Handwritten Computer Music Programming, Composition, And Design

A Dissertation Proposal for the Center for Computer-based Music Theory and Acoustics, Stanford University

Spencer Salazar

March 16, 2014

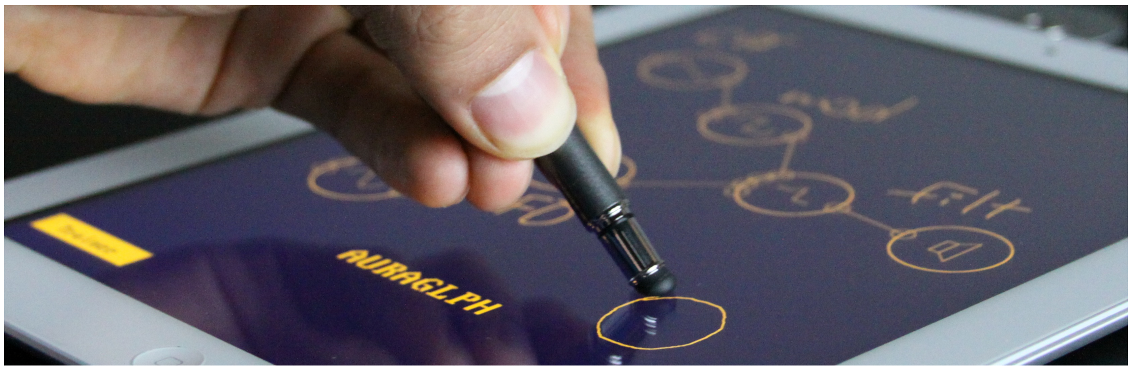

Figure 1: Auraglyph, an iPad application developed to demonstrate concepts of handwritten computer music.

1 Introduction

Touch-based computing has profoundly altered the landscape of mainstream computing in the early 21st century. Since the introduction of the iPhone in 2007 and the iPad in 2010, scores of touchscreen devices have entered the popular consciousness – mobile phones, tablet computers, watches, and desktop computer screens, to name a few. New human-computer interaction paradigms have accompanied these hardware developments, addressing the complex shift from classical keyboard-and-mouse computing to multitouch interaction.

For my dissertation, I propose exploring a new model for touchscreen interaction with musical systems that combines use of stylus-based handwriting input with direct touch manipulation. This system might provide a number of advantages over existing touchscreen paradigms for music. Stylus input, complemented by modern digital handwriting recognition techniques, replaces the traditional role of keyboard-based text/numeric entry with handwritten letters and numerals. In this way, handwritten gestures can both set alphanumeric parameters and write out higher level constructs, such as programming code or musical notation. A stylus also allows for modal entry of generic shapes and glyphs, e.g. canonical oscillator patterns (sine wave, sawtooth wave, square wave, etc.) or other abstract symbols. Finally, the stylus provides precise graphical free-form input for data such as filter transfer functions, envelopes, and parameter automation curves. In this system, multitouch finger input continues to provide functionality that has become expected of touch-based software, such as direct movement of on-screen objects, interaction with conventional controls (sliders, buttons, etc.), and other manipulations. Herein I discuss a proposal for designing, prototyping, and evaluating a system developed according to these concepts, which I have code-named “Auraglyph.”

