Tiny Book of Technique Report

Alexander Bell & Luke Bodwell

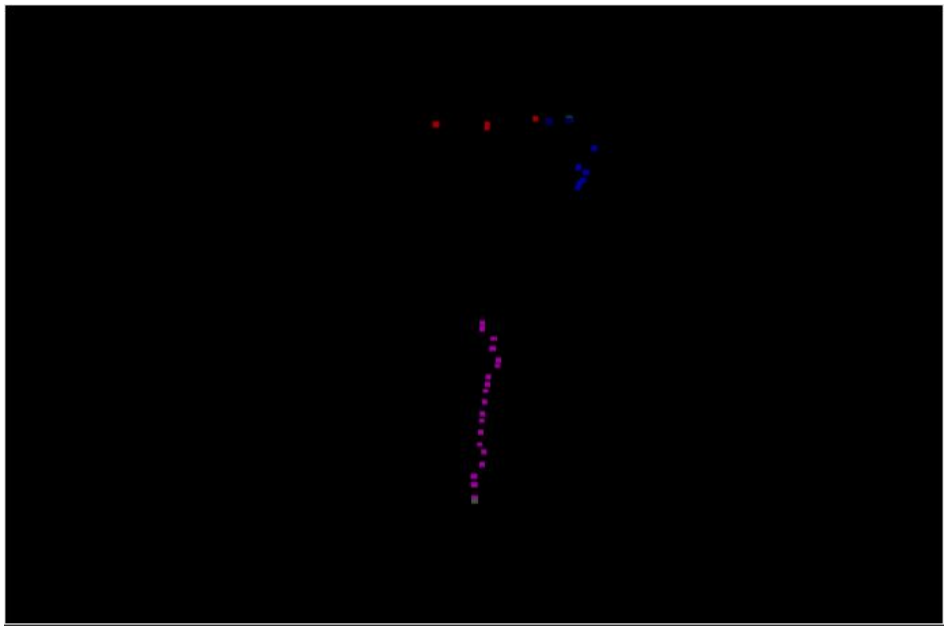
Intro

For our work on *The Tiny Book of Technique* we prioritized two main features, those being a particle-based visualizer and the ability to dynamically load lesson content into the application and split up these tasks between the two of us. Alexander worked on the visualizer and Luke worked on the loading of lesson content.

Running the Visualizer

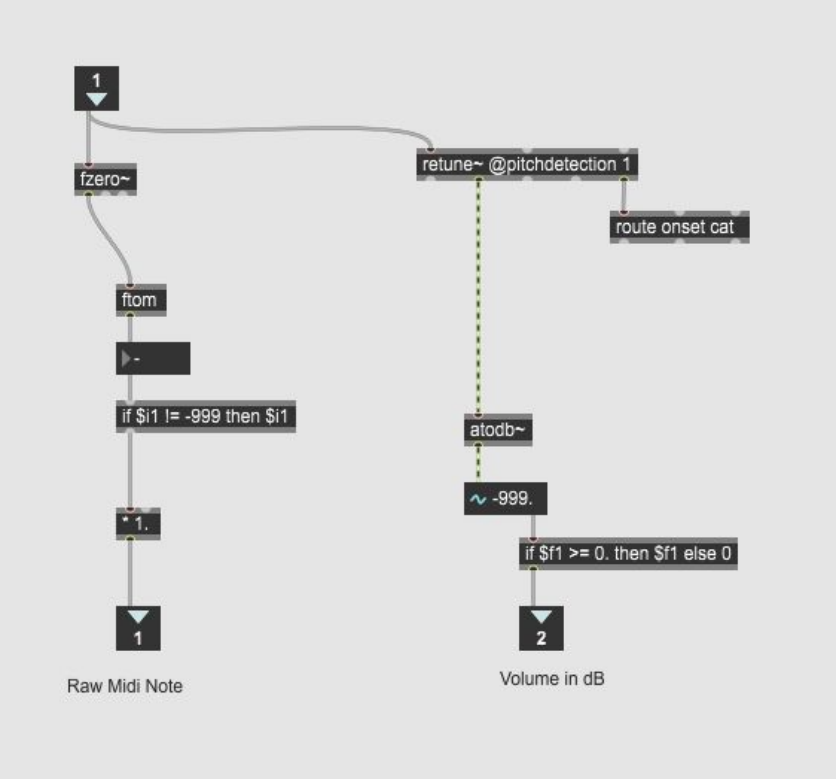
To run the visualizer portion of the program, a separate package named “gl3” from Max must be installed. It can be found by opening File->Max Package Manager and in the top right searching for gl3. Once it is installed, open up Max preferences by navigating to Options->Preferences. Then, make sure you have all selected at the top, and scroll down to the section labeled “Jitter”. From there, change the selected value in “OpenGL Engine” to gl3.

