Technical Documentation

Last Updated 01/05/12

System Architecture

The system is broadly divided into two components, the audio engine and iPhone application. The audio engine is implemented as a series of pure-data patch files which provide functionality relating to audio playback, loading and saving audio files and synthesised instruments. The iPhone application is written entirely in Objective-C and provides the application’s user interface and the algorithms for extracting image properties and translating them into drum patterns and melodies.

The following diagram shows the static state of all (non-utility) objects in the application, a detailed description of each follows:



# Audio Engine

The audio engine is broken down into 4 key objects which are represented on the diagram. These are contained within the “.pd” files included in the source repository. Pure-Data introduces new terminology for what object oriented programming calls “objects”. Pd files which are intended to operate as a standalone program are called “patches”. Objects which are intended to be a component in a patch are called “abstractions”, and within either object there may be a number of sub-patches. In the diagram above, SoundSystem is the patch which provides the audio functionality behind our application. Polytracker, Vocoder and SimpleInstrument are abstractions. The pd files may be modified in either Pure-Data or Pure-Data extended which may be downloaded at <http://puredata.info/downloads>.

### SoundSystem

The Objective-C application loads the SoundSystem patch and libpd fetches any required abstractions behind the scenes. The SoundSystem patch provides the interface to all functionality required by the application and does so through responding to messages provided by the message system of libPd. This includes playback controls, controlling song length and instruments, audio playback and capture to file. SoundSystem accepts the following messages:

* startPlayback – The application should send this event to begin playback
* stopPlayback – The application should send this event to stop playback
* pausePlayback – The application should send this event to pause playback, the application itself keeps track of the play, pause and stopped states for updating the UI.
* recordSong – The application should send this event to begin recording to a file. It is important the application sends no other events during this as it could cause various bugs. We solve this issue by displaying a “saving..” dialog and disabling input until completion. Playback progress notifications are used during recording to update a progress bar.
* $0-melodyVolume – Controls the volume of the Vocoder instrument
* $0-drumVolume – Controls the volume of all SimpleInstruments
* $0-numInstruments – Allows instruments to be disabled, currently always set to 5
* $0-length – Set the length of the track, the song pattern is always updated immediately after
* $0-loopPlayback – Enable or disable looping playback
* $0-soundFile0, $0-soundFile1 … - Specify the audio sample used by each instrument, soundFile0 is the Vocoder instrument. The remaining inputs are drum samples.

SoundSystem also outputs the following messages which the application should listen for:

* $0-notifyProgress – Provides notification of playback position within the track, value is the number of notes played.
* stopPlayback – Event fired after the playback reaches the end of the track if repeat is disabled
* recordDone – Event fired after completing recording to a wav file, we had to include a large pause before firing this to allow Pd some time to flush data and close the file as Pd offers no way to receive a notification when the file has been closed.

### PolyTracker

Polytracker is an abstraction which provides a mechanism for packing multiple instrument tracks into a single table. A table in Pd offers functionality similar to array lookups and can be supplied data from an array in the application. Upon receiving a position in the sequence of notes in the left inlet PolyTracker outputs the values of each note for each instrument. The right inlet to polytracker accepts messages “numInstruments” and “length” which control the size of the array and number of notes fired off for each input. PolyTracker provides a global array called “pattern” which should be provided a sequence of values. Sequences are stored in memory such that the sequence of notes for instrument 1 follows directly after the sequence of notes for instrument 0 and so on.

### Vocoder

Vocoder is a complex patch which is a modified version of the phase-vocoder example provided with Pd. The patch is responsible for playback of a soundfile with a modified pitch at run-time. The complexity of this object is in the implementation of time stretching, as the object allows for the length and pitch of an instrument sample to be modified individually. When running on the phone this object consumes a large amount of processing power, therefore we are limited to playing a single pitch modified instrument in our app. In future versions of the app if this could be optimised more instruments could be played concurrently which would allow for a baseline for example. One strategy for optimising this might be to re-implement performance critical objects in libpd using the audio dsp of the iPhone. The left inlet of the vocoder object receives a pitch modification value (measured in cents) and the right inlet receives the filename of the instrument sample.

### SimpleInstrument

SimpleInstrument handles playback of an audio file with modified volume. We use this for percussive instruments such as drums, but otherwise it behaves similarly to the vocoder object. SimpleInstrument requires significantly less processing power than the vocoder instrument and so should be used instead where possible. The left inlet receives the volume of playback, which should lie in the range of 0-1 and the right inlet receives the filename of the instrument sample.

# Objective-C Application

The user interface and rhythm generation algorithms are implemented in Objective-C. To compile the application we require XCode 4.3.1 and the external libraries included in the subversion repository. With the Objective-C project there are also a number of unit tests which cover the image property extraction calculations. The objective-c application is divided into the following objects.

### AppDelegate

The AppDelegate is the standard entry point to objective-c applications. We initialise the global objects for PdAudioController here.

### ViewController

The standard layout for a Cocoa-Touch UI application is to have a ViewController class that responds to user-interface interaction. Over time this class became a hub-class in our application which manages the interaction between the audio engine and the entire user interface. Additionally the algorithms for providing song patterns are implemented within ViewController. If the project is developed further refactoring the generation algorithms into their own classes should be considered. Other than a small number of application created user interface objects, all event handlers are connected to user interface elements in the form designer. ViewController also manages displaying and closing the “Saving in progress” screen, instrument selection screen and the SoundCloud upload screens.