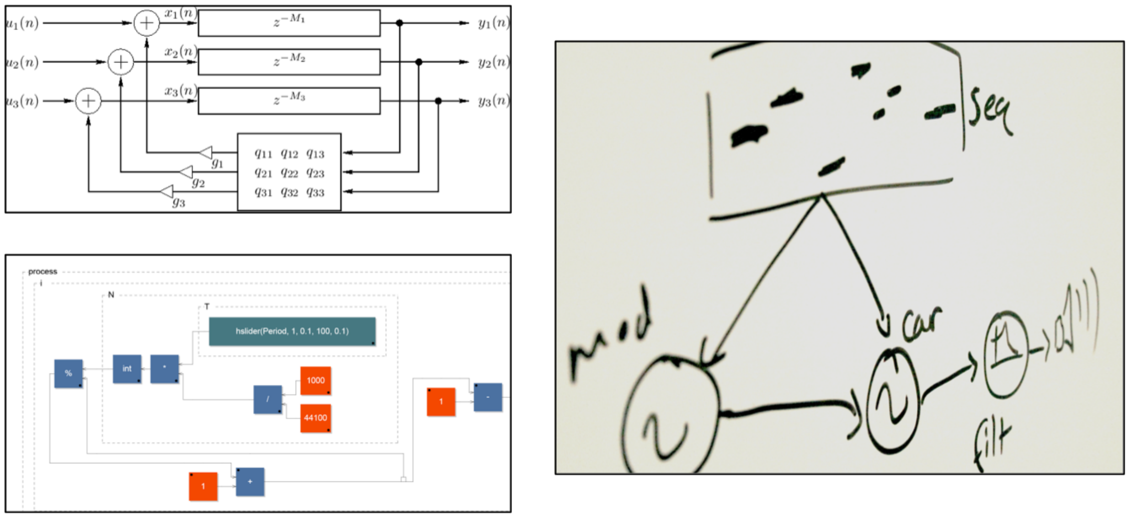

Figure 2: Example block diagrams. Top-left: a feedback delay network [13]. Bottom-left: compiler-generated block diagram from the Faust programming language [4]. Right: hand-drawn block diagram.

2 Related Work

In my research, handwriting recognition has previously found little use in interactive computer music. Fujinaga et al. explored character recognition of traditional staved musical notation [2, 3], as have Miyao and Marayuma [7]. Direct, graphical manipulation of computational data via pen or stylus, augmented by computer intelligence, goes back as far as Sutherland’s Sketchpad [14] and GRAIL and the RAND Tablet by Ellis et al. [1].

Object graph-based programming languages such as PureData [11] and Max/MSP [22] have tremendously influenced the space of visual computer music design and composition. These systems provide both a status quo of graphical music programming and illuminate potential hazards to this approach. More recently, the Kronos programming language extended functional programming models to a visual space in the context of real-time music performance [9]. Mira, an iPad application, dynamically replicates the interface of desktop-based Max/MSP programs, joining conventional music software development with touch interaction [15]. Faust is a text-based functional programming language for designing block diagrams that are compiled to audio processing programs or plugins [10].

Mobile application developer Smule [18] has built software instruments and music experiences tailored to the characteristic features of mobile devices. For instance, Ocarina utilizes the touchscreen, microphone, and accelerometer of a mobile phone to create a unique, compact musical instrument with a network and location-based social aspect [17]. Leaf Trombone further extends this social aspect with real-time, globablly networked musical interactions for critiquing musical performances [20]. Magic Fiddle brings these ideas to the iPad, whose larger size accommodates a different set of interactions, such as finer grained pitch control and touch-based amplitude control [19].

This proposal has been directly motivated by developments in the ChucK programming language [16]. Efforts to work with ChucK code on mobile touchscreen interfaces has led to porting miniAudicle, ChucK’s dedicated development environment [12], to the iPad. However, writing ChucK code with a touchscreen virtual keyboard has proven so far to be unnatural and generally inferior to conventional keyboard-based coding. Similarly, the popular desktop music programming environments SuperCollider [6] and PureData exist in mobile software primarily as backend audio engines rather than full-fledged software development interfaces.

3 Handwritten Computer Music

This proposal is motivated by the desire to better understand the distinguishing capabilities and limitations of touchscreen technology, and, using these as guiding principles, to enable expressive music interactions on such devices. Complex software developed for a given interaction model — such as keyboard-and-mouse — may not successfully cross over to a different interaction model — such as a touchscreen device.

The initial insight leading to this proposal was that numeric and text input on a touchscreen might be more effectively handled by recognizing hand-drawn numerals and letters, rather than an on-screen keyboard. I soon realized that handwriting recognition could also handle a substantial number of handwritten figures and objects beyond just letters and numbers. A user might then draw entire object graphs and audio topologies to be realized as an audio system or musical composition, in real-time, by the underlying software application. Such a system might even be extended to support actual software programming via handwriting.

From here a natural metaphor for interaction arose, that of a handwritten block diagram. In both computer-generated and manually produced variants, block diagrams are utilized extensively in digital audio and computer music development processes (Figure 2). We would like users of our system to feel as if they are drawing a block diagram of their musical system on a whiteboard or sheet of paper, and then for the software to manifest that design.

