CS 4620 Practicum Programming Assignment 6
Animation

out: Friday 14th November 2014
due: : Monday 24th November 2014

1 Introduction

In this assignment, we will explore a common topic in animation: key frame animation. You will solve some written problems to get a better understanding of the underlying theory of animations and then write code to animate scenes using key frames.

Key frame animation is a technique wherein we define the layout of a scene at certain times (key frames), and we create frames between them by cleverly interpolating between our defined key frames. This rapid rendering of scenes gives us the illusion of animation. The award winning animation short film *Hunger (1974)* was one of the first applications of key frame animation.
`http://bit.ly/1EDvU6R`

2 Written Problem 1 - Interpolating in 2D

On another page are pictures of a square object (with its edges marked so you can tell them apart) under three pairs of linear transformations. For each pair:

1. Write the 2x2 transformation matrices for each pose; call them M_0 and M_1.
2. Write the matrix $M_{0.5}^{\text{lin}}$ by linearly interpolating elementwise halfway between M_0 and M_1.
3. Write each of these two poses in the form $M_i = R_i S_i$, where R_i is a rotation and S_i is a symmetric matrix with positive entries on the diagonal.
4. Use this decomposition to interpolate a matrix $M_{0.5}^{\text{pol}}$ that is halfway between M_0 and M_1.
5. Draw the two intermediate states of the object (you can print out the PDF and draw on it if you want).