Visualization



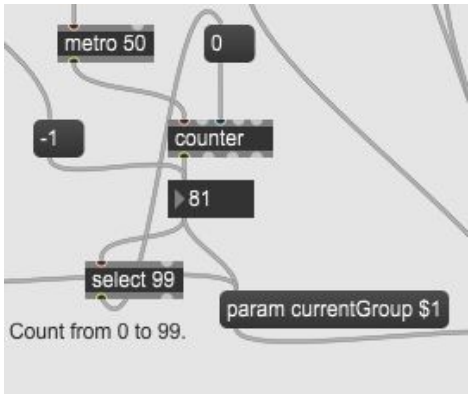
Example of the Visualization Window

For the Visualization portion of the software, we used the framework from a separate music visualization project used by WPI students. At the top, in green, and right side, in blue, of the visualization sections are where the default values exist. These can be easily set to a different message currently by changing what's in the message box, then clicking the button to pass the message along to the shader program. Most of the work was done in the shader, which is an external file called visualization_shader.jxs. This file was completely changed from the original version. Now, it takes in a pitch from the signal that has been converted to a number, and the volume of that note in decibels. It calculates what the color should be based on the pitch and how fast it should move based on the volume, and pseudo-randomly chooses an x velocity, also based on the volume. Currently, only the information from the subpatch named "split-notesL" is used, but since most recording devices are single channel, this shouldn't pose a large issue.



split-notesL subpatch

To calculate when a particle should be sent, a metronome is started and every time it changes number, it will alert the shader with the message of which particle has been called to move. Each particle is assigned a number when the program is first loaded. If the metronome count is equal to the particle's assigned number, then the shader uses that as the sign that it's time for the particle to be sent. The particle will reset to the default position (0, -0.5, 0) if the particle were to stop moving or start moving down.



The metronome system, as described above

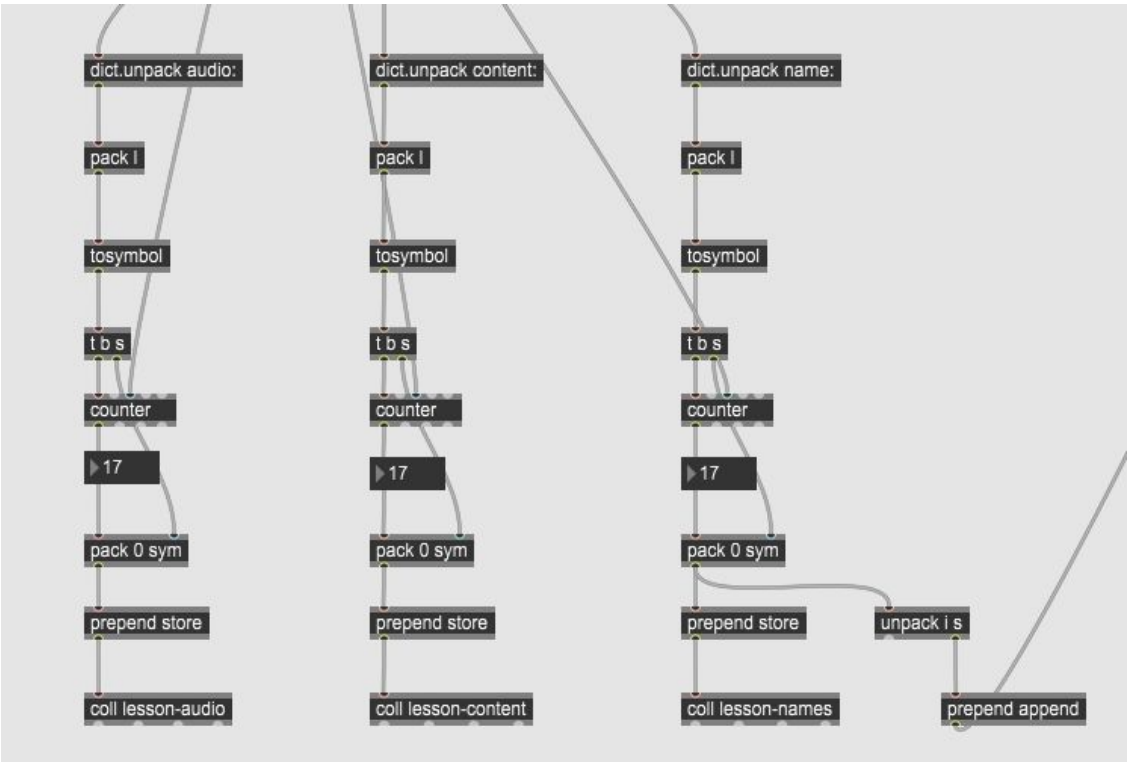
Lessonization

```
{
    "lessons":
    [
        {
            "name": "Day 1",
            "content": "day1.html",
            "audio": "day1.wav",
            "visualization-settings": {
            }
        },
        {
            "name": "Day 2",
            "content": "day2.html",
            "audio": "day2.wav",
            "visualization-settings": {
            }
        },
        {
            "name": "Day 3",
            "content": "day3.html",
            "audio": "day3.wav",
            "visualization-settings" : {
            }
        },
        {
            "name": "Day 4",
            "content": "day4.html",
            "audio": "day4.wav",
            "visualization-settings": {
            }
        },
        {
            "name": "Day 5",
            "content": "day5.html",
            "audio": "day5.wav",
            "visualization-settings": {
```

