

CS 175: Assignment 5

Key Frame Animator (Part I)

Due:

Monday, October 26th, 11:59pm

Assignment Objectives

In this project you will complete the code necessary for a keyframe animation system with linear interpolation. In such a system, the user defines the state of the world at *fixed key times*. *Intermediate frames* are generated via interpolation (for now just linear) to yield an animation.

Task 1: Keyframes

You begin with code from assignment 4 that encodes the robots and camera poses as Rigid Body Transformations (RBTs), and implements the arcball interface. Your first task is to implement keyframe infrastructure.

You will have a linked list of key frames. Each key frame will store the state of the world, including all of the joint angles and the positions and orientations of the robots and sky cam. At any given time, you will be in a *current frame*.

At any given time you also have a *current state*, which stores the joint angles and positions and orientations of the robots and sky cam that is being displayed on the screen. When the user manipulates the scene, the current state is updated.

You should implement the following hot Keys:

- space : Copy current key frame to current state
- u : update: Copy current state to current key frame
- > : advance to next key frame (if possible). Then copy current key frame to current state.
- < : retreat to previous key frame (if possible). Then copy current key frame to current state.
- d : delete current keyframe (unless it is keyframe 0). retreat to previous key frame, copy to current state.
- n : create new keyframe immediately after current keyframe. copy current state to new keyframe. advance to new key frame.
- i : input keyframes from input file. You are free to choose your own file format.
- w : output keyframes to output file. Make sure file format is consistent with input format.

Task 2: Linear interpolation

Your next task is to implement simple linear interpolation to create an animation from the key frames. You will have to interpolate each joint angle, as well as the positions and orientations (represented as quaternions) of the robots and sky cam. In between any 2 keyframes, you will interpolate the above quantities using simple linear interpolation.

During the animation, you will create intermediate states by evaluating the linear interpolator at intermediate times, and then use these intermediate states to display a frame. For the orientations you will need to use quaternion linear interpolation, as explained in class. This will require you to implement the quaternion “power” function.

The quaternion `power` function is applied to a quaternion and returns a quaternion. The C/C++ function “`atan2`” (in `<math.h>`, or `<cmath>`) can be useful, depending on your implementation. Remember that the `power` function is computed in order to interpolate rotations, and we would like those to be interpolated “the short way”. This means that you will have to make sure to negate any input quaternion to the `power` function that has a negative w-coordinate before computing its `power` and returning the result.

The animation will be viewed from whatever viewpoint is current. This can be changed using the ‘v’ key from `asst2`.

Hot Keys:

- `y` : Play the animation
- `+` : Make the animation go faster, this is accomplished by having one fewer interpolated frame between each pair of keyframes.
- `-` : Make the animation go slower, this is accomplished by having one more interpolated frame between each pair of keyframes..

You can think of the sequence of keyframes as being numbered from -1 to $n + 1$. You will only show the animation between frames 0 and n . This will be useful when doing Catmull-Rom interpolation in the next assignment. Because the animation only runs between keyframes 0 and n , you will need at least 4 keyframes in order to display an animation. If the user tries to play the animation and there are less than 4 keyframes you can ignore the command and print a warning to the console. After playing the animation, you should make the current state be keyframe n , and display this frame.

Interface Specifications

When any hot key is pressed, the console should print out an appropriate message.

Implementation Notes: Animation Playback

For playing back the animation, you can use the GLUT timer function `glutTimerFunc(int ms, timerCallback, int value)`. This asks GLUT to invoke `timerCallback` with argument `value` after `ms` milliseconds have passed. Inside `timerCallback` you can call `glutTimerFunc` again to schedule another invocation of the `timerCallback`.

A possible way of implementing the animation playback is listed below. Calling `animateTimerCallback(0)` triggers the playing of the animation sequence.

```
static int millisec_between_keyframes = 2000; // Controls speed of playback.
static int millisec_per_frame = 1000/60; // Draw about 60 frames in each second

// Given t in the range [0, n], perform interpolation and draw the scene
// for the particular t. Returns true if we are at the end of the animation
// sequence, or false otherwise.
bool interpolateAndDisplay(float t) {...}

// Interpret "value" as milliseconds into the animation
static void animateTimerCallback(int value) {
    float t = (float)value/(float)millisec_between_keyframes;

    bool endReached = interpolateAndDisplay(t);
    if (!endReached)
```

```
        glutTimerFunc(millisec_per_frame, animateTimerCallback, value + millisec_per_frame);
    else { ... }
}
```