6.837 Final Project: Project Reina

Larry Wang

January 18, 2017

1 Introduction

Project Reina is a 3D music visualizer that parses MIDI files to animate a person playing the trumpet with accurate fingerings.

2 Purpose

Project Reina is inspired by the anime *Sound: Euphonium!*, which features beautifully hand drawn animations of various musicians playing their instruments (Reina is a trumpet player in the show). It is also inspired by music rhythm games such as Guitar Hero, which include animations of musicians that are either musically inaccurate or made painstakingly on a song by song basis to match the music. The goal of Project Reina is for the user to input any song and an accompanying MIDI file to procedurally generate in real time musically accurate animations of musicians. It is also intended to allow the user to "program" a show via keyframes on arbitrary parameters such as lighting and camera angle.

2.1 Scope

While the end goal is to feature several supported instruments, the scope of this project is limited to animating the fingers of a trumpet player to match the position of the three valves via analytic inverse kinematics. The system also currently provides an extensible API for key framing various time based parameters for the performance, which is used in the demo to animate the colors of note blocks.

3 Background

3.1 Inverse Kinematics

Inverse Kinematics is a heavily researched field in computer graphics. From the animator's perspective, it is often easier to specify the ending position and orientation of a particular joint than to determine what necessary rotations are needed to reach the desired position. There are two main approaches to solving an inverse kinematics problem: analytic and iterative. In an analytic solution, a closed form solution for the possible joint configurations is found. Analytic solutions are preferable if possible as they do not require expensive iteration cycles. For more complex systems however, such as a human skeleton, analytic solutions cannot be readily derived. Iterative approaches such as Cyclic-Coordinate Descent and Jacobian methods have been developed to solve arbitrary joint configurations.

For the problem domain of trumpet fingerings, I decided to use a 2D analytic solution and adapt it to 3D space. The animation needs to be in real time, so a fast solution is preferred. Moreover, I plan to include several more instruments later that need to be simultaneously animated via IK, so speed is a necessity. I consulted chapter 4 of Professor Harry Asada's *Introduction to Robotics* [Asadf] for analytic solutions to 2D problems.

3.2 Keyframing

Keyframing is staple tool of animation. The idea of keyframing of a parameter is to set "check points" at certain points in time and interpolate between those check points (rather than explicitly

specifying the value of the parameter at every time step). Typical keyframe systems also allow the user to choose what type of interpolation is used. Most animation and video packages offer their own keyframe system. I wanted to design an easily extensible C++ based keyframing infrastructure to easily modulate arbitrary parameters over time.

4 Approach

4.1 Unreal Engine

Project Reina is built with Unreal Engine. While Unreal offers its own IK system and keyframing system, for the purpose of this project I developed my own methods.

4.2 Music Subsystem

The music subsystem (written in C++) is involved in converting MIDI and JSON files to a convenient representation and creating an event based system for notifying listeners when notes begin and end (and what fingerings to use). It is completely decoupled from the graphics subsystem.

4.3 Inverse Kinematics

The overall approach is to fix the hand position and orientation to be reasonably aligned with the trumpet and then treat each of three relevant fingers (index, middle, and ring) as independent 2D IK problems through projection.

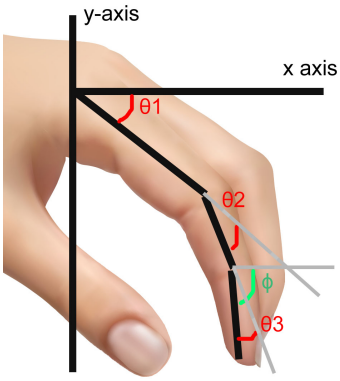

Figure 1: 2D IK problem applied to fingers

4.4 2D IK

In the 2D IK problem, we have three bones and their corresponding lengths and three joints to manipulate (see figure 1) We are given an end position for the the tip of the last bone (tip of the finger position) and the angle ϕ the last bone makes with the x axis. We must calculate the three joint angles necessary to reach the end point. This is just a geometry problem and the derivation can be found at [Asadf]. It is important to note that with this geometry two configurations may be found (or none at all), so the system must choose the best one. To resolve the choice, I specify min and max angles for each of the joints, as shown in Table 1, which I determined by examining the rotation limits of my fingers. If a valid solution exists it is chosen. If no solution exists, I set each angle to zero.

