Pausch Bridge MIDI Visualizer

Spring 2016, Group 3

1 Abstract

This document describes a MIDI-based music visualizer that can be run on the Pausch Bridge. The lights on the bridge mimic the keys of a piano, with different pitches being mapped to corresponding positions along the length of the bridge.

2 Running the Project

2.1 Dependencies

The project is in Python, and requires two additional Python libraries: `pysmf` and `pyOSC`. Additionally, the visualizer requires `PyOpenGL`. All of them can be installed through pip. (You may need to `pip install pyOSC --pre` if the problem with the version numbers hasn't been fixed yet.)

2.2 Overview and TLDR

The main project is contained in the `midi/` folder. Suppose we have a file `foo.mid`. If we want to turn it into a light show and view it using the local visualizer, we need to complete the following steps:

1. Convert `foo.mid` into `foo.mid.notelist`: `python SongReader.py foo.mid`
2. Start the server: `python BridgeServer_GUI.py`
3. Send the notes to the server: `python SendMidi.py foo.mid.notelist 0 localhost`
4. (Optional) Play an audio file that will be synced to the light show.

If we want to play `foo.mid` on the bridge instead of viewing it locally:

1. Convert the file as before.
2. SSH to the bridge and start the server there: `python BridgeServer.py`
3. Either from your local computer, or from the bridge itself, send the notes: `python SendMidi.py foo.mid.notelist 0 pbridge.adm.cs.cmu.edu`
4. (Optional) Get some very loud speakers and sync your audio to the show.

2.3 Converting MIDI

The MIDI standard is generally too messy to convert directly into lights on the bridge, so we convert it to an intermediary format `.notelist`, which strips out all of the cruft of typical MIDI files and leaves only basic information about notes, such as their times,

pitches, and durations. The script `SongReader.py` will perform this conversion. It can be used with the command

```
python SongReader.py foo.mid
```

where `foo.mid` is the MIDI file we want to convert. It will then create `foo.mid.notelist` in the same directory, which the rest of the system will be able to use.

2.4 Starting the Server

We provide two servers: one (`BridgeServer_GUI.py`) that displays a visualizer on the local machine, which is useful for testing out different songs, and one (`BridgeServer.py`) that is meant to run on the bridge and interface with the lights. Neither of them take any command-line arguments, so they can be started by typing

```
python BridgeServer[_GUI].py
```

The GUI version will initialize itself on localhost by default, while the actual bridge server will initialize itself on `pbridge.adm.cs.cmu.edu`. The servers will only accept packets sent specifically to the address they are initialized on. Practically, this means that if you want to stream a light show from the bridge to itself, you can't send to localhost – you have to send packets explicitly to `pbridge.adm.cs.cmu.edu`.

2.5 Streaming the Notes

Once the server is running either locally or on the bridge, you can begin sending the notes to the bridge. The script to do this is `SendMidi.py`, which can be used with the command:

```
python SendMidi.py <notelist> <wait time> <address>
```

Here, `notelist` is the `.notelist` file we created in step 1, `wait time` is the time in seconds that we should wait before sending the first note, and `address` is the network address where the server is running. In particular, this address should be localhost if we are running the GUI server, and `pbridge.adm.cs.cmu.edu` if we are running the server on the bridge.

The wait time is useful in case we want to sync the light show with an audio file (e.g. an mp3), but the audio file has some leading silence. By specifying a short delay, we could delay the start of the light show until the start of the actual audio, assuming that we start playing the song at the exact same time that we start running `SendMidi.py`. (However, the better solution is just to trim the audio file so that there isn't any leading silence.)

So, if we want to use `foo.mid.notelist` as the light show and we already have the server running on the bridge, we would type `python SendMidi.py foo.mid.notelist 0 pbridge.adm.cs.cmu.edu`. If we simply wanted to preview the show on our own computer using the visualizer, we would type `python SendMidi.py foo.mid.notelist 0 localhost`.

Note that the script gives a 5 second countdown before starting the show. You can use this in case you need to start music manually at the right time.

2.6 Combined script

We have a small script `run.sh` that can sync the start of the show with the start of an mp3 file. The syntax for this script is

```
./run.sh <notelist file> <mp3 file> <address> <midi delay> <mp3 delay>
```

The first three arguments are self-explanatory. The MIDI delay is the time in seconds to delay before starting to send the MIDI, in case the MIDI begins immediately but the mp3 has leading silence. The mp3 delay is the reverse – the time to delay before starting to play the mp3, in case the mp3 begins immediately but the MIDI has some leading silence. (Again, it's better just to edit both so that they start immediately. If you do, then both arguments can just be 0.)

3 Conclusion

If you have questions, email Chris at `christoy@cs.cmu.edu`.