MoMu: A Mobile Music Toolkit

Nicholas J. Bryan, Jorge Herrera, Jieun Oh, Ge Wang
Center for Computer Research in Music and Acoustics (CCRMA)
Stanford University
660 Lomita Drive
Stanford, CA, USA
{njb, jorgeh, jieun5, ge}@ccrma.stanford.edu

ABSTRACT

The Mobile Music (MoMu) toolkit is a new open-source software development toolkit focusing on musical interaction design for mobile phones. The toolkit, currently implemented for iPhone OS, emphasizes usability and rapid prototyping with the end goal of aiding developers in creating real-time interactive audio applications. Simple and unified access to onboard sensors along with utilities for common tasks found in mobile music development are provided. The toolkit has been deployed and evaluated in the Stanford Mobile Phone Orchestra (MoPhO) and serves as the primary software platform in a new course exploring mobile music.

Keywords

instrument design, iPhone, mobile music, software development, toolkit

1. INTRODUCTION

Motivated by the newly blossoming field of mobile music [4, 3, 13, 7, 2, 16, 15], the Mobile Music (MoMu) toolkit offers a collection of application programming interfaces (API) and utilities focusing on mobile music development and design. The initial MoMu release focuses on usability and rapid prototyping for the iPhone OS with a particular emphasis toward unifying audio input/output, synthesis, and graphics with the onboard sensors now available on commodity mobile phones including accelerometer, compass, location, and multi-touch as seen in Fig. 1. More specifically, the fundamental design goals of MoMu include:

- Real-time audio, synthesis, and control
- Consistent conventions for external sensor access
- Unified common functionality for mobile music
- Focus on ease of use, setup, and installation
- Open source C, C++, and Objective-C code

The design focus enables programmers with little or no prior mobile development experience to rapidly develop interactive audio applications, while concentrating on musical and aesthetic considerations. MoMu builds upon the iPhone OS SDK as well as several open source software packages including the Synthesis ToolKit (STK) for sound synthesis and processing [11, 12], OSCpack [1] for networking via Open Sounds Control [17], and a Fast Fourier Transform (FFT) implementation adapted from the CARL software distribution [9]. To maximize performance on current mobile hardware, MoMu has been implemented largely in a low-level language (C/C++). The open-source nature allows for custom modifications or additions for production level applications. As far as our experience has shown, such an approach tends to be more familiar to computer musicians and audio developers alike, easier to learn, and lends itself to greater code reuse for future platforms. To encourage academic researchers and commercial developers to focus on more musical and interactive applications, MoMu is released under a BSD-like license. In the remaining paper, we discuss the components of MoMu and evaluate its use in a mobile phone orchestra [15] and classroom setting.

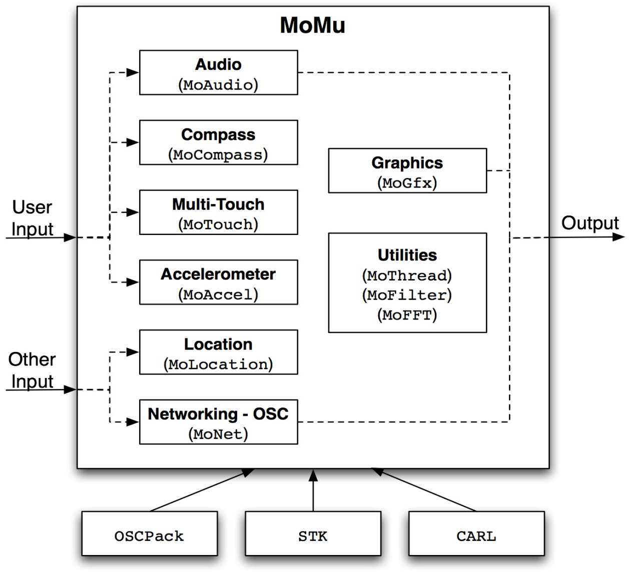

Figure 1: MoMu Overview.