Angle	Max	Min
θ_1	80	-70
θ_2	0	-135
θ_3	0	-45

Table 1: Angle limits (in degrees) for finger

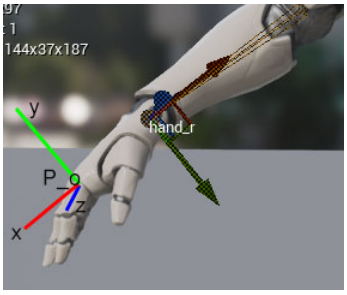

Figure 2: Axes for hand joint in 3D space. Note we reverse the direction of each axis and treat the finger joint location as the origin

4.5 3D to 2D

The most challenging part of this IK problem is moving from the 3D space the fingers live in to a 2D representation. Global versus local coordinate frames and axis directions must be carefully determined. The following is the algorithm I developed for moving from 3D to 2D.

1. Obtain the rotation matrix for the *hand* joint in *world* space. Call this **R**.
2. Determine the world space x, y, and z axis for the rotation matrix **R**. **We flip each axis for convenience** such that for the right hand, the x axis faces away from the wrist, the y axis points away from the palm, and the z axis points toward the thumb
3. Obtain the *world* space location of the first joint of the finger (closest to the hand). Call this P_o.
4. Obtain the *world* space location of the destination. Call this P_d
5. Project P_d onto the xz plane with P_o as the origin of the plane. Using the coordinates in the xz plane we can determine the rotation around the y axis needed to bring the finger inline with the point. Note, we used the origin of the finger joint and the axis of the hand joint.
6. Project P_d onto the xy plane with P_o as the origin of the plane. These are the xy coordinates that will be plugged into the 2D IK problem. Obtain the bone lengths in *world* space from the 3D models.
7. Set the rotation around the first finger joint's *local* Y axis to the angle found in step 5.
8. Using the resulting 2D IK angles, Set the rotation around each finger joint's *local* Z axis to the angle found in step 6.

Through this algorithm, we only need to animate and obtain the position of the trumpet valves and the fingers will follow along. We set ϕ, the direction the last finger bone makes with the x axis of the hand, to -115 degrees.

4.5.1 Keyframing

In a music visualization, it is nice to be able to animate parameters over time such as lighting and camera changes. The heart of the keyframing framework is a templated class called KeyFrame.

```
template <typename T>
class KeyFrame{
public:
    float t;
    T value;
    KeyFrame(float t, T value) : t(t), value(value) {
    }
    virtual ~KeyFrame() {
    }
```

```
    virtual T eval(KeyFrame<T> prevKeyFrame, float t){
        return value;
    };
};
```

A KeyFrame consists of both a time stamp and the parameter value associated with it. Crucially, it has an override-able eval function that takes time t and the previous keyframe to return the parameter value at time t. To design different interpolation modes for KeyFrames, such as a linear or quadratic, only the eval function needs to be overridden. Moreover, the parameter value type is templated, so any class that supports arithmetic operations can be used. An example subclass, KeyFrameEaseIn, has the following eval implementation for quadratic easing:

```
virtual T eval(KeyFrame<T> prevKeyFrame, float currentTIme) override{
        float totalTimeDiff = this->t - prevKeyFrame.t;
        float percentThere = (currentTIme - prevKeyFrame.t)/totalTimeDiff;
        percentThere *= percentThere; //quadratic
        return prevKeyFrame.value + (this->value - prevKeyFrame.value) * percentThere;
    };
```

Furthermore, there is a helper class called PerformanceEventEmitter that takes a vector of KeyFrames and automatically handles calculating the current parameter value at the performance time. In the demo, the color of the note blocks is modulated using this key frame system.

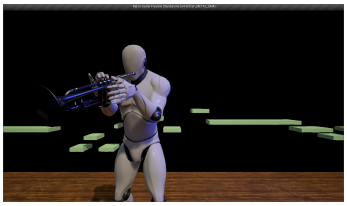

Figure 3: Screen shot of demo

5 Results

The final result for this project features an animated trumpet player playing the trumpet along with the music with accurate fingerings, driven by the corresponding midi file. The system runs in real time and the finger animations themselves are fairly realistic, although it could benefit from a true three bone 2D analytic solution. The current solution requires specifying the angle the last bone make with the x-axis, but the ideal solution would only require the end position itself. A demo video can be found at https://www.youtube.com/watch?v=b7Yyh4w5R70

6 Conclusion

Both the IK and keyframing could have been done using Unreal Engine built in functions, but it was a fun challenge to implement them myself. In particular I've found the transformation and hierarchy sections of the course to be extremely helpful for the project and giving me the confidence to think in 3D space. Next steps include supporting more instruments, such as a drumming animation which guides the drum sticks to the correct drum/cymbal, using iterative IK methods.

References

[Asadf] Harry Asada. Introduction to robotics. https://ocw.mit.edu/courses/mechanical-engineering/2-12-introduction-to-robotics-fall-2005/lecture-notes/chapter4.pdf.