lessons.json File

In order to “lessonize” the application, we decided to use the dictionary object (dict) in Max to load in lesson information dynamically from a JSON file holding an array

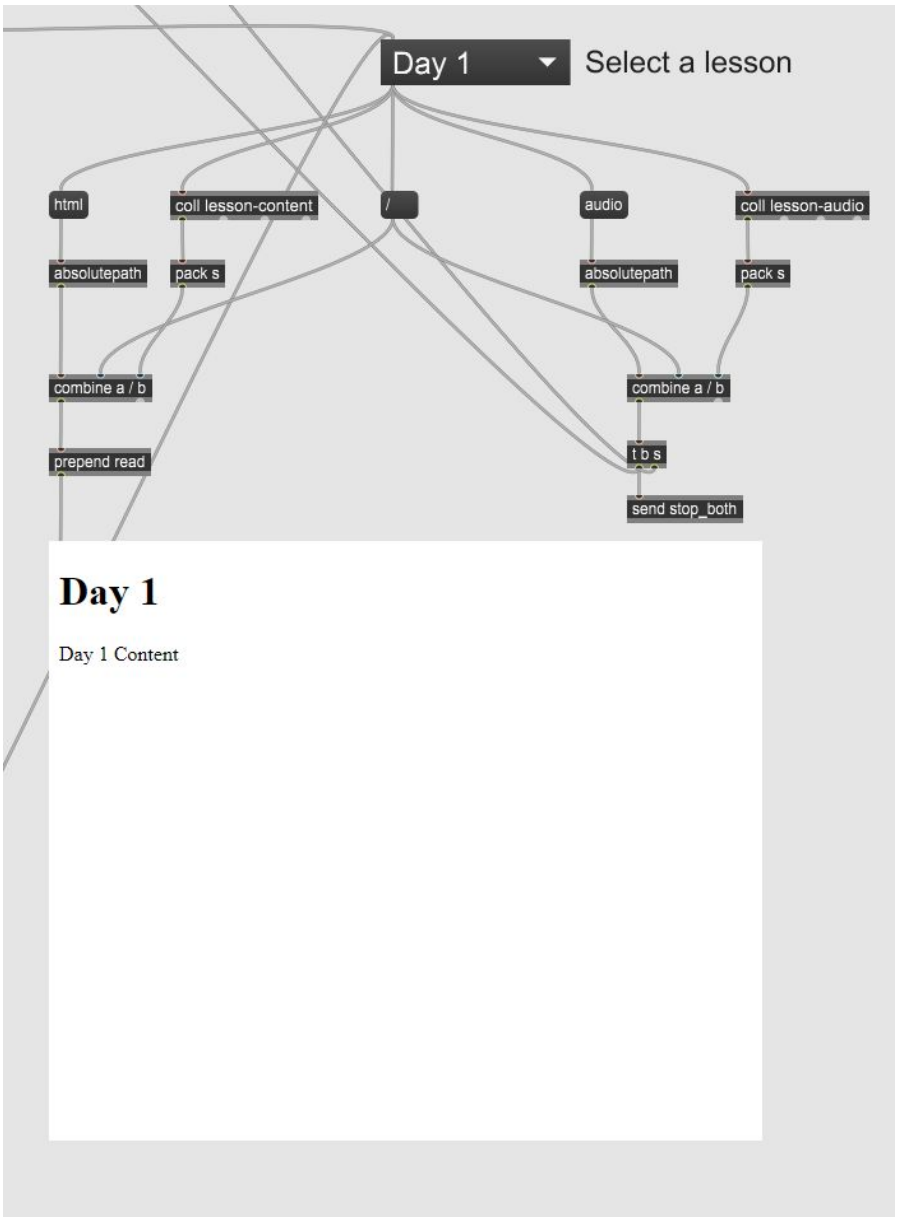
of lesson objects. We wanted to avoid hardcoding this information in favor of a more extensible solution. This approach allows Wendy to easily add new lessons or modify existing lesson content without having to touch anything in the Max patch itself. Once the lessons.json file is loaded into a dict, it's iterated through and object keys are unpacked in order to place their values into various collection objects (colls). A counter is used to properly index into these collections during iteration. Currently, lesson names and the names of the lesson content and lesson recording files are stored in collections. There is an empty sub-object in each lesson object called visualization-settings. Wendy expressed some interest in being able to easily change settings for the visualizer in different lessons. While we didn't have time to fully implement this feature, key-value pairs could easily be added to these sub-objects, loaded into a sub-dictionary, and iterated through for each lesson. These values could be passed in to the visualizer to control behavior dynamically. Any other desired attributes could likewise be added to the lessons.json file in the future to support new features.



