Real-Time 3D Animation of a Piano Player and Piano Synchronized to a MIDI Audio File

Abstract

We have created a real-time 3D model of a musician playing a piano synchronized to a user specified MIDI audio file. The musician's hands and the piano keys are accurately animated based on the music that is playing. The entire animation can be displayed from any point of view using an augmented OpenInventor ivview interface.

Team #5

Nilesh Agarwalla, Igor Kaplansky, Ying Lee [nilesh, ibkaplan, leeying]@mit.edu

Introduction

Our team had certain requirements when thinking up ideas for our final project:

- music-related - all 3 members of the team have some musical background
- multimedia - video alone is boring. We wanted to incorporate audio and video to provide a fuller experience.
- practical - we wanted our project to have some practical use
- real-time animation - we felt we would learn a lot more implementing a real-time system rather than rendering a pre-scripted movie.
- fun and challenging - most importantly, we wanted the project to be fun to implement and the

results rewarding.

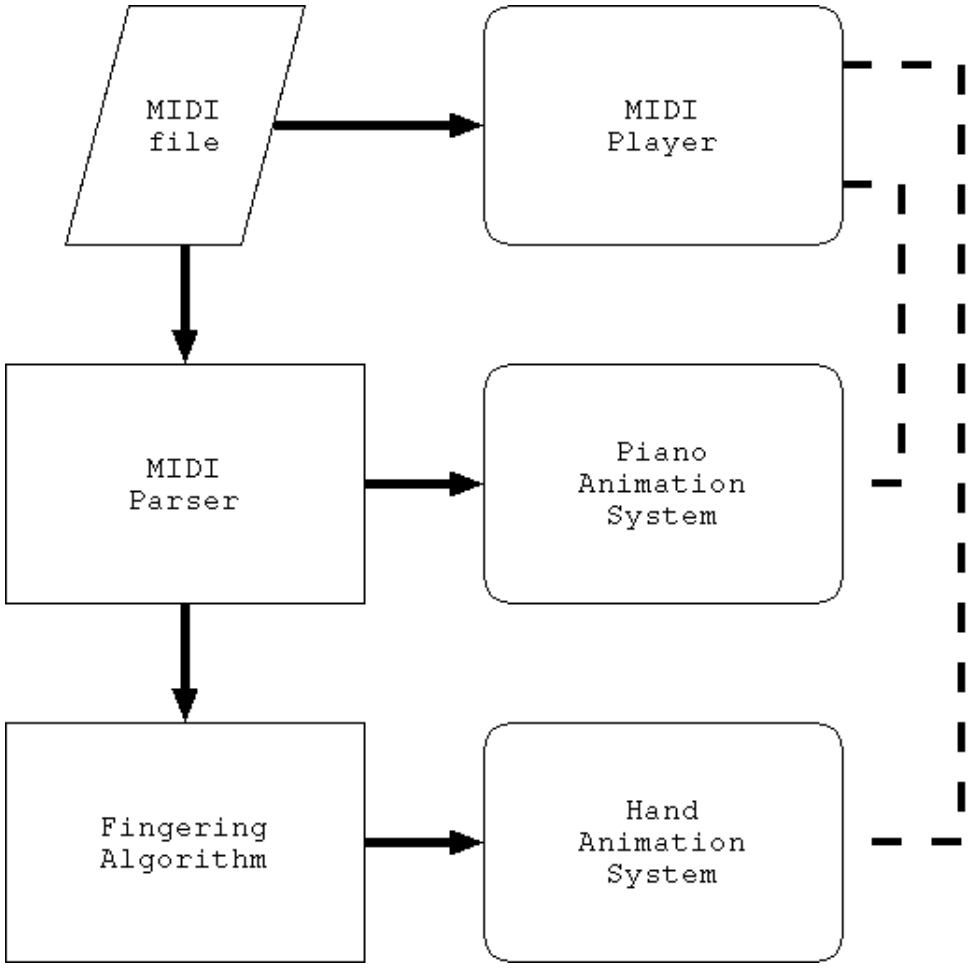
We decided that a real-time animation of a piano player encompassed all of the ideals listed above. Determining an efficient algorithm for deciding hand positions and fingerings is a challenging music theory problem. Synchronizing the animation to the music and balancing resources between graphics and audio is a challenging multimedia implementation problem. The project is practical because it is possible to extend the system to be a piano teacher that plays the music aloud, displays the appropriate notes to press, and shows the correct key fingerings.

Developing a real-time animation incorporates some concepts introduced in class. Since multimedia animations can take a lot of resources, we take advantage of the mind interpolating fast moving images to reduce the number of animation frames and thus reduce the processing resources required for animation. We are extending upon the OpenInventor skills we learned in class by taking advantage of its animation capabilities. We also use the concept of modeling objects as parametric entities to expedite the implementation of the animation.