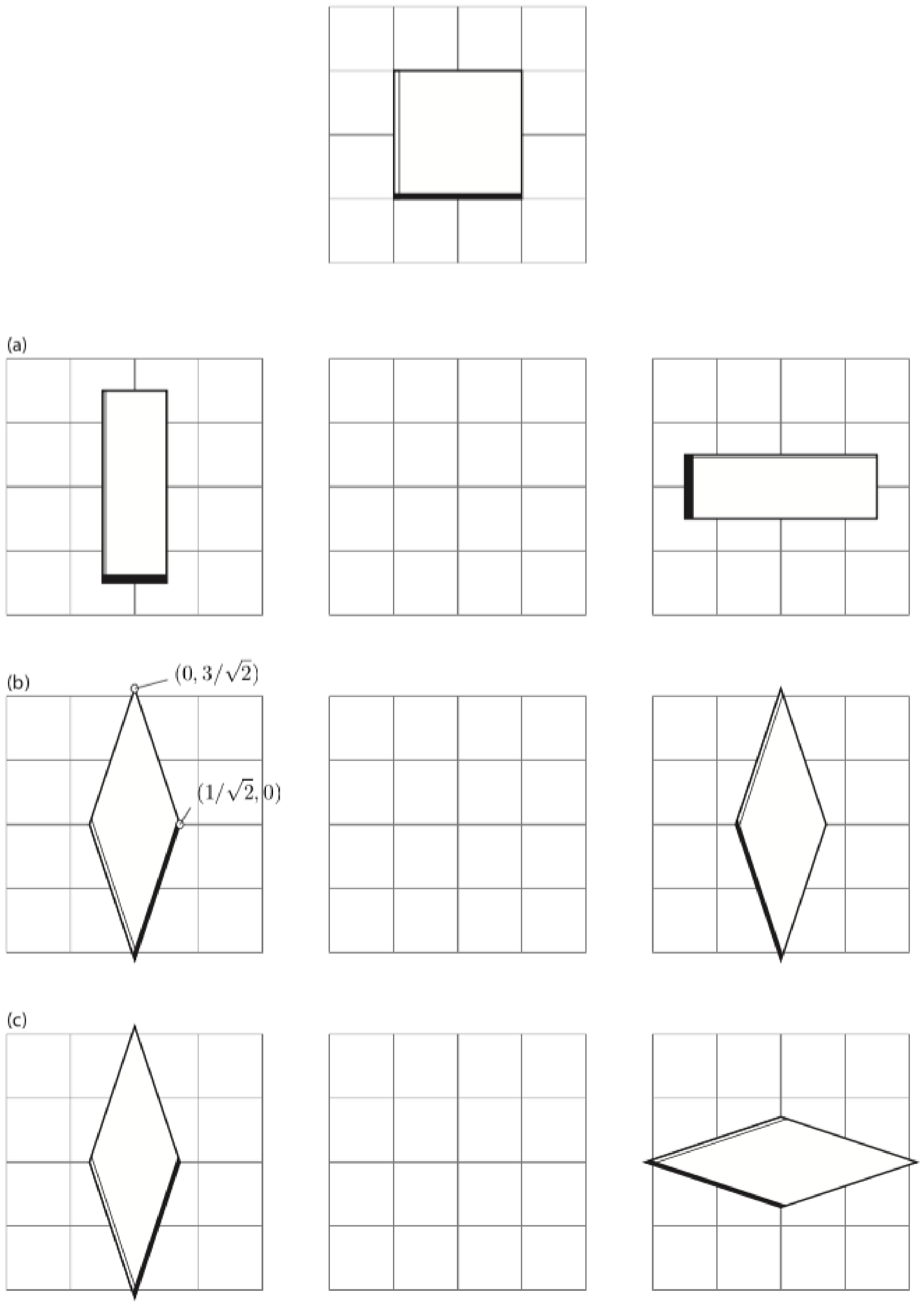

Figure 1: Written Problem 1

3 Written Problem 2 - Interpolating 3D Rotations

Using the `egl.math.Quat` class to convert between quaternions and rotation matrices.

1. Write down the quaternion q_1 that corresponds to the identity rotation and the quaternion q_2 that corresponds to a rotation of 180 degrees around the x axis.
2. Using the formula for spherical linear interpolation, generate the quaternion q_3 that is one-fourth of the way from q_1 to q_2. What are the axis and angle of the rotation q_3 represents?
3. Convert q_3 to a 3x3 rotation matrix $M(q_3)$. Verify that starting with the identity and applying this matrix 4 times results in the rotation corresponding to q_2.
4. Write down the matrix corresponding to a rotation of 90 degrees around x followed by a rotation of 90 degrees around y, and convert it to a quaternion q_4. Do the same with y and then z, producing q_5.
5. Using the formula for spherical linear interpolation, generate the quaternion q_6 that is one-fourth of the way from q_4 to q_5. What are the axis and angle of the rotation q_6 represents?
6. Compute the quaternion q_7 for the rotation that rotates from q_4 to q_6—that is, $M(q_6) = M(q_7)M(q_4)$. (It might be easiest to do this by converting everything to matrices, determining the rotation, and then converting back.) Verify that starting with $M(q_4)$ and applying $M(q_7)$ four times, you end up at $M(q_5)$.
7. What are the axis and angle of the rotation q_7 represents? Spend a few minutes holding a book and rotating it around its various axes, until it seems clear that this is the right answer.

4 Problem 1: Keyframe Animation

4.1 Getting Started

For the programming assignment, we will implement the primary features of a keyframe animation framework. A new commit has been pushed to the class Github page in the master branch. We recommend switching to your master branch, pulling this commit, and creating a new branch (e.g. A6 solution) and committing your work there. This new commit contains all the framework changes and additions necessary for this project. For this project, we use `lwjgl`, so you may need to update some settings of your build to have this external library work properly. We have included the correct jar files, but you may need to configure your native library location. To do this in Eclipse, go to `Project -> Properties -> Java Build Path -> Select Libraries -> Select the lwjgl Drop Down Menu -> Modify Native Library Location`. Modify this setting so that it matches your OS.

We have added a framework in the package `cs4620.anim` that stores keyframes and provides an interface for editing them; you will build on that framework and implement a few main features that enable correct interpolation of animations between keyframes.

4.2 The Interface

At first glance, your window may not like the one in the picture. Hover over the bottom of the window to reveal the timeline, complete with highlighted keyframes. We have implemented a keyboard based animation interface:

- Moving between frames:

  `[ ]`

  : We use the [and] keys to move to the left and right keyframes.
- Adding a frame: To add a keyframe at the current frame, press the `n` key. The new keyframe will be initialized to the current state of the scene when you clicked the button.
- Editing a keyframe: To edit a keyframe, navigate to that keyframe, edit the transformations at that state, and press `n`.
- Removing a keyframe: Navigate to a keyframe and press the `m` key to remove a key frame.
- Playing and pausing the animation: The \ key is used to play or pause the animation.