Lesson data unpacking

Once the collections containing different attributes of the lesson objects are filled, the lesson names are sent to a umenu object to populate the drop-down in the UI. When a lesson is selected, its index in the umenu is sent out the first outlet and is used to obtain the other attributes from their respective collections as the indices will be the same across all of these objects. The absolute path on the user's computer of the html and audio directories in the root directory of the patch are determined and the names of

the content and audio files are appended to these paths. For some reason the absolutepath object was unable to find the files when given a path that included a subdirectory. For this reason it is assumed that all files will be placed in the expected directories and only the names of the files themselves are stored in the JSON file. The HTML file path is sent to a jweb object to render the lesson content using an embedded browser and the audio file path is sent to the buffer handler to be prepared for playback and visualization.



Drop-down propagation and file loading

Future Work

There are a number of things that could be improved upon in the work we completed so far. Firstly, the current implementation using the metronome + counter system in visualizing the audio has a number of bugs associated with it. The most easily found bug of this type is when audio is stopped/paused and the notes reset to their original position, one to three more particles will be sent out afterwards. This was temporarily fixed by sending the message to reset again 0.1 seconds after the first reset message was sent.

Currently, the contouring of the pre-recorded file does not update when the lesson is changed. The contouring is set by the values in the example file and represents the Day 1 Lesson Example. In addition, the values passed on by the lessons.JSON file under the visualization-settings object are currently empty and not attached to the messages being passed to the shader. These two issues can be fixed, however require some time and reorganization to do so.

Wendy expressed some interest in having a second dropdown menu to select exercises within a lesson. This would allow lessons to have individual recordings for each exercise instead of having one very long recording. We didn't have enough time to implement this, but it could be done by adding an exercises array to each lesson object that would hold a list of exercise objects containing names and paths to audio files. The one audio file attribute in the lesson object would be removed and some changes would have to be made to how the dict object is processed. New collections could be used to store subcollections for both exercise names and audio paths unpacking the exercises array from the dict object. These collections would be used to propagate the second drop-down menu with exercise names and load in the correct audio file for each exercise.

Conclusion

We accomplished our two main goals of creating a particle-based visualizer and lessonizing the application. We tried to implement these features with extensibility in mind. To achieve this, a JSON file is used to store lesson information that is easily editable. The Visualizer was built with many adjustable values that can be loaded in from the JSON file as desired. These two goals were combined with the framework set up by previous students working on the Tiny Book of Technique project in order to provide an extensive visualization of audio both on disk and recorded live.