# Music Box Generation from MIDIs

6.S079 Midpoint Project Report

Trevor Walker and Tiffany Lu

## Motivation

The problem we are tackling with our final project is allowing users to generate custom music boxes by simply uploading a MIDI music file containing the melody. The key goals in this task are to decrease the cost to the user of creating a custom music box and to make it easier to generate boxes by allowing a user to upload the desired song instead of having to manually input each note and the rhythms while still maintaining good sound quality. The custom 3D-printed music box was suggested in class; however, inputting the song for the music box is very clunky, and the fully 3D-printed box does not sound very good due to the plastic's thin sound, so we plan to integrate a steel comb in order to make the custom boxes sound like traditional music boxes.

## Technical Approach

In order to generate the custom music boxes, we will build a node.js web application that allows the user to upload a MIDI file and will output the .STL and .DXF for 3D printing and laser cutting. There are four main components to this: the JavaScript MIDI parser, the cylinder generation, the comb generation, and the box generation. The web application will provide an interface that allows users to upload a file and then subsequently view and download the resulting files.

### *MIDI Parser*

The role of the JavaScript MIDI parser is to extract the relevant note and timing data from the MIDI file to pass to both the cylinder and comb modules. The web application provides an input field for the user to upload a file, and once chosen, it is read as a binary string and fed into a JavaScript MIDI parser called jasmid (https://github.com/gasman/jasmid). The parser surfaces a "header" portion of the MIDI file, which contains the MIDI format, number of tracks, and the timing interval, as well as a "tracks" portion of the file, which contains the actual note data. The "tracks" property contains a list of events for each track. We use only the MIDI "note on" event to signify when a note should be played, ignoring the "note off" event because a music box cannot sustain notes.

To generate the comb, we require all unique notes that are played through the whole song, so we iterate through the "note on" events to gather the unique MIDI note numbers. To generate the cylinder, we require [comb-tooth number, timing] pairs. We use the list of unique notes to get the comb-pin numbers, and to get the timing, each event also has a delta-time property, which indicates the time between the current event and the last, so we build the [comb-tooth number, timing] pairs by accumulating those differences. These two arrays are passed on to their respective modules.

*Comb*

The comb component is a Python script that takes in the list of unique notes in the song in the MIDI note number format and outputs a .DXF file for laser cutting. The lengths of the teeth on the comb are calculated by taking the MIDI note numbers and converting them to frequency, and using the following equation to calculate the length:

$$L = \sqrt{0.162\frac{a}{f}\sqrt{\frac{Y}{d}}}$$

where a is thickness, f is frequency, Y is the Young's modulus, d is density, and L is the length. For the stainless steel we intend to cut the comb out of, the constants are:

$a = 0.36$ in

$d = 0.289$ lbs/in$^3$

$Y = 2.85 \times 10^7$ ksi

The output .DXF is created using the `ezdxf` Python library.

*Cylinder*

The cylinder component is an OpenJSCAD routine, which takes in the [comb-tooth, timing] pairs as described above and generates a 3D-printable cylinder (.STL) with pins in the corresponding locations. The spacing between the pins both along and around the cylinder is variable and depends on the length of the song and the number of unique notes the song contains.

We calculate the spacing of the pins around the cylinder using the length of the song and the circumference of the cylinder, and we calculate the spacing of the pins along the cylinder using the number of unique notes the song contains. To place the pins on the cylinder, we iterate through the array of pairs and calculate the angle offsets depending on the timing.