Goals

- Understand the MIDI format and extract the appropriate data
- Learn how to animate using OpenInventor
- Accurately model hand, finger, and instrument movements
- Synchronize the animation with the music

Design Overview



Before the animation can begin, the MIDI file must pass through 2 steps of processing.

1. The MIDI Parser reads the MIDI file and generates a linked list of chords sorted by increasing time. A chord in our system is considered any change in any note. So, a chord can be a single note or an event that turns off a note. The result is used by the piano animation directly.

2. The Fingering Algorithm reads in the chord list generated by the MIDI Parser and assigns fingers to each note of each chord. A scoring heuristic is used to decide between multiple key fingering possibilities. The result is used by the Hand Animation.

Once all the chord and fingering data is generated, the system is ready for animation. The starting and stopping of the animation is controlled by the MIDI Player. The Hand and Piano Animation Systems query the MIDI Player for the elapsed time of the song and then finds the currently chord in the chord list.

The animations use the data as follows:

1. The Piano Animation System simply reads the notes in the current chord and animated the corresponding keys on the piano.

2. The Hand Animation System reads the fingering data for the current chord. The animation is more complex than the piano because it must translate the hands properly to be aligned with the piano and must set of each angle in the finger to match the corresponding key on the piano.

Achievements

Our team feels we accomplished almost all of the goals as stated in the initial proposal. We have successfully written a MIDI parser and player that is suitable for our project, modelled a parameterized piano and hand in OpenInventor, synchronized the animation to music in real-time, and designed a valid key fingering algorithm.

Due to time contraints, we ended up implementing only the piano player and piano instead of an entire band. We feel that the framework we've developed and the knowledge we've gained can be easily extended to add other instruments, however.

There were several obstacles that we had to work around to complete the project:

- Synchronizing animation to music - We decided on using the MIDI format as our source of music data for several reasons:

  1. Notes are discretized and explicitly defined in a MIDI file
  2. Time data is stored on a per note basis which makes synchronization easier
  3. Playing MIDI files takes little system resources

  Since the animation needed to be synchronized to the music, we felt it necessary to implement a simple MIDI player that we could tie closely with the animation system rather than use a third-party program. We extended some sample SGI MIDI player code and linked it to the OpenInventor interface. The MIDI player runs as a separate thread to prevent audio skipping. For synchronization purposes, the time elapsed in the MIDI player can be queried by the animation engine to nanosecond accuracy.

  Although extending the sample MIDI player code saved us a lot of time, the robustness of the MIDI player suffered. The MIDI player only plays a subset of all possible MIDI formats. In particular, it cannot handle MIDI files that contain more than one tempo change in them. Currently we detect most files which are invalid, but ideally, we would liked to have a system that could play from any MIDI file. It would have been technically possible to write a more robust MIDI player from scratch, but we felt the benefits wouldn't warrant the time required. Fortunately, many MIDI files exist that DO work properly with our current MIDI player.

  The core of the animation engine lies in the MIDI parser which was written using standard MIDI libraries. The file is processed to generate a list of chords sorted by time which is then used by the animation systems to determine which notes needs to be pressed at any given instant.

- Animate efficiently - We tried two schemes for animation. Our first approach put much importance in the quality of the animation. We interpolated as many frames as we could between the start and end frames for each key/finger that needed to be pressed. Although the animation looked great, this turned out to be much too processor intensive, and caused the animation to lag behind the music. We noticed that the animation of the keys and fingers were relatively quick

events, so for our second approach, we decided to improve speed by skipping interpolation all together.

- Augment the OpenInventor interface - Although we extended the OpenInventor interface to add buttons to Play and Stop a MIDI file, we wanted to add much more functionality with extended MIDI player controls such as rewind, fast forward, MIDI file loading, and dynamic tempo changing. However, We decided that adding rewind and fast forward buttons would have increased the complexity of the synchronization system. Also, We didn't have time to figure out how to add controls more advanced than primitive buttons although it would have definitely been possible.

- Make parameterized 3D objects that look realistic:

    - The piano body is modelled using an IndexedFacedSet and then wrapped in a wood-grain texture to make it look more realistic. The keys, composed of simple stretched cubes, are then overlayed onto the piano body. Each key has a rotation associated with it that is set and unset by the Piano Animation System to simulate the pressing of keys.

