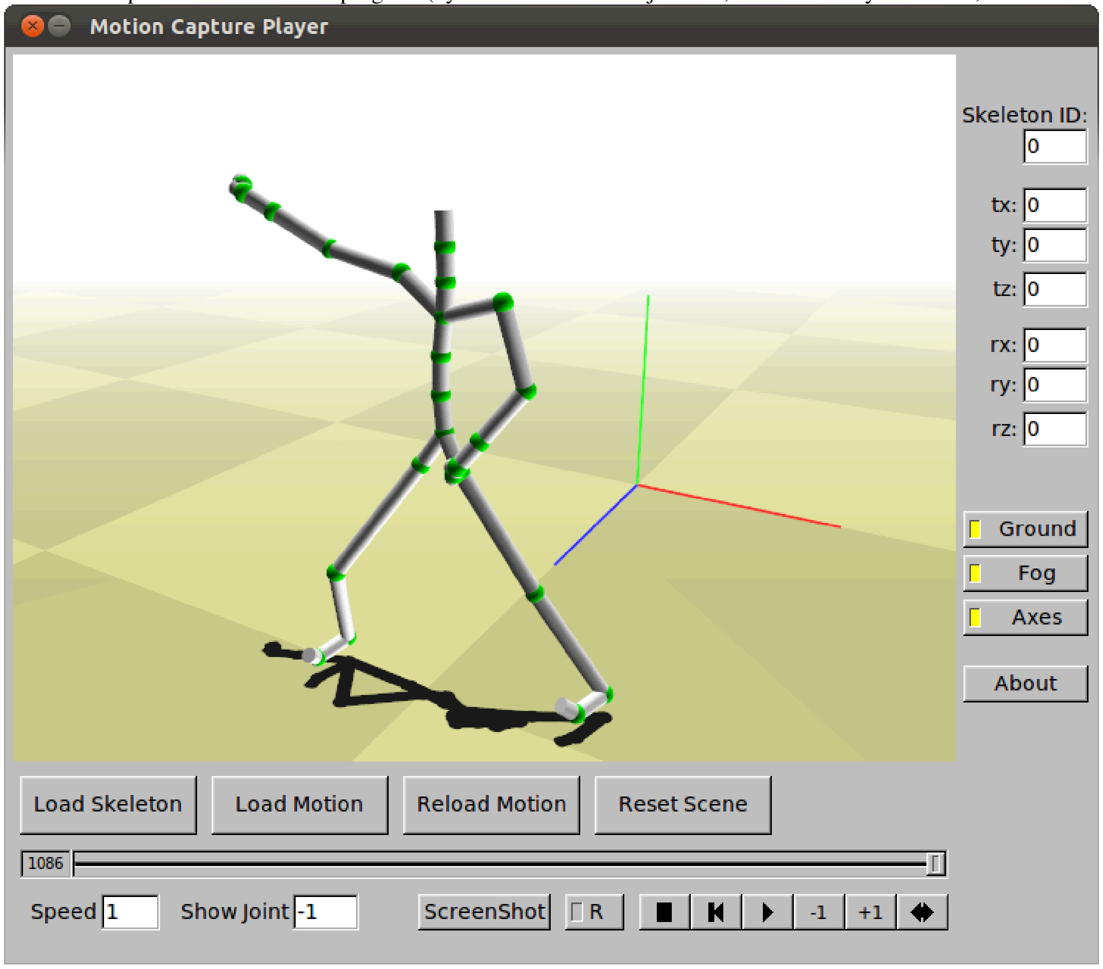
The *motion file* (AMC) gives each bone's rotation angles (in Euler angles) relative to the bone's reference local coordinate system. The bone's reference local coordinate system is defined in the skeleton (ASF) file. One entry is given for each frame, at a frame rate of 120 fps.

The ":FORCE-ALL-JOINTS-BE-3DOF" keyword in the header (if present), is not a standard AMC keyword. We added this keyword because the result of quaternion interpolation may produce three-dimensional Euler angles, even when original Euler angles were only one or two-dimensional. You do not need to worry about this keyword. The "interpolate" driver (starter code) will create it automatically as needed, and the "mocapPlayer" will correctly load any motion files that use this keyword.