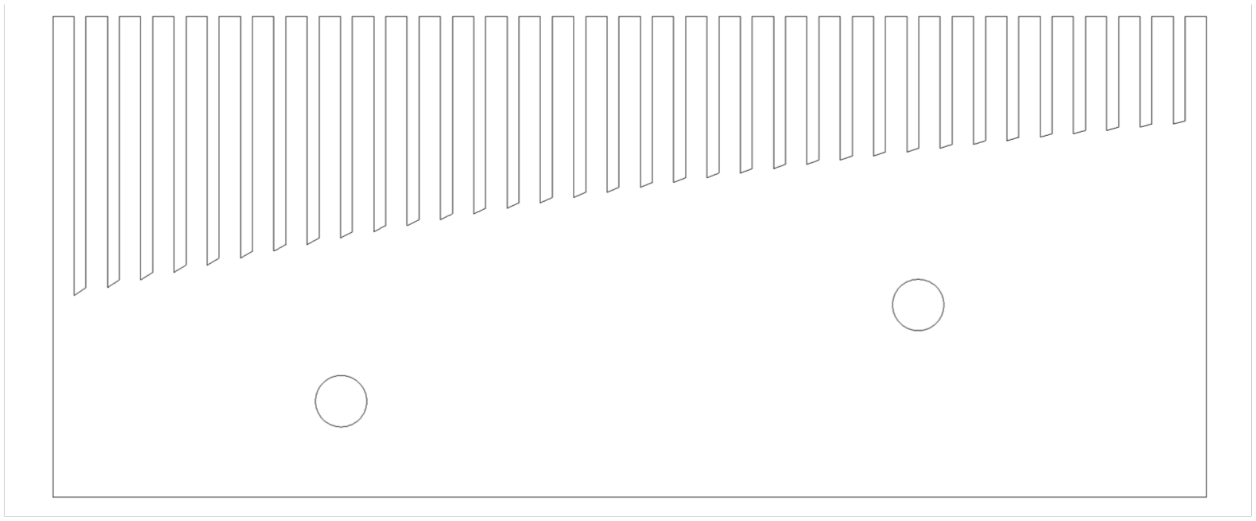
*Music Box*

The box component is designed statically as a CAD file, which provides slots to insert the cylinder and comb. The box uses a worm drive attached to a crank to turn the cylinder.

## Progress & Current Results

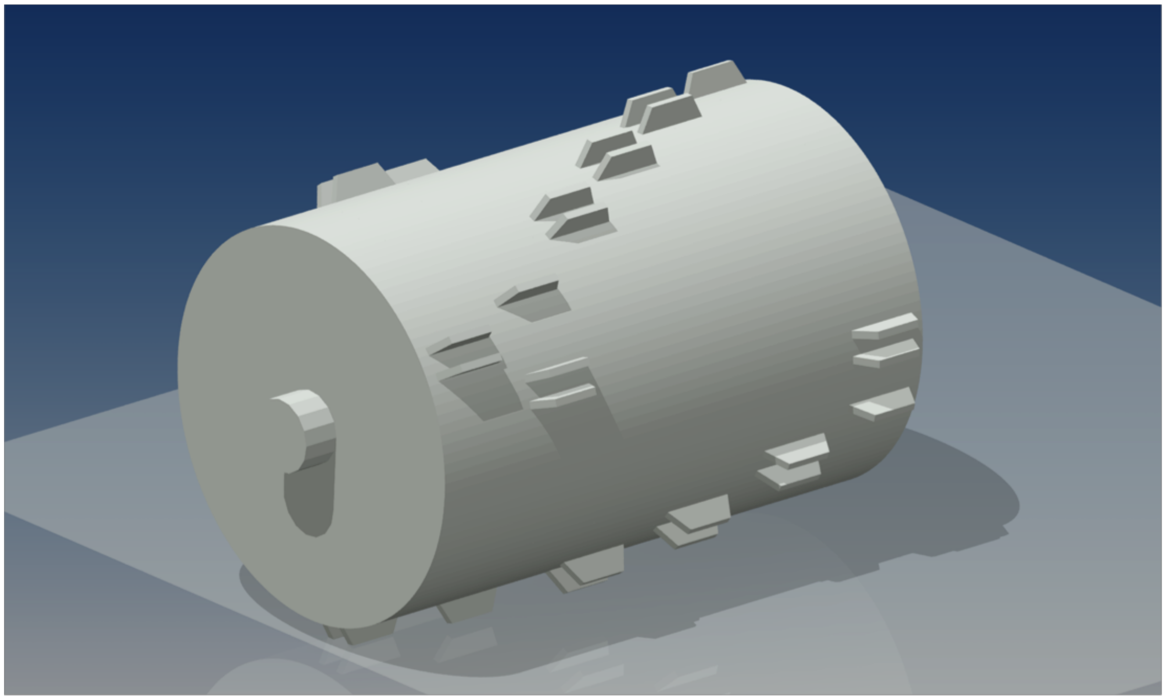
Currently, each of the separate components as described above are implemented, but they have not yet been integrated with each other. The MIDI parser is currently a static HTML page that takes in a user-uploaded file and outputs the relevant data to the console.

