JunctionBox for Android: An Interaction Toolkit for Android-based Mobile Devices

Lawrence FYFE
InnoVis Group
University of Calgary
2500 University Drive NW
Calgary, AB T2N 1N4
Canada
ljfyfe@ucalgary.ca

Adam TINDALE
Alberta College of
Art + Design
1407 14 Avenue NW
Calgary, AB T2N 4R3
Canada
adam.tindale@acad.ca

Sheelagh CARPENDALE
InnoVis Group
University of Calgary
2500 University Drive NW
Calgary, AB T2N 1N4
Canada
sheelagh@ucalgary.ca

Abstract

JunctionBox is an interaction toolkit specifically designed for building multi-touch sound control interfaces. The toolkit allows developers to build interfaces for Android mobile devices, including phones and tablets. Those devices can then be used to remotely control any sound engine via OSC messaging. While the toolkit makes many aspects of interface development easy, the toolkit is designed to offer considerable power to developers looking to build novel interfaces.

Keywords

Android, OSC, mobile, interaction, toolkit.

1 Introduction

The Android operating system [Google, 2012a], developed by Google Inc., runs on a wide variety of mobile devices, including phones and tablets. With the easy availability of these devices, it becomes desirable to develop sound and music applications for them.

One application scenario involves using interfaces to control sound in which the interface is one computer and the sound generator is another. The advantage of this scenario is that the quality of the audio on mobile devices will likely not be as good as with combinations of audio interface and speakers in which both are designed to provide the highest quality audio experience. Another related advantage is the possibility of multi-channel scenarios that go beyond the two channels available on portable devices.

For example, an Android device would handle input by a performer, with the option of projecting the interface, while another computer handles all sound processing. This is not a hypothetical scenario but is the current musical practice of the first author. With this kind of scenario, certain aspects of development, like messaging between computers, can be made easier by a toolkit that puts commonly used or repetitively coded features into a single place for reuse.

JunctionBox is an interaction toolkit for building sound control interfaces that addresses this scenario. The toolkit is written in Java and is designed as a library for the Processing development environment [Reas and Fry, 2006]. JunctionBox bridges the interaction gap between the visual design possibilities offered by Processing and the sound control possibilities afforded by the Open Sound Control (OSC) protocol [Wright, 2005]. Figure 1 shows how JunctionBox relates these two functions. It does this by handling touch interactions and by making it easy for developers to map those interactions to OSC messages (via the JavaOSC library [Ramakrishnan, 2003]). JunctionBox has mapping features that are described in detail in an earlier paper by the current authors ([Fyfe et al., 2011]).

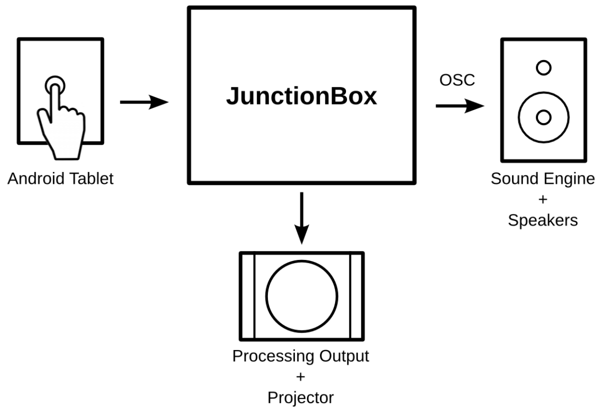

Figure 1: How JunctionBox serves as a hub for both sound and visuals.

2 Background

Other toolkits exist that have features similar to that of JunctionBox. The MT4J toolkit [Laufs et al., 2010] has many features for multi-touch interaction building and is available for use with Android. However, MT4J, lacks the OSC mapping capabilities that make JunctionBox a toolkit for building sound and music control applications. TouchOSC [hexler.net, 2012] offers functionality that is similar to what is offered by JunctionBox in which widgets can be mapped to custom OSC messages. Widgets include faders, push buttons and rotary controls and others that are are skeuomorphs of controls for physical hardware devices like mixers.

Instead of simply offering widgets, JunctionBox is a full-fledged development toolkit that is meant to be used by developers who are not necessarily interested in skeuomorphs. The JunctionBox approach to interaction design can be summed up as BYOW, for build your own widgets. The focus of JunctionBox is to provide functionality like OSC message mapping while offering the full range of visual design options available via Processing. This approach to toolkit design is meant to encourage greater creativity in the design of both interactions and interfaces.