This metaphor provides several notable benefits. Writing and drawing with a pen or pencil on a flat surface is a foundational interaction for an incredible number of individuals, as this activity is continuously developed from early education around the world. Moreover, perhaps the ultimate goal of software interfaces is to simplify the mapping between the user's conceptual abstraction of an idea and the translation of that idea to a digital representation. The extent to which this goal is met determines the ease of a user successfully inputting their parameters to the software and interpreting its output. The block diagram is a pervasive conceptual abstraction, produced for education (Figure 2, top-left, was taken from a text book), system design (Figure 2, right), and numerous other applications, despite that creating these diagrams is rarely necessary for actually building the represented system. Therefore, a block diagram-oriented design system, with a pen-and-paper interaction metaphor, goes great lengths towards diminishing the distance between human thought process and digital construct.

4 Auraglyph

The primary tangible goal of this dissertation is to produce a complete software environment demonstrating the core principles of handwritten computer music. The working prototype of this environment is called *Auraglyph*.

A high-level model of interacting with Auraglyph is shown in Figure 3. The user's viewpoint is represented by the iPad in the center, acting as a window into a larger system. This viewpoint may be freely moved about the space using conventional gestures (swipe, pinch) to scroll and zoom throughout the space. The lower plane, extending infinitely in two dimensions, contains control logic, programming code, and unit generators, comprising the bulk of an Auraglyph program's functionality. Some unit generators may contain an internal network of logic and ugens designed by the user, and the contents of these may be examined by further zooming in to the container ugen. The upper plane contains user interface controls, which have been defined in the main programming area. This plane exists primarily for interacting with these controls, allowing for Auraglyph programs to create a clearly defined interface.

In producing this dissertation I would like to design, implement, and evaluate a number of interactions that might comprise handwritten computer music, detailed below. My intent is to build these interactions within Auraglyph, an iPad application created specifically to support this research.

