Nicholas Waggoner

Chris McGilliard

Physics 498 – Physics of Music

May 2, 2005

Music Morph

Have you ever listened to the main theme of a movie? The main theme always has a number of parts. Often it contains your love theme, your heroic theme, and even a bad guy theme. Most of these parts of the song have essentially the same melody, but the way that melody is played determines the way each theme feels. Coming from a computer science background, we began to ask the question: Since the songs are so similar, is there a way to generate the other themes of a piece, given one of them? After thinking about this question for only a few minutes, we realized that we didn't have any of the groundwork laid to attempt it. So instead of trying to solve this question we decided to work on a different, but similar, project. We would look at trying to morph or transition between two songs.

The idea of transitioning or even morphing from one song to another has been around for a little while. Similar concepts have been around for quite some time in other areas, most notably pictures. You can remember many Photoshop-like programs where it could morph someone's face into another person's face. They all involved providing a starting point as well as some parameters or help from the user before it would run. This is just taking that idea to a new medium. This medium is now the universal element of music.

To be able to transition between pieces, morph from one section of a song to another section of another song seamlessly, or create solos automatically would be very valuable. This is what this program aims to provide.

There are actually numerous ideas stemming from the basic thought of automating song transitions. The original idea involved starting with a song or melody and creating a secondary song or melody, based closely on the first. It would be different enough to be a separate song; yet close enough that you could tell it was the same feel or style. This is a very difficult problem due to the lack of a concrete goal to work towards. A very similar idea was to start with both an original song as well as a goal song. Starting with the original song, we could slowly morph towards the goal song over time. This gave us a concrete goal to work towards, as well as a solid basis to start from. It is much easier to successfully reach a goal if you can actually see it.

Another option involved transition. Instead of morphing one song into another, one could start at one song and create a likable and logical transition to the next song. This would not necessarily involve changing the original song or the goal song, rather it would use those as bases to jump from and the middle ground in between (the transition) would be all new, but work it's way from the original to the goal.

The first step in implementing this project was to pick a language and a platform. We chose Java for a couple of reasons. The largest reason was the large number of packages that are included in the language. These packages helped make writing the parser much simpler. Another contributing factor for choosing Java is its portability. The University tends to push Linux systems, mainly because they are open source. However, the

majority of computer users still use Windows. Keeping both groups in mind, Java runs on a virtual machine that can be run on most systems. This allows our code to be run without modifications on nearly any machine.

After we chose our platform, we needed to find a suitable music format to use. We started by looking at the popular formats for storing music. We decided to stray away from those because of their complexity. After searching for a week or two, we ended up finding the ABC file format. This format allows for 4 octaves of notes, chords, triplets, bar notation, repeats, key signatures and more. However, we have learned through our CS background that baby steps are often the best path to follow to complete a project. We decided to look at only a subset of the notation. We initially removed everything except for the notes from the body of the music. We also retained most of the information contained in the header, which holds things such as the key signature.

We determined that computational costs for parsing were insignificant in comparison to the other parts of our code, so we decided to implement the parser by using a multi-pass scheme. The first pass simply read the file in character by character into an array in memory. This allowed us to have random access to the file for future use. The next pass had the job of removing the information that we were not going to examine. It semi-intelligently handled most symbols. However some symbols, like the ones signifying chords, could not be handled intelligently and the application would throw an exception saying the parser failed. The final phase of the parser looped through the generated note ids and filled out the statistics class for all of the notes used in the song.

The morph class did all of the fun work. During the earlier versions of the code, it

would generate rules to be used later to morph the two songs. More will be said about rule generation after the stats class is discussed. In later versions, the class contained hard coded rules that held parameters that could be used to adjust the speed of morphing. The main function in the class was, surprisingly enough, called morph. Its only direct parameter is the melodies class, which contained all the arrays after being parsed and a hash table of stats for each note. The morph function would repeatedly make calls to attemptChange. This function would apply the rules and return a success or failure code. It would continue to call the function until enough notes were changed that it was considered an iteration. After each iteration, the newly modified song would be written out to file so it could be examined.

The stats class is a container class that slowly got migrated out of our code. The purpose of this class is to contain all the statistics for a given note in a song. These statistics included frequency of the note, previous note frequencies, next note frequencies, location in measure, and other statistics that were for later versions of the code. In early versions of the code, the morph class would use the statistics to generate rules that would morph between the two characterized songs. Using these automatically generated rules, the morph class would then try and apply them repeatedly until the goal song was reached. Unless the songs were nearly identical, the rules that were generated were too specific and would require a large number of iterations to converge, if it would converge at all. Many times, the rules would form a cycle where notes would get changed in a sequence and eventually end up the same after several tries applying the rules. Other times, the rules would be so specific that the rule would never get applied. If there were enough of these rules, the attemptChange function may get called repeatedly and never

make a change. In the end, we decided we needed to move away from the auto-generated rules and move to a more structured environment that we could control. At this point we began to try and write our own rules.

For this application we attempted to decide on rules for the music to follow. We needed these rules so that when the application was running it would be making music that still sounded good to us. These rules would help the program shy away from “bad” notes as it was morphing from one song to another. Determining what bad notes are is a much more difficult process than we imagined.

There were many rules that were much harder to get to work. Many of these ended up being too restrictive and not allowing the music to move forward. The code would then take way too many iterations to converge. Repeating a pattern in either the original or goal melody would have provided a solid, good sounding piece to repeat, but that involved finding the pattern that sounded good, saving it, and being able to get back to it easily. Keeping a solid time signature would have made sure that each measure would remain with the same feel instead of having notes carrying over measures all the time. However, this involved keeping track of where in the measure the morphing process was running as well as limiting the possible note lengths at any given time to what was left in the measure. Limiting note lengths was also a possibility. This would have always given us three triplets in a row, for example, avoiding any kind of really weird syncopation. But once one note was picked as a triplet, then the next two were forced into being triplets to fill out the trio, for example. The same was true for other note values and their similar limitations.