Motion capture player instructions

The "mocapPlayer" player can play back any motion capture sequence (AMC file), on any skeleton (ASF file). A simple camera control is provided. You can drag the left mouse button to move the camera, middle mouse button to zoom in/out, and the right mouse button to rotate the camera.

Below is the picture of GUI for the program (by Yili Zhao and Jernej Barbic; first version by Steve Lin, Alla Safonova, Kiran Bhat).



Displaying the animation

You first need to load the skeleton (ASF) file. Then load the motion (AMC) file. You can now view the motion using the player buttons. You can save the current frame as an image by pressing the Screenshot button. To record a sequence of images (to make a video), first press record (R) button, then play the animation. All the frames played while the record button is enabled will be saved to disk as images.

Displaying two or more motions simultaneously

You can also display two motions on top of each other, say, to visualize the difference between the input motion and the interpolated motion. In order to do so, first load the first ASF file and AMC file, then load the second ASF file (may be same as first, but must be still loaded), and the second AMC file. You will see the two motion on top of each other.

Adjusting playback speed

By default, the player plays the motion at normal speed, that is, 120 frames per second. In order to slow down the motion, enter the corresponding value into the "Speed" box. For example, to make the playback 5x slower, enter 0.2 into the "Speed" box.

Displaying the frame of a joint

You can display the frame of a joint by entering its index (positive integer) into the "Show Joint" textbox. The default value is "-1" (no joint shown).

Interpolation

Use the provided "interpolate" driver application to perform the interpolation. This driver loads a skeleton and motion file, performs interpolation, and outputs the resulting AMC motion file. You can then load this AMC file into the mocap player for visualization. You can use the "Reload" button to reload the AMC motion (useful, for example, when tweaking your interpolator class). See the command line options for how to execute various interpolation techniques. For example, to perform Bezier Euler interpolation, using N=10, use:

```
./interpolate 131-dance.asf 131_04-dance.amc b e 10 131_04-dance-be-N10.amc
```

Tips

- Q: How to convert from a rotation matrix to quaternion?
  A: Starter code already provides this conversion (quaternion.cpp).
- Q: How to convert from a quaternion to rotation matrix?
  A: Starter code already provides this conversion (quaternion.cpp).
- The matrix given in Shoemake paper for converting from quaternion to matrix has different signs than the matrix given in the Parent book. However -- you need not worry about this -- simply use the routine in our provided Quaternion class.
- Q: How to convert from a rotation matrix to Euler angles?
  A: Starter code already provides this conversion (interpolator.cpp).
- Q: How to convert from Euler angles to a rotation matrix?
  A: See the course slides.
- Q: How do I compute a_n and b_n control points for Bezier curve using the quaternion angle representation?
  A: The formulas for computing a_n and b_n using quaternion representation are given in the Shoemake paper (page 249).
- Q: Bezier for Quaternions does not work correctly in my implementation. Are the formulas in the book/paper correct?
  A: The formulas are correct. We implemented the algorithm using the formulas in the book/paper and it works.
- Make sure you read paragraph 4.4 on page 250 in Shoemake paper. When you compute a_n, you can control the quality of the interpolation by bringing a_n closer to $q_{n.}$ The good value is to set (a_n , q_n) segment to 1/3 of the original length. This point is also discussed in Parent's book.
- In Double(p, q) = 2(p*q)q – p, p*q is a dot product.
- Be careful about angles being given in degrees vs radians. AMC files give angles in degrees, but many C functions (e.g., those from the standard C library, such as sin, cos, acos, etc.) use radians.
- Q: Do we need to obey the joint angle limits?
  A: You do not need to worry about it in this assignment.

Extra credit

Here are some extra credit ideas:

- Make the OpenGL renderer prettier.
- Analyze the computation time of the different interpolation techniques.
- Support keyframes that are non-uniform in time.

Also, you might be awarded credit for any other creative or interesting solution to an issue in your implementation, as long as it is well-documented in your readme file. So, if you can think of any interesting feature, feel free to implement it.

Please note that the amount of extra credit awarded will not exceed 20% of this assignment's total value.

---

by Jernej Barbic, Yili Zhao and Yijing Li, USC; credits also to the CMU graphics lab