2. API

The design of MoMu can be divided into two general topics involving mobile music development: access to onboard sensors and other useful programming abstractions. The incredibly diverse onboard sensors now available on current smartphones include audio input/output, accelerometer, compass, location, networking, and multi-touch. For our purposes of mobile music, such sensors have a transformative effect on the gamut of new musical experiences.

As a result, we have focused our design efforts on specific APIs for providing easy access to these elements (as found in §2.1.1 and §2.2). The more general category of other useful programming tools including audio synthesis, analysis, and processing, common graphics functionality, and general utilities such as threading, filtering, and fast Fourier transform (FFT) functionality.

Onboard sensor functionality encourages a unified design when considering the goals outlined in §1. MoMu treats each sensor as a "singleton" resource server—each individually managing its respective data that can be dispatched to the clients of the given API. Because of this, MoMu builds upon the iPhone SDK, providing an added layer of abstraction with the hope of simplifying setup and usage. Such a design is not limited to iPhone OS and should lend itself to similar implementations for alternative platforms such as Android or webOS platforms.

MoMu provides global access to necessary sensor data through a number of static C++ classes. No objects need to be instantiated or maintained. For a given resource, the necessary data is accessible either via polling, interrupt-driven event handling, or both. Event handling is achieved via the registration of client callbacks, typically requiring a single line setup. More specifically, a client of the resource implements a callback method of a predefined prototype, registers with the resource, and instantly begins processing. Additionally, all resources allow users to pass data through the callback structure (via a `void *` function parameter) and register multiple client callbacks per API (audio is limited to one callback).

2.1 Audio

At the foundation of MoMu is a powerful and simple-to-use API (`MoAudio`) for real-time audio input/output (I/O), routing, and computing. The API abstracts the complicated audio setup, while exposing the core interface for audio I/O needed for full-duplex real-time audio. The goal is to significantly reduce the start-up cost for experimenting and developing high-performance digital audio applications. Additional functionality is added with a port of the Synthesis Toolkit (STK) for the iPhone OS.

2.1.1 *I/O and Routing*

A low-latency, full-duplex solution for audio I/O is fundamental to interactive audio processes. The iPhone SDK provides access to the audio subsystem in a variety of manners, each of which requires significant setup. MoMu's `MoAudio` API encapsulates the most advanced audio I/O access on the iPhone (using the Remote I/O AudioUnit). This approach requires sophisticated initialization, but is also the only approach to support full-duplex audio I/O, and generally has the most favorable audio latency. As a concrete measure of `MoAudio`'s usefulness, the code to set up audio processing and routing encapsulates over seven hundred lines of iPhone SDK code as two function calls exposing only the key parameters (e.g., sample rate, buffer size, and number of channels).

`MoAudio` is modeled loosely after RtAudio [6, 5], and provides the mechanism to register a callback that handles both audio input (from the onboard or headset mic) as well as stereo audio output. The callback function is to be prototyped as follows:

```
void <AudioCallback>
(Float32 * buffer, Uint32 numFrames, void * userData)
```

As in earlier versions of RtAudio, `buffer` is processed inline–initially containing input audio samples (from the A-to-D converter) requiring the user to overwrite the buffer with audio output (for the D-to-A converter/speakers). This leaves maximal flexibility for the programmer to "wire in" any audio analysis and synthesis algorithms, using the callback as the interface to the audio subsystem. Note the buffer contains samples in 32-bit floating-point values (internally, `MoAudio` converts fixed-point to floating point before entering the callback, and converts back to fixed point to render the audio).

Another task of `MoAudio` is to set up and handle audio routing. On devices such as the iPhone, there is a fairly complex routing system that must reconcile the onboard speaker and microphone against possible outboard headsets (with or without a microphone). These routes can change dynamically, requiring different responses and changes to the behavior of the program. `MoAudio` currently handles these route changes internally in an isolated method. While this is not yet exposed in MoMu, it is possible to modify the source of `MoAudio` directly and achieve any desired routing behavior.