We also tried to limit the next note in a sequence. For example, normally a C# would not follow a C very often. There are also other similar sequences that have been deemed “not good sounding.” These also limited the possible notes too much and would not allow the code to converge quickly. We also wanted to force the melody to stay based around a certain note (middle C, for example). Then whenever a random note was chosen that was beyond the point deemed too far, it would be thrown out and a new note would be chosen that would be closer to the base. There was also the possibility of using chords and banning certain notes in a certain chords. A simple example would be a C-C# chord, which would sound like a nice screeching sound. We definitely did not want that. Chords however are too complex to deal with at the point we are with this project. There were many rules that we thought of that would have been nice to be able to use, but concerning convergence they were too restrictive for them to be realistic at this point.

To start off, we decided that we should limit the range that a note can change in between iterations. A note should not be able to jump three octaves between iterations because then the song sounds like it is jumping all over the place and is random and chaotic. Limiting the range relative to neighbor notes was also important. This also limited the amount of jumping around by limiting the distance between two consecutive notes. The next step was to limit actual notes that were allowed. For example, if a song is in C major, then very few notes should be outside of the C major scale. Allowing a few accidentals here and there had to be allowed because they are allowed in normal music.

There are a lot of other things that could be tried concerning this same idea. Many of them we simply did not have the time for due to it only being a semester project and the difficulties we ran into concerning the rules. Many of these are directly related to some of

the rules that didn't work. We ignored the complexity of note lengths, accents and the position of notes in a measure, in favor of staying with the simple morphing idea and getting it to work well. There are also areas left involving dynamics and accents, speed changes and time signature changes, ties and slurs, and especially chords. Chords could be very valuable in being able to create a simple chord progression for musicians to play along with. The computer could display what chords it was picking as it was picking and the musicians could follow its lead, without having to come up with all the chord progressions themselves. On the same note, adding in other instruments or background elements to the automated program could allow it to create entire scores or songs by itself. Even something as simple as adding in a drum beat and a bass line to go behind the melody that is being played would add to the complexity of the output a great deal. There could be other options involving what kind of instruments the melody is designed for which would also be useful for writing an entire score.

There is also a possibility of morphing only a piece of a song to another piece of a song instead of the whole song. This would create a middle ground between morphing and transitioning and could morph to the next song faster. Instead of morphing the whole melody, choose a few bars and morph those to a few bars of the goal melody. The problem arises with finding the logical spot to begin the morph of the few bars. You could find the few bars that were the most similar between the two pieces and morph those to each other. If the song involves repeats, then it would not be too hard to use the last few bars of the repeat.

Another option would be to make it fast enough to run in real time so that the application could be set to run and could be told to stop when the user is sick of the music

it is creating. The user could also change parameters on the fly so as to change the music on the fly as well. This would turn it in to a tool that DJ's could use in new way. There are numerous advancements in this field that only require people to have the time to explore them and think up new ways to implement them.

You can hear these ideas being used in movie soundtracks, transitions between songs in a continuous set, improvisational jazz or blues, and many other places. In movie soundtracks one theme must transition or slowly morph into another theme as the movie moves that way. In a set of continuous songs a song in one key has to be able to transition to the next song in, possibly, a different key. Many people have a lot of difficulty with improvising or coming up with new ideas to use for it. Each of these ideas has been done, but has not been automated before. To have an automated program that could suggest good ways to do these transitions could be extremely valuable.

Bands probably agonize over how they are going to get from song A to song B smoothly without having to completely stop and start up again. They have to worry about what transition sounds and feels good, picking the right chords and the right chord progressions. Most are not worried that it sounds absolutely revolutionary, rather that the transition is smooth and sounds good enough to get to the next song without needing a complete break.

Movie score composers must deal with the same issues in moving from the hero's theme to the love theme to the bad guy's theme all in seamless transition. Most composers agonize over writing the main themes first, then attempt to fill in the rest of the time in the movie by adjusting the existing themes. These adjustments can involve

changing the feel of the existing theme (a heroic theme with dark overtones as our hero contemplates joining the dark side, for example). They can also involve transitioning between themes (After just conquering the dark side, our hero returns home to his true love, for example). These transitions have to be smooth and seamless without having to just play a couple random chords. They need to contain pieces of the old theme while it transitions to the new one. They also have to have adjustable times for how long it takes to transition. Something in the movie could be a drastic change or it could take quite awhile for it to switch over. The composer has to adjust to all of these possibilities. An automated program could give them many ideas for these transitions very quickly.

Jazz and blues music involves a great deal of solo improvisational work. For those that are not very comfortable with making things up on the spot, this program could help them out to an extent by providing ideas to use. If improved to work in real time the program could actual come up with solos in the right key and play along. By inputting the length of the desired solo, the key, and time signature, it could write it's own music. One could also set up a groove to repeat and then allow the program to solo over the groove for any amount of time, providing automated background music that would constantly be changing and yet remaining in the same designated style. However, we are a still a long way away from having computers play with a live band and sound good without considerable prompting from a DJ.

In the end, even the simple questions that we asked were fairly complicated. We worked through a number of solutions, and as to be expected, many of the solutions failed. However, each failure provided us with new insight. Each subsequent attempt built off the knowledge learned from past failures. After working through all of these attempts this semester, we realize that the project is poised to be able to head in many directions. While the morphing and transitioning between two songs may not be as smooth as we originally hoped, we are very pleased to have laid down the framework to address this and even more complicated problems.