The comb Python script takes in a list of notes to put on the comb and outputs a .DXF file as seen below.



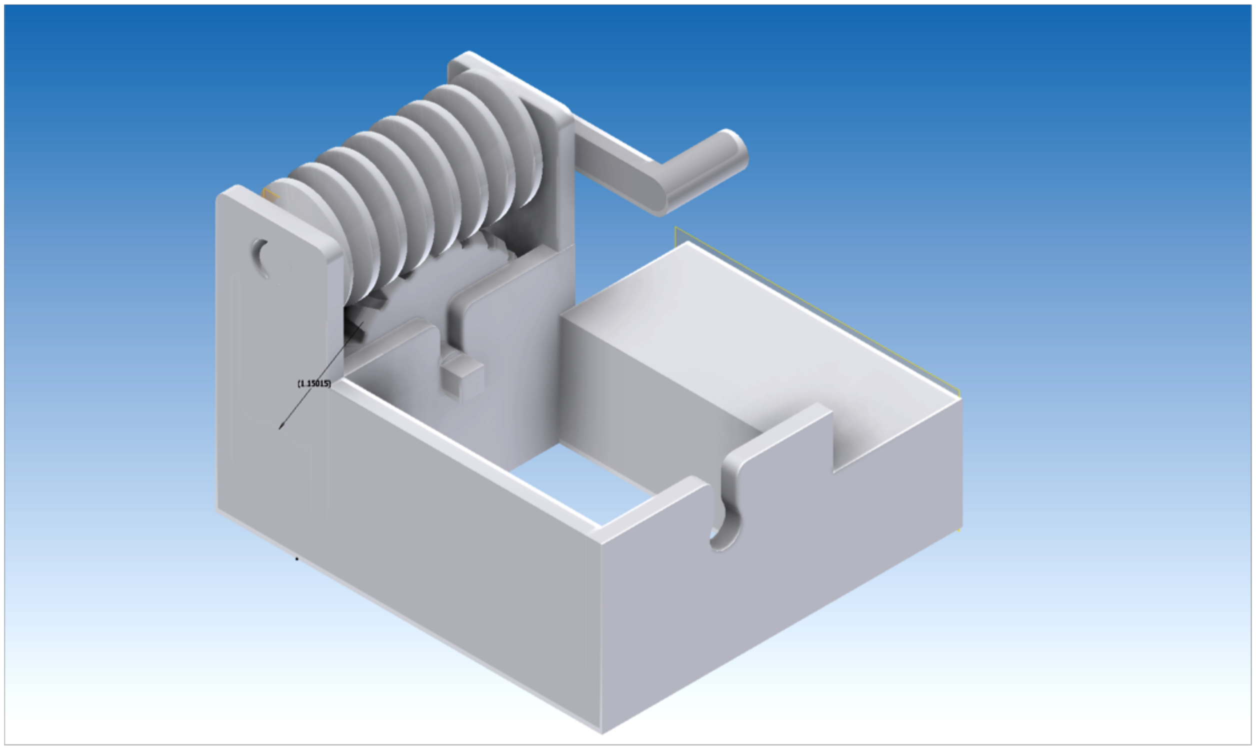
*Example generated comb .DXF*

The cylinder takes in the note, timing pairs into an OpenJSCAD routine and produces an output as can be seen below with a post and square indent on either side to fit into the box.



*Example generated music cylinder for Twinkle Twinkle Little Star*

And finally, a render of the music box can be seen below.



*Music box render*

<u>Next Steps & Expected Results</u>

The next steps for this project are to connect all of the four components together into the final web application. We also need to print and laser cut the generated objects to ensure they work together and make adjustments as needed. We plan to evaluate the laser cut comb by comparing the output frequencies with the desired ones, and we will compare the sound quality and overall function of the music box with both real and fully 3D printed ones.

We expect our final product to allow the user to upload a MIDI file and be able to download their corresponding .STL and .DXF files in the same interface. We also expect to have a few printed music boxes, generated by our system.