2.1.2 *Sound Synthesis*

As a foundation for audio signal processing and synthesis, an iPhone OS port of the Synthesis ToolKit (STK) is included with the initial release of MoMu. This is not the first instance of STK on mobile devices—MobileSTK previously ported STK to the Symbian OS [4]. STK offers a collection of signal processing and algorithmic synthesis tools designed for rapid development and ease of use, complimenting the similar design goals of this work. While a vast majority of the STK has been ported, some less commonly used classes (for mobile music) have been left unsupported including socket-related functionality and RTAudio/RTMidi functionality. Because of the extremely common use, audio file loading and playback as supported by STK also has been added in the form of `MoAudioFileIn` in a single file light-weight adaptation.

2.2 Sensors

In addition to full-duplex audio input and output, typical smartphones offer a collection of additional sensors. The iPhone 3GS includes a three-axis accelerometer, compass, networking, and location tracking technology. Accelerometer, compass, and location sensing all offer unique control that can be creatively leveraged in conjunction with networking technology. The following sections will address the individual access of such sensors via MoMu, all following the design outlined in §2.

2.2.1 *Accelerometer*

The three-axis accelerometer is accessible via polling or callback structure in the `MoAccel` class and built upon the iPhone SDK's `UIAccelerometerDelegate` protocol and `UIAccelerometer` class. Using `MoAccel`, the accelerometer is typically accessed through the following callback prototype:

```
void <AccelCallback>
(double x, double y, double z, void * userData).
```

Once inside the registered callback, any further accelerometer processing such as smoothing or tilt processing can be performed and connected to outside data via the `void *` parameter. A method for sample rate control is also provided.

2.2.2 *Location*

Location sensing, either in the form of Global Positioning

Systems (GPS) or cellular/wifi network-based techniques, can give accurate location data such as latitude, longitude, altitude, and speed. `MoLocation` encapsulates the iPhone OS CoreLocation Framework and conveys data back to API clients via a callback:

```
void <LocationCallback>
(CLLocation * newLoc, CLLocation * oldLoc,
void * userData).
```

As with compass sensing, location-based mobile music is relatively unexplored, showing great promise for locative music and media interaction where musical interaction is parametrized by the location or heading of a given user across the globe.

2.2.3 Compass

Starting with the 3GS model (June 2009), iPhone hardware provides access to one of the first instances of a digital compass on commodity mobile phones. Layered on top of the Core Location Framework, `MoCompass` provides quick access to heading direction (true and magnetic), time stamp, and heading accuracy. Clients of `MoCompass` can poll compass values or register a callback with the following prototype:

```
void <CompassCallback>
(CLHeading * heading, void * userData)
```

The novelty of compass sensing for mobile music has shown promise and offers an exciting new method of control, further enhancing the connection between mobile devices and the physical world.

2.2.4 Multi-touch

As the iPhone offers a multi-touch interface that detects up to five points of contact, the `MoTouch` class has been written to provide easy access to the touch information. Clients of `MoTouch` implement their multi-touch callback using the following syntax:

```
void <TouchCallback>
(NSSet * touchSet, UIView * view,
const std::vector<UITouch*> & touchVec,
void * userData)
```

Users can access the multi-touch data using either an unordered `NSSet * touchSet` or time-ordered `std::vector<UITouch*> & touchVec`. The former set is natively provided (via subclassing `UIResponder`), providing location and attributes of touches, as well as instance methods provided by the `UITouch` class. In contrast, the latter (C++ Standard Template Library `std::vector`) adds a time-ordered structure to the touch information, giving users a consistent way of tracking touches over time. `MoTouch` currently leverages the Objective-C runtime system to dynamically subclass any `UIView` within a running application, simplifying multi-touch setup, and unifying syntax under the event-based callback paradigm [8].

2.3 Networking

As established in the music technology community, Open Sound Control (OSC) has greatly simplified communication with ubiquitous implementations across multiple languages [17]. MoMu's `MoNet` API builds upon OSCpack and handles all OSC related network activity in a simplified manner [1]. More specifically, `MoNet` wraps the method calls required to send and receive OSC messages into a single method call. This limits the possibilities offered by OSCpack, but provides a fast and simple way of dealing with common tasks. For more complicated or specific tasks, the provided and slightly modified version of OSCpack can be used as a stand-alone API. A common scenario of sending a simple three parameter message to IP address `192.168.0.1` over port `8888` can be performed as follows:

```
const int n = 3;
char types[n] = {'i','f','s'};
MoNet::sendMessage( "192.168.0.1", 8888,
"/message", types, n, 5, 3.14, "some text" );
```

The variables `n` and `types` define the number and types of arguments to be sent, respectively.

