A Simple Musical Model in Nyquist

Zack Sussman

zjsussma@andrew.cmu.com

Table of Contents

SECTION 1

Overview

1.1 The Philosophy Behind the Model

As a musician, when I compose and listen to music, I want to value certain aspects of the music above others. In general, abstraction levels allow us to force value with respect to certain aspects of music. When composing within a language such as Nyquist, I want the power for my code to be in alignment with what I feel the music to be. The musical idea I have is within me, and the sound is simply a realization of it. The code should formally communicate this musical idea. It should be in alignment with the experience of a musician who is digesting such a musical idea.

1.2 The Basic Model

In the basic model I have produced in Nyquist, we have the ability to define musical timings (rhythm), along with a pitch specification for a multi-voice composition. We are able to give names to moments in time that have meaning to us, and tell the computer to place sound during those names in musical ways. Specifically, we can add some number of voices to all play notes during those moments of time together. This creates what we call a section. Then, we can put sections together to create a musical work. The computer auto-mixes the sounds together so the musician doesn't have to think about mixing.

1.3 Drawbacks

I have produced a very simplified model in Nyquist, it is more of a proof of concept. The current version does not give control over the sound itself, only the pitches and moments in time that the pitches occur. A sinusoidal oscillator is used with a fixed envelope. Additionally, no thought is given to the countless other aspects of music besides melody and rhythm. However, I believe the notion of the time specification in my model along with how I used it to create a musical pitch specification is fundamental enough to carry great weight in further developing this model to include other musical aspects.

SECTION 2

Time Specification

2.1 Rhythmic Perception

This section explores how I decided to represent how we perceive rhythm. My claim is that at the most fundamental level, we feel rhythm as moments in time that we perceive in a certain light. One moment might feel heavier, another might feel lighter. Each moment has its own feeling. The way I represent this feeling is by giving each moment what I call a **label**. My claim is that what it means to perceive a label musically only really comes down to two things. The first is regarding duration. How much space does it take up? Imagine that someone told you to perceive 5 seconds in your own mind. Would you try to perceive 1 second 5 times? Would you try to perceive 3 seconds once and then 2 seconds once? What does it even mean to perceive 1 second?

I claim that musically, whenever we perceive a single duration, what it means is that we feel a heaviness at the beginning of that duration, the heaviness drops to weightlessness around the middle of it, and then it scales back up to full weight towards the end. It is this oscillatory type of motion that allows us to perceive a dynamic **swing** through time. Music is about combining together these swings in interesting ways.

For the purposes of my simple model, I will have a single root duration (what most musicians would describe as **the pulse**). All duof swings must be felt as some multiple of this root duration. This is a rhythmic limitation, but could easily be generalized later by allowing for multiple distinct root durations which relate to each other in specific ways (this is what is called metric modulation). To set the length of the root duration in

seconds, use the function set-unit-pulse(t) and pass in the amount of seconds t.

2.2 An Overview of Labels

At their core, a label is simply a way to give a name to either a swing through time, or a grouping of swings through time. A label has two core properties: containment and duration.

- Containment specifies for a label, whether it is felt as a single swing through time (if so then it has no containment). If it has containment, the containment specifies the sequence of labels that are said to be contained within this label. Perceptually, this larger chunk of time is perceived with the concatenation of the perception of each of the labels within. This allows us to view rhythms as trees.
- The duration specifies the length of the total swing for the label. If it has containment, then the duration is forced to be the sum of the durations of the labels it contains. If the containment is empty, then the duration defines how long the swing is.

2.3 Simple Labels

A simple label is the name I give to a label that does not have containment. In my Nyquist model, there are three ways to create simple labels.

The function add-simple-label(name, duration) creates a label that has the name name (name is a string) and which corresponds to a single swing of length duration.

There is an important note here: it is not musical to view the duration as an amount of time. We are bad at perceiving time; however, we can perceive labels. The duration argument thus is not an amount of time, but a label. This begs the question, if you need to pass in a label to make a label, where do you get the first label from? There are two ways:

1. There always exists a label which is the single label with a duration given by the root duration. In Nyquist you can use *unit-pulse-symbol* to specify this label.
2. Alternatively you can use the function n-unit(n) to refer to a label which has duration n root durations and use the duration of this label.