3 JunctionBox for Android

The original version of the JunctionBox toolkit was not developed for Android. However, since it was written in Java, porting to Android did not require a complete rewrite of the existing code. The main difference between the standard Java version and the Android version of JunctionBox is that TUIO [Kaltenbrunner et al., 2005] is not needed on the Android version since Android devices have a separate system for handling touch interactions. Android-based systems get touch information via handling the data contained in the MotionEvent class [Google, 2012b].

Both versions of JunctionBox contain a Dispatcher class that is responsible for handling touch interactions that ultimately get mapped to OSC messages for controlling audio. Figure 2 shows the relationship of the MotionEvent class to JunctionBox classes including the Dispatcher. The mapping process is exactly the same for both the standard version of JunctionBox and the Android version.

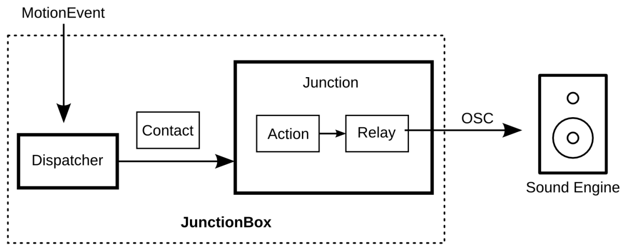

Figure 2: JunctionBox/Android internals with classes shown as boxes with solid lines.

Differences in using the Dispatcher in code have been minimized with both versions using the same code for construction. A new Dispatcher will take four arguments: the width and height of the screen/touch interface and a socket destination for OSC messages. The following code constructs a new Dispatcher object:

```
Dispatcher dispatcher = new Dispatcher(
    screenWidth,
    screenHeight,
    "192.168.1.1",
    7000);
```

All code written for the standard version of JunctionBox will work with Android with the addition of a single method made available in the Android Dispatcher class. The additional method, handleMotionEvent(), gets touch information from the MotionEvent class. The Dispatcher's handleMotionEvent() method is utilized by creating a dispatchTouchEvent() method in the code for the Processing sketch:

```
boolean dispatchTouchEvent(MotionEvent ev) {
    dispatcher.handleMotionEvent(ev);
    return true;
}
```

While only a single additional method is required for the Android version of JunctionBox, internally an important difference in touch location handling is that TUIO touch tracking data is normalized relative to the size of the interface while for Android devices, touch position is absolute. This difference is made transparent to the developer and the same code will work in both systems with only one minor change, the addition of the new handleMotionEvent() method. Making differences like this transparent is part of the JunctionBox approach of making development easier without taking away the ability to build highly customized interfaces.

4 Interfaces

The first author of this paper wrote the code for JunctionBox not only to provide the community with a toolkit for creating networked instruments but for his own performance practice. That practice currently consists of the building of touch interfaces for Android devices that run the interface to handle touch input. That interface then communicates with another computer running audio via OSC. The audio computer used to develop the following interfaces runs Ubuntu Linux, JACK and Pd.

4.1 Orrerator

The Orrerator interface has widget-like controls but with a much more stylized appearance. It is designed both as a general instrument that could be used for a variety of pieces but also specifically for the composition by the first author entitled *Sol Aur*.

The metaphor for the Orrerator is the Orrery or a model of the solar system, as shown in Figure 3. The sound engine controlled by the Orrerator is written in Pd and features four FM oscillators. The interface has four planet buttons that light up when toggled by simply touching the particular planet. Each planet button turns a single FM oscillator on or off.

In addition to on or off controls, each planet button can be rotated around its orbit by touching the orbit area and rotating it around the center. Orbit areas look increasingly lighter towards the inner planet. This change of rotation will detune that particular FM oscillator from its base frequency by a factor of between one and two with the starting vertical position being one and a full rotation back to that same position being two. Orbital rotations are limited to 360 degrees from the starting position.

On the left and right edges of the interface are two sliders: the left slider changes the index of modulation and the modulation frequency and the right slider changes the gain of all four oscillators.

4.2 Distance2

The Distance2 interface is designed specifically for a composition entitled *Distance 2 (Toshi Ichiyanagi)* by the first author. It features ten tiles numbered 1-5 with half of the tiles being

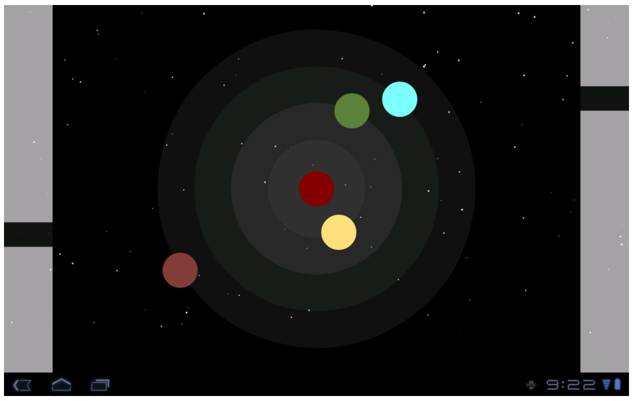

Figure 3: The Orrerator interface with the first and third planets active and playing.

