Audio Fuego

A comparison of live audio handling across multiple languages

Gordon, Ben bcgordon@ucsc.edu
Nussbaum, Joshua jsnussba@ucsc.edu
Rodriguez, Octavio orodrig2@ucsc.edu
Kayed, Rashad rkayed@ucsc.edu

March 11th, 2015

Overview

Over the past two decades, computers have had a massive effect on art and the creative process in general; they have revolutionized the way in which nearly all forms of art are created, distributed, sold, and consumed. In particular, electronic music has reached a critical mass of popularity in the United States over the past three years - though it has seen mainstream popularity in Europe for the better part of the past decade. This has created a booming industry (and an even larger software piracy community) centered around music production software.

Today, the overwhelming majority of audio software tools are programmed in low-level systems code, as signal processing is computationally intensive and efficiency is a must (especially in a live performance setting). Usually, this means C/C++, as these have been the dominant systems languages for decades; they are well-tested, have robust libraries, and massive communities of people working to improve them. However, this singular focus on using systems programming to write audio software has left an opening for audio software written in different languages to fill a somewhat different niche. The web audio workstation AudioTool (www.audiotool.com) is a premier example of this; it allows users to create and publish music using a wide array of synthesizers, drum machines, and effects, all from the browser, and all without leaving their site.

Project Outline:

Our goal was to implement the most basic of music creation tools - a drum sequencer - across multiple languages, to better understand what audio handling entails and which languages were well suited towards it.

To begin with, we set out to define a set of features which would be included in the project in each language. Other features could be added, depending on time constraints and ease of implementation, but these features would have to be included in every different version. This feature set is quite small:

- ❏ One measure, sixteen quarter-notes, in 4/4 time
- ❏ Four different audio samples - kick, snare, clap, and hi-hat
- ❏ Adjustable tempo
- ❏ Ability to pause and reset all notes

We decided that the visual aspect would be a low priority; though a flashy GUI is certainly nice, it tells us nothing about a language's abilities in regards to audio handling.

Languages:

We initially set out to implement Audio Fuego in Haskell and Python. We chose Haskell because we thought that writing music software in a functional language was unprecedented, and thus worthy of exploration; Python was chosen because it is a distinctly high level language, and therefore a good contrast with the C/C++ based audio software currently dominating the market. However, in both of these languages we encountered the same problem: there were no currently maintained libraries which could handle live audio sampling. There was a Haskell library for audio synthesis, but none suited for live audio playback; since a drum sequencer does no actual synthesis, but simply plays back samples at a given frequency, this did not meet our needs. Though Python had a library suited towards audio playback, it has been unmaintained for six years, is poorly documented, and is quite complicated to use.

In the end, we abandoned the idea of using Haskell and Python, and resorted to languages which we knew would have libraries we could count on to handle the nitty-gritty details of audio I/O - Objective-C and Javascript.

Javascript

Software Design:

To abstract away the complexities of using HTML5's web audio API, we incorporated a small open-source library called Buzz, which made the loading and playing of audio files a breeze. Since we were using Javascript, it was an obvious choice to include JQuery in our code as well, to simplify DOM interactions.

The flow of the program is quite simple. Given a BPM, the program calculates the time in milliseconds between beats, and sets the *run()* function to be executed at that interval. A global variable is used to keep track of which note the sequencer should play next; at each execution of *run()*, the function checks, for each sound, whether the note corresponding to that global variable has been selected. If it has, it uses the Buzz library to play the sound; at the end of the function's execution, it increments the global variable. Whenever a user selects a new note to be played, it executes an event-driven callback to a function, updating an internal data structure that represents which notes are currently selected.

Visual Design:

Since Javascript is thoroughly intertwined with HTML and CSS, building a GUI was extremely easy, and Twitter's Bootstrap framework made it a breeze to clean up the interface and make everything look pretty. Google's material design principles also played a role in the design of the GUI; some snippets of CSS/LESS code from a page on www.codepen.io was incorporated into the code base. Links to Google's page on material design, as well as the codepen page from which CSS was used, are included at the end of this paper.

Issues Encountered:

Unlike Objective-C, Javascript is single-threaded, which presents some issues for implementing a beat sequencer. Timing is very important when it comes to audio software; since the UI updates and the audio I/O are mutually exclusive, there is occasionally some delay between the two. This manifests itself as noticeable, though minor, inconsistencies in the timing of the beats.