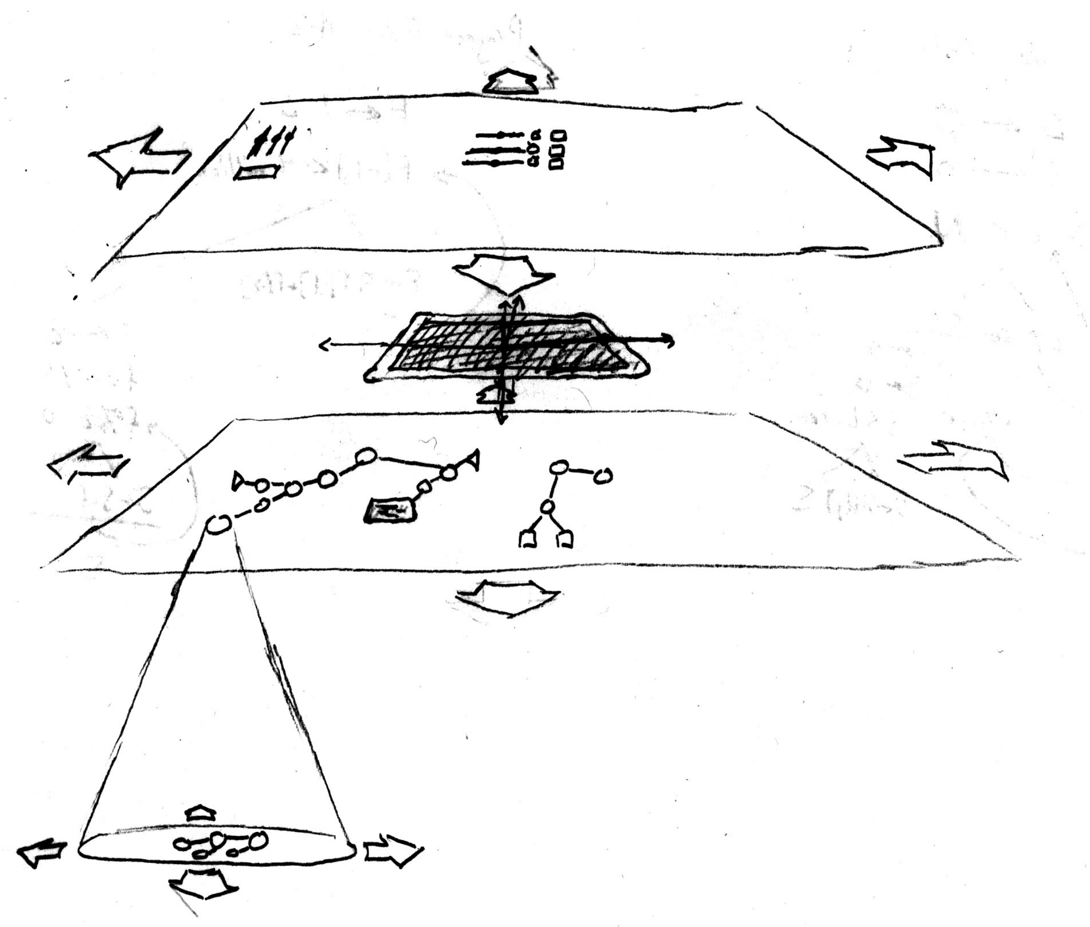

Figure 3: High-level model of Auraglyph interaction.

The fundamental environment within the app is a large, on-screen canvas. Everything that exists in a program created with Auraglyph is fundamentally an object on this canvas.

The user draws various figures on this canvas. Figures that match a designated vocabulary of *glyphs* are recognized by the system, and converted into an *object* that corresponds to that glyph. For example, drawing a circle glyph might cause the hand-drawn circle to be replaced by an audio unit generator object, and drawing a square glyph might create a control-rate processor object (See Figure 7).

Each object represents a functional block, performing some unit-sized task. Objects can be interconnected by drawing connections with a stylus. A given object might represent a reverb unit generator, an audio oscillator, a timer, a control curve, or some sort of textual code. Each object type has a defined set of inputs and outputs, determining the function of each connection between objects. These concepts are common to many graphical, block-based music programming environments.

Modifying and altering the parameters of these objects leverages multitouch and stylus interaction. Handwriting and multitouch input both offer distinct advantages for musical control and design. For example, one might set a filter frequency or bandwidth parameter by writing the number by hand into the system. A user might directly draw the desired waveform for a wavetable oscillator, envelopes, or the magnitude transfer function for a filter. Parametric equalizers and resonant filters can be sweeped by hand, modifying cutoff, bandwidth, and gain parameters with touch gestures.

Code objects support direct input of written programming code. Short blocks of code can be written out within an object, and unbound variables are automatically turned into inputs to the object. An instance of such code might simply be a small mathematical function, or a multi-case logical algorithm. To this end, Auraglyph's programming language ought to be designed specifically for pen input. For example, a user wouldn't be able to necessarily write as many characters of this programming language, due to the slowness of stylus writing, but these characters wouldn't be limited to a linear, sequential arrangment. A handwritten programming language would need to address both of these issues, and others.

