B557 Final Project: Monophonic Audio Segmentation and Resynthesis

Stephen Karukas
Github Project: github.com/resynthesis

Background

Digital instrument synthesis tends to consist of either recording many isolated acoustic instruments or creating a very detailed synthesis model. Libraries made these ways are often prohibitively expensive, and they don't include some aspects of human performance (notably, transitions between notes) due to the isolated nature of the source recordings. On the opposite extreme, simple sinusoidal models may be calculated from recordings of an instrument, but they don't account for change in timbre across an instrument's range or across time—though not very effective, this approach is cheap and automated. Despite their differences, the general idea is the same: take an acoustic instrument *F* and emulate it using a digital instrument *F'*. For this project, I use pre-recorded audio as materials for synthesized audio, an automated (though less controlled) approach than the first and a more realistic approach than the second.

Problem

As a formal description of the problem, an algorithm is given *F(X),* a recording of a monophonic instrument *F* playing an unknown piece of music *X*. It is also given *Y*, A passage of music given as a "score" (symbolic representation). These pieces are unrelated, though in my project I limited the recordings to only sustained, legato passages. The goal is to synthesize a recording *F'(Y)* that sounds convincingly like instrument *F* playing passage *Y*. A successful algorithm would be able to easily generate a versatile digital instrument by only taking in unlabeled input recordings.

One modification to this problem would include a score for *X* as input, but a pitch-tracking approach worked well enough that score-following was not necessary.

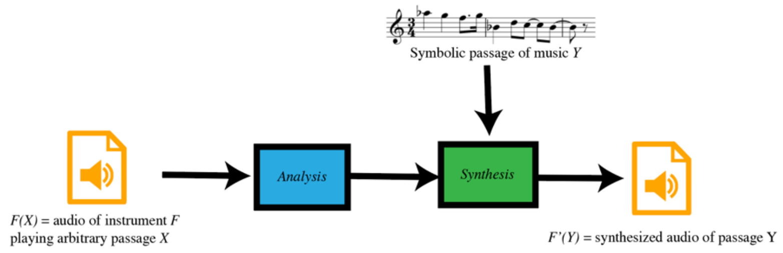
Symbolic passage of music *Y*
Analysis
Synthesis
F(X) = audio of instrument *F* playing arbitrary passage *X*
F'(Y) = synthesized audio of passage Y

Figure 1. General workflow for this problem.

Approach

My approach is to segment the audio, extracting sustained notes and the transitions between them. Between the analysis and synthesis steps is the storage of this extracted audio in the form of a "database" of WAV files, files stored by frequency. It is not assumed that the frequencies in *Y* will be in this database. The audio samples are later modified and spliced together in order to synthesize the output audio. Storing these values on the file system allows for multiple symbolic scores *Y* to be synthesized from the database.

I use an offline implementation where a score is delivered all at once rather than performed. With some optimization, it should be possible to adapt this approach to one that is near-real-time, though there would have to be a clever solution for sustaining notes for an arbitrary amount of time. It is currently implemented in R, and the source code is available in the /R folder of the Github project (the project is too large to include in this report).

I evaluated this project using three monophonic audio files:

- `R/samples/partita_flute.wav`, a recording of the Sarabande from J.S. Bach's *Partita in A minor*
- `R/samples/schindler_violin.wav`, a recording of *Jewish Town (Krakow Ghetto, Winter '41)* from Schindler's List, composed by John Williams
- `R/samples/stravinsky_clarinet.wav`, a recording of the first movement from Igor Stravinsky's *Three Pieces for Clarinet*

Analysis and Segmentation

In order to segment the audio input *F(X)*, my approach is to calculate *X'*, an estimated representation of the piece *X* that corresponds to the samples within *F(X)*. For this I use fundamental frequency information output by the probabilistic YIN (pYIN) algorithm [1], which uses shifting-lag autocorrelation followed by Viterbi decoding to calculate a stable time-varying f_0 estimation of an arbitrary signal. This algorithm applies well to monophonic instruments, as they tend to be highly periodic and harmonic, with a single fundamental frequency. The pYIN algorithm was very accurate on all recordings used in this project, so in general I will assume that its output is the true frequency sequence of the input audio during non-silent moments in the audio: $X' \approx X$.

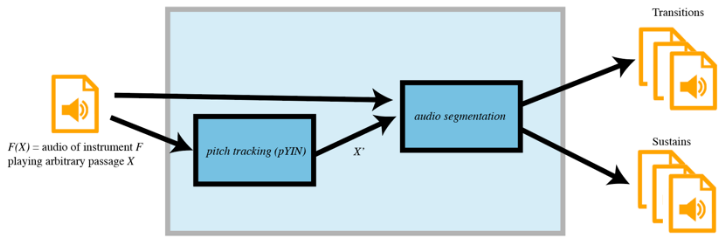

Figure 2. Analyzing and segmenting the input audio.