There was another issue that was discovered early in the development process: the handling of different audio file types is inconsistent across browsers. In particular, Firefox was unable to play WAV files, which is almost universally used for drum samples. Interestingly, WAV is listed by Mozilla as one of the audio file formats supported by Firefox; nevertheless, switching development to Chromium immediately resolved the problem. Therefore, Chromium/Chrome is recommended as the browser to use when playing with the Javascript version of Audio Fuego.

Objective-C

Software Design:

Implementing an audio sequencer from the start seemed like it would be easier in Objective-C because the language can take advantage of multithreading. However, it turns out that all I/O events must be executed on the main thread. So once a sound was encountered, the application would asynchronously dispatch the sound to play after the UI updates on the main thread. So we couldn't fully take advantage of multithreading, but were still able asynchronously dispatch sounds to be played on the main thread right after the sound was encountered.

```
        NSArray *tickingVoices = [_groove.voices filteredArrayUsingPredicate:predicate];

         NSUInteger blockTick = _currentTick;

         dispatch_async(mainQueue, ^{ //Check for sound and if it's encountered play it after the UI update

             [_delegate groover:self didTick:blockTick];

                                                    if (_didTickBlock) {
                                                                _didTickBlock(blockTick);
                                                    }

             if (tickingVoices.count > 0) {
                 [_delegate groover:self voicesDidTick:tickingVoices];

                                                                if (_voicesDidTickBlock) {
                                                                            _voicesDidTickBlock(tickingVoices);
                                                                }
             }

         });
```

The flow of the program is very similar to the Javascript with the main loop of the sequencer computing the interval between the sounds based on a given bpm, then calling the run function again at that interval to check if any notes in that incremented index have been selected. When a new rectangle is checked the internal data structure is updated to reflect that.

Visual Design:

It was hard to make our apps look the same as there is no bootstrap equivalent to designing native iOS apps. CGContextRef is a built in framework for drawing 2d shapes in objective-c. Here is a snippet of the code used to draw the body of the sequencer

```
   for (int i = 0; i < rows; i++) {

       NSUInteger columns = [_delegate gridView:self columnsForRow:i];

       for (int j = 0; j < columns; j++) {
           float height = rect.size.height / rows;
           float width  = rect.size.width  / columns;
           float x      = j * width;
           float y      = i * height;
```

```
            if ([_delegate gridView:self isSelectedAtRow:i column:j]) {
                CGContextSetRGBFillColor(cxt, 1.0, 0.1, 0.1, 1);
                CGContextSetRGBStrokeColor(cxt, 1, 1, 1, 1);
            } else {
                CGContextSetRGBFillColor(cxt, 1, 1, 1, 1);
                CGContextSetRGBStrokeColor(cxt, 0, 0, 0, 1);
            }

            CGContextStrokeRect(cxt, CGRectMake(x, y, width, height));
            CGContextFillRect(cxt, CGRectMake(x, y, width, height));
        }
    }
```

This function is called every time one updates the display by touching the rectangles to add or take away sound. To determine where the user is touching, the UIResponder class touchesBegan: withEvent: was used to get the point that the user touched on the screen and then from that we can determine the rectangle by taking the x and y points and dividing them by the (width/height divided by the number of columns/rows) respectively and rounding that number up to the nearest integer. We then determine based on the row and column number what rectangle we selected and mark it selected in a global array. We then redraw the whole body and color the selected rectangle red.

```
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *touch = [touches anyObject];

    CGPoint point = [touch locationInView:self];

    NSUInteger row = (NSUInteger)(point.y / (self.bounds.size.height / [_delegate rowsForGridView:self]));
    NSUInteger column = (NSUInteger)(point.x / (self.bounds.size.width / [_delegate gridView:self
columnsForRow:row]));

    [_delegate gridView:self wasSelectedAtRow:row column:column];

}
```

Objective-C is a compiled language, not an interpreted language like Javascript, so testing the program was a bit more of a process. To test the various screen sizes I would have to recompile the project for a different iOS device in the simulator, however with Javascript one could simply resize the window to test every possible case.

The application runs at around 25mb and uses 100% of the CPU when looping through sounds because of the continuous loop on the main thread.

Google’s Material Design principles:
http://www.google.com/design/spec/material-design/introduction.html

Material Design codepen:
http://codepen.io/zavoloklom/pen/Gubja