Another way to create a simple label is using the add-simple-multi-label(name, duration-containment) function. This function lets you quickly create a simple label that has a duration which is based on the concatenation of duration of other labels. duration-containment is a list of labels such that the duration of the simple label with name name that is generated is the sum of the durations of the labels in that list.

2.4 Grouping labels

The difference between grouping labels and simple labels is in how they are perceived. A grouping label has a non-empty containment, and is perceived by the concatenation of the perception of the labels in the containment of the grouping label. To create a grouping label in my Nyquist model, there are two ways.

The first is to use the function add-grouping-label(name, containment) which creates a grouping label named name such that the containment is the list containment. Containment is just a list of labels.

The next is the function add-multi-label(name, containment, duration-label). This is a shortcut way to create a grouping label out of labels that don't yet exist. Because the containment labels don't yet exist, they don't yet have a duration. Thus, the duration-label argument is used to fix the duration of the entire grouping label, and the durations of the containment labels subdivide duration-label evenly.

2.5 Using Labels

From inside Nyquist, labels are just strings. Whatever name you give a label will cause that label to exist with that name. Thus, when a function takes in a list of labels, it's really just a list of strings. You must be careful that these strings correspond to labels that you actually made using the functions above.

3

SECTION

Voices

3.1 The Basic Idea

Once we have a way to talk about **when** things happen, we want to be able to talk about **what** happens. It is natural for us to perceive sounds that we hear as coming from different sound sources. We can view such a source as a voice. If we want to add sound to our rhythm, we can make a new voice using the function add-voice(name). This will create a voice layer in your piece that has name name (name is just a string). You will then be able to specify this voice to play certain notes at certain moments in time.

3.2 Pitch Specification

Usually the way that melodies are described is as a sequence of pitches in time. However, I don't think this is very musical, because it doesn't reveal the proper musical repetition that most melodies have.

The power of the label specification is that by making a tree of labels where each pair of nodes either have the same name or different name, we can view those moments in time as either being the same or different with respect to some dimension of sound. The dimension I programmed in this model is pitch.

I think the most musical way to specify a melody is to be able to make a statement of the form:

When this rhythmic pattern occurs, play these notes

And we have the ability to do exactly this because labels specify rhythmic patterns! This leads us to the pitch specification function:

voice-add-melody(name, through-labels, pitch-sequence). name is the string corresponding to the name of the voice you want to play the melody. through-labels is a list of labels. pitch-sequence is a list of builtin Nyquist pithches.

The effect that this has is that the voice now is sensitive to some labels if it sees them. Specifically, if it is asked to perform from a certain label, then as it traverses the label tree, the first time it sees the first label in through-labels, for every leaf in that label, it will play the first pitch in pitch-sequence. After that, the first time it sees the next label in through-labels, it will switch to the next pitch in pitch-sequence (and so on and so forth). If it exhausts either list before finishing the tree, such list loops back to the start. See comp.sal for examples as well as the Sections part of this document for information about how this leads to actual generation of sound.

3.3 Automixing

When the audio is computed for a melody, higher notes are weighted lower and lower notes are weighted higher. This is just a nice feature so that the user doesn't have to worry about it.

4 SECTION

Piece Layout

4.1 Sections

Now we just need to put it all together to make music. A section is a set of voices that all play together. For each voice, we need a label for that voice to 'play' from. To create a new section, use can use the function add-section(name, voice-and-labels). name is a string and specifies the name of the created section. voice-and-labels is a list of lists where each list has the first item as the name of a voice and the second as a label. The section corresponds to the player together of all the voices from their specified labels at the same time. Alternatively, you can use the function add-voice-to-section(new-section-name, old-section-name, new-voice-rhythm) to create a new section which is the same as a previous one but with one more voice added. new-voice-rhythm is a single list where the first argument is the voice name to add and the second is the label for that voice to play.

Finally, to get the audio for your music, you can use the get-music(sections) function. sections is a list of section names and it will compute the audio for each one in succession and return the result.