Similar to onboard sensor access, handling incoming OSC messages is done by callback registration. `MoNet` is capable of handling different incoming messages simultaneously via multiple callbacks, therefore requiring the user to associate each message to a callback. For example:

```
MoNet::addAddressCallback( "/messageA",
messageACallback, userData);
MoNet::addAddressCallback( "/messageB",
messageBCallback, NULL);
```

In this example, `"/messageA"` and `"/messageB"` will trigger `messageACallback` and `messageBCallback` respectively. In this case, `messageACallback` receives `userData`, while `messageBCallback` receives `NULL`. It is important to mention that each callback runs on a different thread and not on the main thread of the application. This allows continuous listening to incoming messages, but imposes slight complications when managing Objective-C data structures. In particular, additional considerations in terms of memory management and user interface element manipulation may be required.

2.4 Graphics

The iPhone hardware (and those of many modern mobile devices) has a dedicated graphics processing unit (GPU) for accelerated 3D graphics and operations. In terms of software, iPhone OS complements the added graphics hardware and provides support for OpenGL ES–a subset of the OpenGL standard for powerful 3D and 2D graphics capabilities. From our experience, utilizing such functionality for mobile phone instrument development encourages a more interactive, responsive, and immersive experience [10, 14].

The OpenGL ES API, however, does not provide many of the higher-level OpenGL functions and does not support the GLU (GL Utility) API. To address these missing functionalities, MoMu's `MoGfx` provides a set of helper functions that implements the traditionally useful utility functions not found in OpenGL ES. They include projection matrix construction (both perspective and orthographic), camera control, loading of 2D images for texture mapping, as well as a "homebrew" 3D vector implementation for physical simulation.

2.5 Utilities

In addition to the core functionality for access to audio, sensors, networking, and graphics, MoMu also provides a number of useful utilities collected from our past work in computer music development.

2.5.1 Threading

The iPhone OS supports preemptive concurrency via kernel threads, identical in use to those found on desktop operating systems. MoMu's `MoThread` provides a familiar C/C++

abstraction for easily creating and spawning new concurrent processes. The abstraction is a thin layer above the `pthread` interface, and it further simplifies thread set up and creation.

2.5.2 *Filtering*

MoMu's `MoFilter` provides a number of commonly used basic filter elements not limited to first and second-order filters (e.g., OnePole, PoleZero, BiQuad) that can be used both for audio, accelerometer, video, or other signal processing. Common tasks such as slewing audio control parameters or smoothing object motion in graphics are ideal examples of where `MoFilter` might be useful.

2.5.3 *FFT*

Based on UCSD's CARL distribution of music software, `MoFFT` adds simple C-based Fast Fourier Transform functionality and commonly used window generators to MoMu. It should be noted that the function prototype parameters of `rfft`, previously defined in CARL, are slightly modified for ease of use.

3. EVALUATION

MoMu was initially developed to facilitate the design, prototyping, and implementation of instruments for the Stanford Mobile Phone Orchestra. This has provided a closed-loop design and evaluation cycle in a practical setting, allowing us to identify and iterate key design issues. For further evaluation, an initial version of MoMu was released as part of an introductory course on mobile music development and design. With the aid of MoMu, students with no prior iPhone programming experience were able to develop applications utilizing full-duplex audio, accelerometer control, multi-touch, and user interface elements all within a single week. Furthermore, students with past iPhone programming experience have given positive feedback regarding the benefits of the API. In addition to an initial classroom release, the authors have collectively worked with the toolkit over several months, developing numerous musical instrument applications for a mobile phone concert as further discussed in [10].