4.3 Keyframes

For this problem, we have extended the scene graph system to store information about multiple keyframes. The `AnimTimeline` class stores a:

- TreeSet of `AnimKeyFrame` . Each key frame contains a frame index number and a 4 x 4 matrix transformation.
- A `SceneObject` that it operates on.

Every time the `update` loop is called in the Game loop, `updateTransformations` is called in `AnimationEngine` which sends transformation events to the corresponding `SceneObjects`.

We have already implemented the functionality of adding and deleting keyframes from the scene.

4.4 Functionality to Implement

You will implement:

- `void updateTransformations()`
  A method that iterates through all the animation timelines, and calls `sendEvent` on the `scene` with the correct interpolated transformation on the object. For example, if the object on the `AnimTimeline` is called `object`, then
  `scene.sendEvent(new SceneTransformationEvent(object));` would begin the propagation of the event.

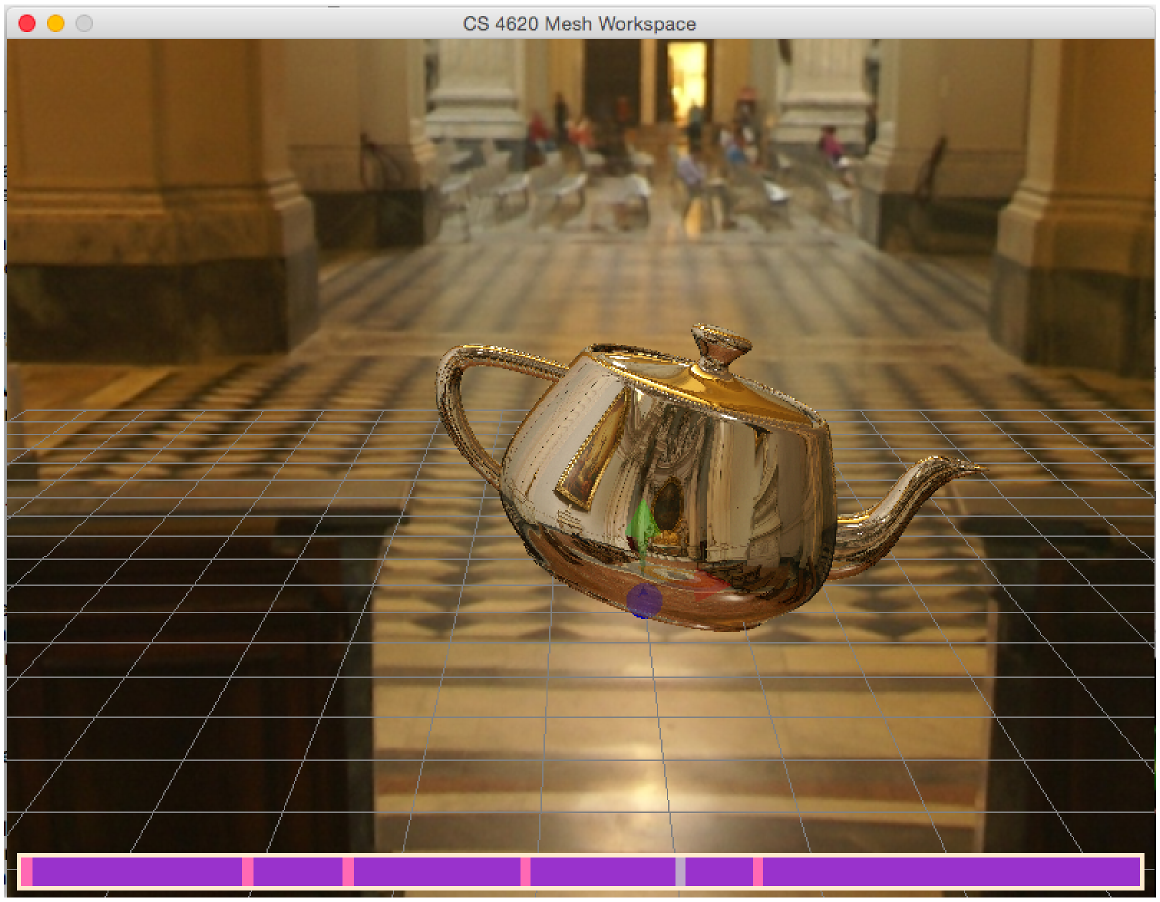

Figure 2: The keyframe animation interface (The timeline appears on hover).