I tried a few different metric in an attempt to segment the audio automatically. I first thought of using amplitude information, but I learned it tended to vary most significantly during sustained notes, which made it unreliable for locating boundaries between notes. Another metric I used was spectral flux, which measures the L2 distance between consecutive normalized STFT frames:

$$\text{flux}(X,t) = \sum_{i=1}^{\text{ncol}(X)} \left(\frac{X[i,t]}{\sum_j X[j,t]} - \frac{X[i,t-1]}{\sum_j X[j,t-1]} \right)^2$$

The assumption was that transitions would contain a variation in the spectrum due to noise and changes in frequency. However, this was not the case; once again, plotting this sequence showed there was significant variation within the sustained notes, especially due the violin's expressive vibrato in `schindler_violin.wav.` It seems like this metric would work very well for detecting harsh onsets, but not legato transitions.

Because changes between notes tend to shift in frequency, it made sense that transitions may correspond to high values of absolute value difference along the pitch sequence p (the output of pYIN converted to pitch using the formula $p = 12\log_2(\frac{f}{440}) + 69$). For example, a transition could be detected when $|p_t - p_{t+1}| > c$ for some parameter c. This was successful, though it introduced a threshold parameter c that varied between recordings. My final choice was a greedy algorithm that adds samples to a range while their pitch is close enough to that of the current range:

```
function detect_notes(p, e, min_length):
    notes = []
    i = 2
    while i < length(p):
        r = a range, initialized to [i-1,i]
        do:
            i++
            expand r by 1
        while |p[i]- mean(p[r])| < e

        if length(r) > min_length:
            add r to notes

    return notes
```

My assumption was that notes that are distinct will be at least e semitones away from each other, where a normal value for e would be 0.5. The currently sounding note will have a mean pitch m, and once a pitch deviates from this range, it is likely not part of the same note. Despite the simple approach, this seems to yield accurate segmentations of the audio. To calculate the transitions, each consecutive pair of notes is checked to see if the distance from the end of the first to the start of the second is shorter than a predefined length. If this is true, a transition spans the gap between them along with some 0.05-second padding on either side. The transitions tend to be around 0.1 to 0.2 seconds long.

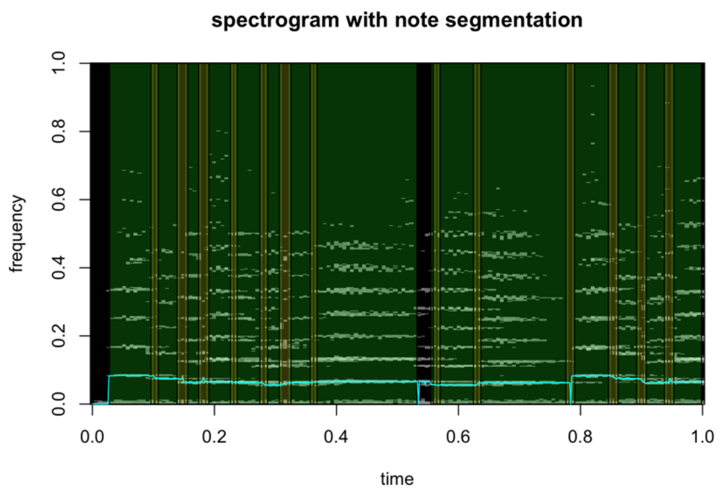

Figure 3. Segmentation of *Schindler's List* violin part.
Blue: f_0 estimations. Green: sustained notes. Yellow: transitions between notes.

After finding the transitions and sustains in the audio file, they are all extracted as WAV files to the file system. The sustains are identified with their decimal pitch (calculated as the mean across the sequence), and each transition is identified with the pitches of the notes on either side of it.

Synthesis

For synthesis of a piece of music $Y=(P,D)$, where P and D are pitches and durations, each pitch $p_i \in P$ is compared to the available sustained recordings. The recording that is closest in pitch to p_i is returned and pitch-shifted to p_i. Then the duration of the recording is stretched in time to match d_i.

To time stretch a sample, the STFT of the sample is calculated, then the first difference of the phases is taken. The STFT frames are resampled to match the requested duration. At the end of this process, the phases are accumulated. The phase differences remain constant, meaning the frequency content is preserved throughout the transformation.

For pitch shifting by a frequency ratio r, a sample is time-stretched by r then resampled in the time domain back to its original length. This has the effect of scaling the frequency content in the sample by r. I used STFT's with N=1024 and H=N/8 throughout this project.