### ImageProperties and ImageSlice

The image-properties object extracts various statistics for use in the song generation algorithm. Images passed in may be downsampled if they are determined to be too large. Statistics are generated for each vertical slice of the downsampled image along with averages of the entire image. We have included a number of unit tests to verify the calculations of a number of properties are accurate. As we were rapidly prototyping various song generation algorithms a number of image properties were added which may or may not be used in the application. Unfortunately we did not have time to develop comprehensive unit tests for all properties. See our test plan document for more details.

### ProgressScreen

The progress screen class handles creating and displaying an overlay screen with a progress bar. The position of the progress bar and title of the screen may be updated externally. We use this to display a progress bar for rendering the audio to file for uploading. Adding more descriptive text and images to this screen would provide a better user experience if there is time to develop this further.

### InstrumentSelector

The InstrumentSelector handles creating and displaying an overlay screen for selection of instruments and instrument packs. We use a picker-wheel to present a list of named instruments. For future development a better user experience could be achieved if the screen included a more detailed explanation and the option to preview an instrument.

Test Plan

We have implemented a number of unit tests which are run through XCode by holding the mouse down over the “Run” button and selecting “Test”. The test report generated will highlight any problems.

Our unit tests cover the ImageProperties and ImageSlice classes. As development progressed more properties were added to these objects some of which are no longer in use. The unit tests do not comprehensively cover all properties of these objects and if possible an effort to improve their scope should be made in the future. It is difficult to generate unit tests for a large portion of this project as the majority of the application is user interface code and can only be tested manually on the device. To be rigorous with testing this, we have maintained a log of bugs in this document. Periodically we have checked the application for re-occurrences of each bug

The remainder of the application is implemented in Pure Data. The majority of issues that can occur can only be tested manually as we are testing the sounds and sound quality. We have kept a record of the approaches we tried in mapping the image onto audio and a log of client feedback. If new approaches or ideas arise they should be documented.

Before committing to the repository, it is important to test the program still runs on the iPhone in Debug and Release mode, as there are a number of differences between the simulator and the device which may cause issues.

Bug Log

Date: 30/01/2012  
Bug: Pd patch runs as expected under Pd and Pd-extended but under libPd produces no output  
Test Strategy: We expected that the problem was likely caused by objects that we use in Pd but are not available under libPd. To test this we removed parts of the program until it works as expected on the iPhone.  
Resolution: Our suspicions were confirmed when we discovered that the expression objects, “expr” and “expr~” do not work under libPd. Unfortunately libPd does not output an error message when this error arises, so great care should be taken to avoid using it.

Date: 06/03/2012  
Bug: Application runs unusably slow during playback on the iPhone 4 hardware but full speed in the emulator (applied to Debug and Release builds). The phone gets very hot during operation so this is likely a performance issue.  
Test Strategy: Use apple’s “Instruments” utility to profile the cause of performance hit.  
Resolution: The vocoder instrument is very performance intensive. Multiple actions were taken to resolve the issue:  
 Reduce audio engine sample rate to 22050 from 44100Hz  
 Use only a single Vocoder instrument in the application  
 Developed a less performance intensive instrument for percussive instruments

Date: 08/03/2012  
Bug: Image properties appear to be using horizontal slices instead vertical slices for images taken with the iPhone camera in landscape orientation.  
Test Stratergy: The problem can be detected by printing the average properties of a slice to console when capturing an image with high variation along the horizontal axis with the phone in a horizontal orientation.  
Resolution: The cause of the problem was that the rather than rotating an image the iPhone camera attaches a “device orientation” attribute. The two possible approaches to solving this problem are rotating the image or re-writing the interpreter to scan horizontal slices. We opted for the former as it produces simpler code and can be merged with the affine image transformation we also use to downsample the image for extra run-time performance.

Date: 09/03/2012  
Bug: Recording never completes if repeat playback mode is enabled while recording  
Test Strategy: The bug can be repeated by enabled playback repeat mode and hitting record, the application should finish recording at roughly the length of the song.  
Resolution: To prevent this happening, safeguards were implemented to disable and re-enable playback repeating within the SoundSystem patch file.

Date: 19/03/2012  
Bug: Image ratio is incorrect when taking images in different orientations  
Test Stratergy: The problem can be reproduced by capturing an image with a portrait orientation  
Resolution: Enabling “auto sizing” in the XCode UI designer solves the problem with display however this required us to implement a method to calculate the actual width on display so that we could correctly position the playback progress bar below the image.

Date: 23/04/2012  
Bug: Application fails to launch with error “application could not be launched” in Release mode on the phone (doesn’t apply to simulator).  
Test Strategy: Web search for possible causes into this behaviour. Cross examination of build configuration differences between “debug” and “release” configurations.  
Resolution: The web search yielded some useful debugging tools for this scenario including accessing device logs through XCode, launching GDB debugging for access to the GDB console and enabling zombie objects for detection. Even with these techniques we found no possible leads however in investigating an unrelated warning “Unexpected size” at link time, we found that disabling link time optimisation fixed the issue. This option was enabled by default when upgrading to XCode 4.3.1.

External Libraries

Our application includes external libraries obtained from the libpd’s and soundcloud-api’s project websites. The copies in our subversion repository were obtained from source distribution. The libpd libraries on the subversion server were last updated on 12 March 2012. LibPd may currently be obtained from <http://gitorious.org/pdlib> (as of 12 March 2012). Be aware that LibPd’s repository was relocated from github to gitorious halfway through the project.

Soundcloud’s libraries and it’s dependencies are:

* CocoaSoundCloudAPI
* CocoaSoundCloudUI
* JSONKit
* OAuth2Client
* OHAttributedLabel

The copies in the subversion were last updated on the 23rd April 2012 and were obtained through the instructions found on <http://developers.soundcloud.com/>. Detailed instructions on how their API should be integrated with an iPhone application may also be found at the same URL.