4.5 Interpolation Overview

4.5.1 Naive Approach

When computing the interpolation of two transformations, the naive approach is to linearly interpolate each of the 12 free values of the 4 x 4 matrix. You may want to try this out and see why this doesn't work so well. See what happens when you rotate an object 180 degrees!

4.5.2 The more accurate approach

Linearly interpolating rotations by value does not work very well. While scales and translations can be linearly interpolated to give good results, the same doesn't apply to rotations. So what can we do? It turns out there is a good way to interpolate between two pure rotations: first express the rotations as quaternions, then use a method called `slerp`, or spherical linear interpolation, to interpolate between the quaternions.

Therefore, the general idea for a more accurate interpolation is:

- Decompose the surrounding 4x4 matrix transformations around the current key frame into rotations, translations, and scales.
- Linearly interpolate the translations and scales, and interpolate the rotations by converting to quaternions, slerping, and then converting the result back to a rotation matrix.
- Recompose the constituents to give a transformation for the current frame.

4.5.3 Decomposing transformations

A homogenous transformation M is a 4x4 matrix that can be decomposed firstly as follows

$$\begin{bmatrix} RS & T \\ 0 & 1 \end{bmatrix}$$

Here, T represents the translation, R the rotation and S the scale. So just how do we decompose the upper left 3x3 portion of the matrix into its constituents? Polar decomposition.

We have conveniently written the `egl.math.Matrix3.polar_decomp` function for you. There is more about quaternions, spherical linear interpolation, and the polar decomposition, in the Animation lecture slides and notes, and in Chapter 17 of the textbook.

4.6 Slerp

Spherical linear interpolation occurs between quaternions, so we need to convert our representation of R as a 3 x 3 matrix into a quaternion for each keyframe, and then convert the interpolated quaternion back to a matrix.

4.6.1 Converting between Rotation Matrices and Quaternions

We do not ask you to implement the inter-conversion between rotation matrices and Quaternions. You can use the class `egl.math.Quat` to convert from a rotation matrix (using the appropriate `Quat` constructor) and to a rotation matrix (using the `toRotationMatrix` method). Feel free to look at the code to understand how it works if you're interested. All you need to know about Quaternions is that they are 4-vectors, and that unit-length quaternions, called unit quaternions, can represent 3D rotations and also interpolate them well. If you want to learn more, take CS 5625 next semester!

4.6.2 Linear Interpolation

Typically, we visualize quaternions as vectors that represent rotation. Yes, they have 4 numbers, but as most things with higher dimensionality, imagine them to be in a lower dimensional space, say 2D, and say "four" to yourself.

Given two quaternions q_1 and q_2, and a t to interpolate such that $0 < t < 1$, a linear interpolation, or lerp, q is given by:

$$q = (1 - t)q_1 + tq_2$$

This will work, if we normalize the interpolated quaternion so that it's a unit quaternion again, but unfortunately the rotation does not happen at constant speed—it will go faster towards t=0.5 and slower towards the ends.

4.6.3 Spherical Linear Interpolation

Intuitively, spherical linear interpolation, or slerp, gets around this problem by interpolating these quaternions (read: vectors) along the sphere they are both a part of.

We'll avoid the details of the derivation of the slerp formula, which are in the lecture slides. Let's just review how slerp is calculated. Given quaternions q_1 and q_2 at the surrounding key frames and a interpolation ratio t such that $0 < t < 1$, we first calculate the angle between q_1 and q_2, called Ω, as follows:

$$\cos \Omega = q_1 \cdot q_2$$

Then the interpolated quaternion q is given by

$$q = \frac{\sin(1 - t)\Omega}{\sin \Omega} q_1 + \frac{\sin t\Omega}{\sin \Omega} q_2$$