    - The human hands were modelled using simple Inventor primitives like spheres and cylinders and then wrapped in a skin texture. Each finger is parameterized with 4 angles: 3 angles specify different bending positions for the finger and the last angle is used to specify the "stretch" or lateral displacement of the finger with respect to its default position. These angles are then manipulated by the Hand Animation System using the key fingering data

    - The animations are not as detailed as we had wanted them to be. We wanted to model an entire human, but instead ended up modelling only the hands and forearms. We also wanted the piano to have strings that vibrated along with the key presses, but for efficiency reasons, we ended up removing the strings all together.

- Determining optimal key fingering - The current fingering algorithm s not optimal, but it is good enough to demonstrate the capabilities of our system. The algorithm had two important steps:

    1. Generate all the possible fingerings for the chord - Certain properties were exploited to speed up fingering generation. These include:

        - Keeping fingerings the same for notes that are sustained from the previous chord.

        - Exploiting physical limitations of the hand. For example, if a chord of 2 notes are separated by 5 white keys. Then, the fingerings cannot be consecutive, it would have to skip at least 2 fingers because of the span.

    2. Evaluate the fingerings using the following scoring system.

        - Give a lot of points for using the same finger to play a sustained note because we want to keep the same finger if possible.

        - Find the lateral translation from the previous chord based on the overlapping notes between the previous and the current range

- Calculate the translation needed to play the next 8 notes using a rough estimation for the middle positions of the rest chords. For example, the middle position for the left hand is note at 1/4 * number of notes, and the middle position for the right hand is 3/4 * number of notes. A smaller translation gets the higher score.

- Add more points for fingers used in the previous chord that is consecutive to the current finger. For instance, a scale should be played with consecutive fingers.

- Subtract points for using the same finger that played a different note previously because it would be better to play with different fingers if possible.

- Add a few points for using the thumb because consecutive translations are usually done by the thumb. For example, playing a scale, the fingering is: 123 * (translation using thumb) 1234

Individual Contributions

In the initial stages, we all worked together on figuring out how to use the animation capabilities in OpenInventor. Once we decided that the project was feasible, we agreed on how to modularize the project and then divided the work among the members of the group.

- Nilesh Agarwalla - Implemented a basic MIDI parser/player, modelled a parameterized 3D piano and linked them to create a piano animation synchronized to the music.

- Igor Kaplansky - Modelled parameterized 3D hands and linked them to the piano animation to create an animation of a person’s hands playing the piano.

- Ying Lee - Designed and implemented the algorithm for determining the proper hand positions and key fingerings.

Lessons Learned

- We learned how to use the various animation libraries and callback systems provided by OpenInventor.

- We learned how MIDI systems work and how to program for them.

- We learned that it is necessary to make tradeoffs between functionality, performance, and complexity of detail to make a practical real-time animation system.

- We learned that it is better to set vertical goals for projects rather than horizontal ones. It is easier to make a primitive system that works and then embellish it rather than focus on making each layer perfect at once.

- We didn’t use a version control system, but at times we felt that would have been helpful to allow multiple people to work on the same file at the same time as well as preventing discrepencies of different versions of the same file.

- Implementing the MIDI parser and player turned out to be much easier than we had thought, while the algorithm for determining the key fingerings ended up being much harder.

Acknowledgements

- The sample MIDI parsing and playing code provided on SGIs proved to be valuable (/usr/share/src/dmedia/libmidi*).
- The 6.837 SoSceneViewer was used as a base for our animation system.
- 6.837 student Andreas Sundquist's colors used in his parametric model of bread were used as a base for colors of our hand model.
- http://www.nepthys.com/textures/ was a source for some of our textures
- We used the program XV by John Bradley to do image converstion for our textures.

Bibliography

- *The Inventor Mentor*:Programming Object-Oriented 3D Graphics with OpenInventor, Release 2 by Josie Wernecke, 1994.

Appendix

Operating Instructions

The animation runs only on SGI O2s. Before starting the system, you must first run "midisynth &", which is the software MIDI synthesizer on SGIs. If you run the system without running "midisynth", an error message will be generated.

piano -midifile < filename > [-channel <0-15>] [-temposcale < scaleFactor >] [-displaychord]

- **-midifile < filename >** where < filename > is the MIDI file you want the system to animate
- **-channel <0-15>** you can specify which channel of the MIDI file you want the system to animate. You can run "synthpanel" while the MIDI file is playing to see a description of what instrument each channel contains. [Default: 0]
- **-temposcale < scaleFactor >** scaleFactor is a floating point number that you can set to increase/decrease the tempo of the song. Values < 16 are slower and value > 16 are faster. [Default: 16]
- **-displaychord** prints the current notes/chords that are being played in real-time to screen.

Nilesh Agarwalla, nilesh@mit.edu

Last modified: Fri Dec 3 16:15:49 1999