4. CONCLUSIONS

The MoMu toolkit greatly simplifies mobile music application development by unifying onboard sensor access and providing a collection of other useful tools and utilities common in mobile music development. By providing an additional layer of abstraction on top of the iPhone SDK frameworks, developers can quickly integrate full-duplex audio, accelerometer, compass, location, networking, and multi-touch interaction under a common design. Each such sensor can be accessed either by polling or interrupt based event handling and can be initialized with a single line of code. In addition to simplifying onboard sensor management, MoMu offers a collection of other useful utilities used for audio synthesis and processing, graphics, threading, and general purpose filtering. As mentioned before, in the hope that the toolkit provides useful functionality for mobile phone music application development, MoMu has been released under a BSD-like license. For more information on retrieving and installing the MoMu toolkit, please see `http://momu.stanford.edu`.

5. ACKNOWLEDGMENTS

The development of MoMu has been made possible by a National Science Foundation Creative IT grant No. IIS-0855758. We would like to thank Spencer Salazar of Smule for being an essential author to the development of `MoAudio` and `MoLog` as well as early adopters of MoMu including Luke Dahl, Rob Hamilton, and Turner Kirk. Additionally, we would like to thank the winter 2010 Mobile Music class at Stanford University. Lastly, we would like to thank the multiple authors of the prior work incorporated into MoMu.

6. REFERENCES

[1] R. Bencina. oscpack, Nov. 2006. `http://www.audiomulch.com/~rossb/code/oscpack/`.

[2] W. Carter and L. S. Liu. Location33: A mobile musical. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 176–179, May 2005.

[3] G. Essl, G. Wang., and M. Rohs. Developments and Challenges turning Mobile Phones into Generic Music Performance Platforms. In *Proceedings of the Mobile Music Workshop*, Vienna, 2008.

[4] G. Essl and M. Rohs. Mobile STK for Symbian OS. In *Proceedings of the International Computer Music Conference*, pages 278–281, New Orleans, 2006.

[5] G. P. Scavone. RtAudio: A Cross-Platform C++ Class for Realtime Audio Input/Output. In *Proceedings of the International Computer Music Conference*, pages 196–199, Gotborg, Sweden, 2002. ICMA.

[6] G. P. Scavone and P. R. Cook. RTMidi, RTAudio, and a Synthesis ToolKit (STK) Update. In *Proceedings of the International Computer Music Conference*, Barcelona, Spain, 2005.

[7] G. Levin. Dialtones - a telesymphony. `www.flong.com/telesymphony`, Sept. 2, 2001. Retrieved on April 1, 2007.

[8] M. Ali. *iPhone SDK 3 Programming: Advanced Mobile Development for Apple iPhone and iPod touch.* John Wiley and Sons, New Jersey, 2009.

[9] F. R. Moore, G. Loy, M. Dolson, R. Wright, and et al. Carl software, 1980. `http://crca.ucsd.edu/cmusic/`.

[10] J. Oh, J. Herrera, N. J. Bryan, L. Dahl, and G. Wang. Evolving mobile phone orchestra. In *Proceedings of the International Conference on New Instruments for Musical Expression*, June 2010.

[11] P. R. Cook. Synthesis ToolKit in C++, Version 1.0. In *SIGGRAPH 1996, Course #17 & 18, Creating and Manipulating Sound to Enhance Computer Graphics.*, volume Available from ACM SIGGRAPH, 1996.

[12] P. R. Cook and G. P. Scavone. The Synthesis ToolKit (STK). In *Proceedings of the International Computer Music Conference*, Beijing, China, 1999.

[13] A. Tanaka. Mobile Music Making. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 154–156, June 2004.

[14] G. Wang. Designing smule's iphone ocarina. In *In Proceedings of the International Conference on New Interfaces for Musical Expression*, Pittsburgh, 2009.

[15] G. Wang, G. Essl, and H. Penttinen. Do mobile phones dream of electric orchestras? In *In Proceedings of the International Computer Music Conference*, Belfast, 2008.

[16] D. Waters. Mobile Music Challenges 'iPod Age'. BBC News Online, Monday, 7 March 2005. Available online at `news.bbc.co.uk/1/hi/technology/4315481.stm`.

[17] M. Wright. Open sound control 1.0 specification. 2002.