5 Extra Credit

Currently, we only use the 2 immediate surrounding key frames of the current frame to generate it. What if we wanted to smooth things out, by interpolating over a larger window. For extra credit,

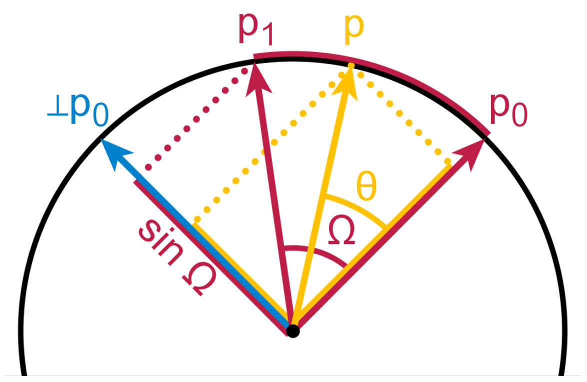

Figure 3: A graphical explanation of spherical linear interpolation

you can choose to implement the Catmull-Rom Spline Interpolation.

5.1 Catmull-Rom Spline Interpolation

The Catmull-Rom spline is a cubic interpolating spline that passes through all its control points. For the interval of the curve specified by control points $P_{i-2}, P_{i-1}, P_i, P_{i+1}$, the spline has the formula

$$P(t) = \frac{1}{2}\begin{bmatrix}1 & t & t^2 & t^3\end{bmatrix}\begin{bmatrix}0 & 2 & 0 & 0\\ -1 & 0 & 1 & 0\\ 2 & -5 & 4 & -1\\ -1 & 3 & -3 & 1\end{bmatrix}\begin{bmatrix}P_{i-2}\\ P_{i-1}\\ P_i\\ P_{i+1}\end{bmatrix}$$

Note that each patch interpolates its middle two control points. That is,

$$P(0) = P_{i-1}, \quad P(1) = P_i.$$

Additionally, the tangents at these points are determined by the remaining two control points as follows:

$$v_{i-1} = \frac{1}{2}(P_i - P_{i-2}) \quad v_i = \frac{1}{2}(P_{i+1} - P_{i-1}).$$

5.2 Finding Control Points

To interpolate the state at some frame f using Catmull-Rom spline interpolation, we need to find four frames to define our control points. Call these frames f_0, f_1, f_2, and f_3. The frames are chosen as follows:

- f_1 is the closest keyframe before frame f.
- f_0 is the closest keyframe before frame f_1.
- f_2 is the closest keyframe after frame f.
- f_3 is the closest keyframe after frame f_2.

That is, we find the four subsequent keyframes in our animation that satisfy $f_0 < f_1 < f < f_2 < f_3$. The time t for the interpolation is determined by how far along frame f lies between f_1 and f_2.

We must also consider a few corner cases. When f is later than the final keyframe of the animation, we can simply copy the state of the final keyframe. Otherwise, if fewer than four control points can be found, you can duplicate keyframes where necessary (e.g., set $f_0 = f_1$ or $f_3 = f_2$ if there is only one keyframe before or after f). Done correctly, your spline should successfully pass through every keyframe in the animation.

5.3 Interpolating Rotations

While using a spline to interpolate scales and translations is (relatively) straightforward, it's not obvious how to generalize slerp to a smooth interpolation using 4 keyframes. As an additional extra credit problem, read about and implement some form of quaternion spline that has the properties (1) evenly spaced keyframes result in constant-speed rotation, and (2) rotations are C^1 continuous across keyframes. [1]

6 Implementation Notes

As with previous assignments, places where you will add code are marked with `TODO`; note that some of these are marked as optional.

7 What to Submit

Submit a zip file containing:

1. Your solutions to the written problems, in pdf form.
2. Your solution organized the same way as the code on CMS. Please include your `.classpath` and `.project` in the submission as well as all the pre-existing files in the codebase, and not merely your modified files.
3. A readme file containing
   - You and your partner's names and NetIDs.
   - The answer for the written problem.
   - Whether or not you have done the extra credit problem.
   - Any problems with your solution.
   - Anything else you want us to know.

[1]Note that this is a fairly tough extra credit problem and you can still get extra credit for spline interpolation of translation and scale, without smoothly interpolating the rotations.