white and half black. The black tiles feature reversed numbers 1-5. Each of the tiles can be moved anywhere in the interface with a single touch. When the tiles are moved into the target in the center of the interface and the touch is released, a sound file corresponding to that tile number is played. Reversed tiles play the same sound reversed. Figure 4 shows the tiles and the target. Sound files play until either they have reached the end of the file or if a touch is applied while playing. This allows the sounds to be paused and moved across the target.

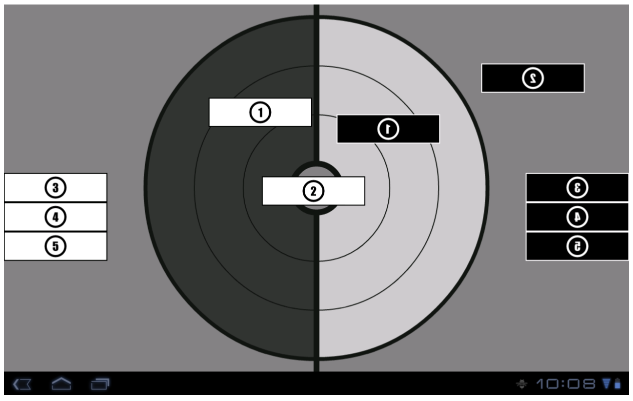

Figure 4: The Distance 2 interface with tile 2 in the center set to play at normal rate.

Tiles placed in the very center of the target are played back at normal playback rate. Tiles in the next circle out play at half that rate. The next circle plays at one-third rate and the outermost circle plays at one-quarter rate. If a tile is paused by touch and moved from one target circle to another, the sound will pause and resume at the new playback rate.

Tiles placed in the circle but on the left of the vertical dividing line play in the left channel while tiles on the right side play in the right channel. Tiles placed in the center (normal rate) play in the center of the stereo field.

The sound engine for this piece is written in Pd with a custom file playback control mechanism designed by the first author. The 5 sound files are all various recorded clips from an interview with experimental composer Toshi Ichiyanagi. *Distance 2*, the piece, is based on a piece by Ichiyanagi called *Distance* [Ichiyanagi, 1961] in which the performer must be at least three meters away from his or her instrument while playing. In a time of computer music in which wireless networking makes this kind of setup trivial, the notion of distance can be explored in other ways. In the context of the piece, distance from the center of the target represents the recognizbility of the recorded clips with the clips getting further away from the original as the tiles get further away from the center.

5 Summary

Bringing the JunctionBox toolkit to the Android operating system allows developers to build sound and music control interfaces on an increasing array of touch devices. With a focus on interface coding rather than providing pre-built widgets, JunctionBox provides developers with an opportunity to create new touch-based interactions with highly customized visuals. Two example interfaces, the Orrerator and Distance2, show some of the creative interfaces that can be built using the toolkit.

6 Acknowledgements

We would like to thank the Alberta College of Art + Design, the Canada Council for the Arts, the Natural Science and Engineering Research Council of Canada, SMART Technologies, the Canadian Foundation for Innovation, and the Alberta Association of Colleges and Technical Institutes for research support. We would also like to thank the members of the Interactions Lab at the University of Calgary for feedback and support during the development of this project.

References

Lawrence Fyfe, Adam Tindale, and Sheelagh Carpendale. 2011. Junctionbox: A toolkit for creating multi-touch sound control interfaces. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 276–279.

Google. 2012a. Android. `http://developer.android.com/index.html`.

Google. 2012b. Motionevent. `http://developer.android.com/reference/android/view/MotionEvent.html`.

hexler.net. 2012. Touchosc. `http://hexler.net/software/touchosc`.

Toshi Ichiyanagi. 1961. Distance.

Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. 2005. Tuio - a protocol for table-top tangible user interfaces. In *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*.

Uwe Laufs, Christopher Ruff, and Jan Zibuschka. 2010. Mt4j - a cross-platform multi-touch development. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '10, New York, NY, USA. ACM.

Chandrasekhar Ramakrishnan. 2003. Javaosc. `http://www.illposed.com/software/javaoscdoc/`.

Casey Reas and Ben Fry. 2006. Processing: programming for the media arts. *AI & Society*, 20(4):526–538.

Matthew Wright. 2005. Open sound control: an enabling technology for musical networking. *Organised Sound*, 10(3):193–200.