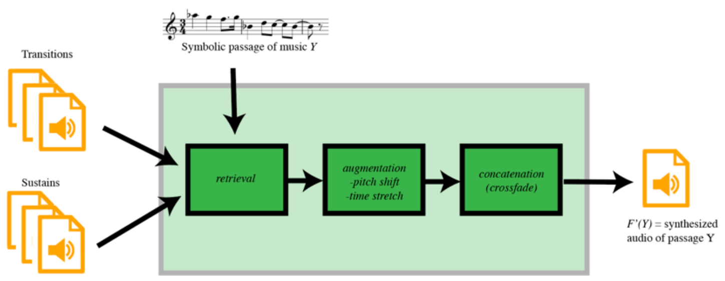

Figure 4. Synthesizing output audio by splicing together stored WAV files.

Transition recordings are recovered by finding those whose interval is the same size in semitones. These are then pitch-stretched to the desired starting pitch. If no such transition exists in the database, none will be synthesized.

To concatenate each pair of transformed time-domain samples s_i and s_{i+1} with a transition z_i, I used an equal-power crossfade between s_i and z_i with overlap $\frac{|z_i|}{2}$ to produce a new sequence b_i. I then crossfaded between b_i and s_{i+1} using the same overlap size. The equal-power crossfade ensures the squares of the envelopes sum to 1, which sounds like a continuous transition to human ears:

$$\text{crossfade}(x, y, t) = x_t\sqrt{1-t} + y_t\sqrt{t}$$

where *x* and *y* are the overlapping portions of two signals, and *t* runs from 1 to $|x| = |y|$.

Results

Only the first 60 seconds of each input audio file were used to generate the synthesized output. Each recording was segmented and tested on three scores: the beginning of *Amazing Grace*, a major scale starting in the low end of the instrument's range, and the same scale an octave higher. It's difficult to quantify results for this problem, so only subjective thoughts about the results can be given:

Clarinet

- The synthesized clarinet for *Amazing Grace* sounds realistic, though there is occasionally a bit of reverberation at the beginning of notes.
- The scales have discontinuities across its range due to some samples being taken from louder sections than others. There is also the issue mentioned above.

Flute

- The synthesized *Amazing Grace* suffered from some discontinuities in amplitude, but otherwise generally has smooth transitions between each note.
- The scales do not sound smooth, and that is because no transitions could be found of interval 2, 1, or -1 in the selected portion. Descending whole-steps however (-2 semitones), do sound smooth. This is a property of this recording, and using more of the file may have allowed a smoother outcome.

Violin:

- Results from the violin recording were not as good, and often the reason was that there were hints of multiple notes sounding—the input violin recording was not completely monophonic, and it also contained some reverb. In addition, it did not "flow"—the violin playing in the recording was very expressive, but when segmented these dynamic changes became abrupt changes between notes.
- The trait about the violin recording that stood apart from the others was its use of wide vibrato and glissandos, but this did not seem to be the cause of the poor results.
- In addition, the extracted sustained notes were not long enough, and adjusting to this with time stretching led to unrealistic artifacts.

- Despite these poor results, by comparing `synthesized/scale-violin [no_transitions].wav` and `scale-violin.wav` one can see the importance of including the transition samples in the synthesis. The latter is a much smoother scale.

Overall, it seems like the main issue with this method is the reliance on only pitch during the segmentation process, as opposed to duration, articulation, and shape. Within none of the results were there errors in pitch, but most had errors related to the other parameters.

Future Work

Automated splicing of audio may be a good-sized final project, but it can only go so far. Here are some other possibilities I hope to explore in the future:

- **Generative Model:** Instead of a database, learn a generative timbral model through analysis of the STFT as a sequence. An attempt to "learn" timbral changes over time, if successful, could allow for synthesis with much more flexible parameters, notably duration and shape. The difficulty of separating timbre from frequency is a hurdle I faced when exploring non-splicing methods for this project, and I don't believe this extension would be easily approachable without a solution.
- **More Synthesis/Analysis Parameters:** One of the main issues with this project was the discontinuity of "line", or varying intensity, between adjacent notes. If segments were analyzed further and stored with more verbose attributes, there could be finer control over the more "human" aspects of the synthesized sound.
- **Other Improvements:** Because this is a "pipeline" sort of process, parts could potentially be switched out or collapsed. For example, it may make more sense to segment the audio while performing pitch extraction. In the interest of time, I used the pYIN algorithm as a black box.

References

[1] M. Mauch and S. Dixon, "pYIN: A Fundamental Frequency Estimator Using Probabilistic Threshold Distributions," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2014)*, 2014.