Rafi Khan, Mike Levine, Jacob Metrick, Hari Ganesan

Mobile Computing, CPSC 434

December 13, 2012

Soundulous

**I. Initial Concept**

Music visualizers have already existed for media players, and recently people have made them on mobile devices as well. Our concept took this idea several steps further: make a music visualizer which a) responds to touch events and b) can be synced to multiple phones.

We began with two possible options to connect to the server: *new* and *join*. With /new, a user begins a new session as the host of the session and either declares a session ID or is given one. With /join, any other user can join that session if they know the session ID created by the host. Once our host is logged into a session, he will pick a song to play. Then, everyone who is logged in or logs in afterwards will be able to see the visualizer and hear the music from their phone, including the host. Then, whenever someone touches his or her own phone screen, it will affect the visualizer on everyone’s phone. If two people touch at the same time, the effects will mix. The song continues, and at any point, the host can add new songs to the play queue, which will load new songs after the current one is finished. If the host leaves, the session is over.

We think the main audience for this app would be teenagers and young adults, and the main use-case would be at a party or event. We think the use-cases stretch beyond that as well; if I’m listening to music on the go, I can just use the app in order to listen to music and simultaneously play with the visualizer. Then, if I meet a friend of mine, I can ask him to join my session.

**II. Technical Challenges**

The trickiest part of our concept was speed – being able to transfer images to a phone while recording touch events and syncing them from multiple phones? This seems like a lot for one server-client model to accomplish within a small time frame. We then sat down and thought of which protocol we wanted to use, either HTTP/TCP or UDP. I’ll briefly highlight the differences of these two approaches. TCP, the more standard protocol, is nice because it requires a handshake, and therefore the server and client both know when packets of information are being sent and when they are being received. However, TCP is much slower because it requires a handshake for every connection it makes. UDP, on the other hand, does not require a handshake and therefore is much faster, especially for rendering graphics and streaming a lot of data at once. However, UDP is not exactly “secure”, in the sense that its not verified when data is sent and received. We ended up using a combination of both, because they both had their advantages, depending on the situation that we required.

We also needed to consider how to make and adapt the visualizer based on the inputs: the song, and the touches and swipes coming from all of the phones. We felt the best way to have the touches affect the visualization was to have each touch signal a change in either volume (decibel level) or a change in the shape of the FFT on the song that is being put through the visualizer.

**III. The Client**

So our model begins with the client. We can post to /new or to /join, and our server will then respond with the port number of the session we’ve either created or joined, so that we can open a HTTP connection and begin to render the images.

Once we’ve joined a session, we can handle various touch events by either recognizing multiple types of events, such as swipes and touches, or simply recognize all events as touches and send them as a set of X-Y coordinates. We picked the latter because we wanted as much processing to be done on the server as possible. Later, we will need a lot of processing power in order to render the images from the visualizer, so the more the server does, the better. Once all of the events are recognized as touches, we sent them via a UDP connection with our server; we simply specify an IP address and a port, and we get the information whenever it is sent in whatever format we specify.

**IV. The Server**

Now that the server has received this information, it needs to do several things: 1) distinguish touch events, 2) interpolate these events as inputs into the visualizer, 3) sync these inputs with the inputs of the song into the visualizer, and 4) send the visualizer as well as the music to the phone. We’ll walk through this step-by-step.

First, we need to get the packets of information from each client and combine the touch events, as well as distinguish which events are “touches” and which are “swipes”. We do this by creating an array of the coordinates and then checking against them. Once we have distinguished them, we can affect the visualizer in the following way: for touches, we use spikes in volume, determined by a rolling average to avoid false hits, in order to change the color output by the visualizer. Swipes or flicks make the image move temporarily before moving back to the center. This is intuitive because one expects that kind of movement after swiping on a mobile device in a particular direction.

However, we are far from done; we need a few more threads. One will do the actual rendering of the images. Another one will receive video data from Processing, the visualization engine that we are using as a basis. We will also need one more to send all of our video data back to the client – to make a total of four threads.

**V. The Visualizer**

We’ve thus far glossed over the brunt of our work: the actual visualizer. Most visualizers work by doing an FFT analysis on the music, and responding to increases in volume over difference frequency ranges. Due to a lack of time, we couldn’t implemented a robust frequency analysis, but we do have very well flushed out volume controls, using the built-in Processing library Minim. Currently, the visualization will “explode” a bit on volume increases, that is when the value from the sound analysis exceeds some value. However, since we’d like to measure volume *changes*, not absolute value, we’ve included a MovingThreshold class that dynamically adjust the threshold as the volume increases or decreases. The rest of visualization was borrowed from an online example.[[1]](#footnote-1)

Compatibility with Android and Eclipse was a bit of an issue—we had to delve into the source code of Processing to see precisely what was happening and adapt our program to handle those cases. In particular, in the Processing IDE, all classes can access the screen, whereas that is not the case in Android. The renderers are also different between the phone and the IDE, so some minor visualization differences are present.

As mentioned previously, it would have been ideal to send video data, so all the phones could independently watch a meaningful visualization. However, Processing’s support for videos is not quite complete yet; therefore, we had to resort to sending just JPG image data.

**VI. Final Results**

We created most of what we set out to do; the trouble we ended up running into was the rendering of images by the Android OS once we had already created the visualizer and changed the inputs accordingly given touch events. The UDP connection didn’t seem to work, so we had to switch back to TCP, and it resulted in a much slower, somewhat “strobe-light”-esque connection. We have to implement the join feature, but that would only take a few lines of code to direct the request to the right location.

**VII. Future Considerations**

There are quite a few things we would need to consider for extending the scale of the app. We would need to think about, for instance, how many people we should allow in each session. If we let a session go out of control, there may be delays, or even worse, the server or the app may crash. What we could do as a way to counter this limitation is allow there to be people logged into who are not able to affect the visualizer as “viewers” rather than “enablers”. We could also implement both public and private sessions, so that one could not only make a private session with one’s friends, but they could also join an existing public session from other people all around the world. More options that we kept in mind were expanding to multiple visualizers, and making the touch functionalities different as well, including detecting other types of gestures and such – we need to make sure that our visualizer responds via each touch event to what closest resembles the user’s expectations.

1. You can see the adopted version at <http://www.openprocessing.org/sketch/5989> [↑](#footnote-ref-1)