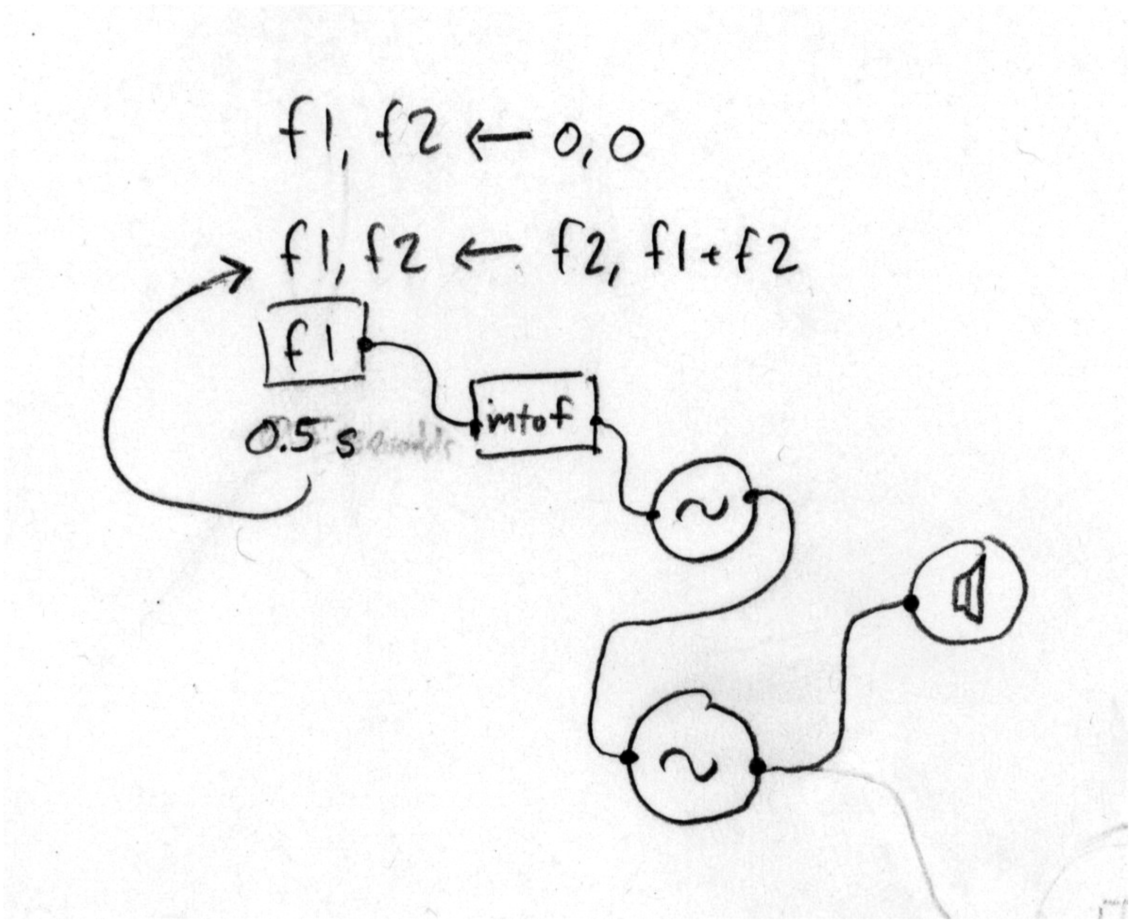

Figure 4: Example of handwritten code integrated into a unit generator graph. This program chooses notes according to the Fibonacci sequence, selecting a new note every half second.

The system facilitates multi-user interactions with objects used to communicate data over the network. Objects might exist to transmit and receive arbitrary control data over OpenSoundControl [21]. Additionally they could provide discovery mechanisms for finding other local-area or wide-area Auraglyph users to make music with.

Stylus input can also be used to hand-write musical notation on staves. Snippets of notation data can then be passed around and modified both with traditional list transformations (add/remove, combine, reverse, etc.) and music-specific transformations (transposition, inversion, tuning, etc.).

Separation of logic and interface. Input and output objects exist for interfacing with the outside world, both on-screen (GUI sliders, buttons, and indicators) and off-screen (MIDI, OSC). For on-screen I/O objects, it is tempting to place them in-line with strictly functional structures, like ugens or code. However, doing so can obscure the relationship between an Auraglyph program's interface and its logic. Therefore, the actual controls represented by I/O objects might be placed into a separate interface space, that is distinct from the code/object/logic space. A user might even use a pinch gesture to "zoom out" to the interface space, and then to "zoom in" back to the logic space.

Novel stylus designs can provide additional expressivity in this system. For example, a stylus with touchscreen-compatible tips on both ends might designate a draw end and an erase end, mimicking the conventional pencil-and-eraser. Similarly, multiple styluses might be distinguished and treated differently by this system, allowing for different "colors." Each stylus in this scenario would support a separate interaction. Distinguishing stylus input from manual touch input also presents interesting design opportunities. Consumer touchscreen styluses generally do not support these features, so specialized styluses may need to be individually constructed to enable these interactions.

5 Evaluation

Of particular interest in exploring these scenarios is Auraglyph's viability as a primary development interface for music software. In what scenarios would I rather use Auraglyph than a desktop software application? What kind of music can I make with Auraglyph? Is the music made with Auraglyph "good?" Does Auraglyph affect how I think about music and computing, and, if so, how? To a significant extent, the answers to these questions form the criteria for success of handwritten computer music.

Direct user testing may also shed light on the utility of Auraglyph for expressive music software programming. Common in the software development industry, this type of design validation gives test users a script of actions to undertake within the program. As each user performs these tasks, relevant metrics are gathered, such as time needed to complete each task, the number of errors made before completing a task, and the user's self-reported assessment of the tasks' difficulty. These tests could be performed with the same script across both Auraglyph and comparable environments like Max/MSP or ChucK. This variety of testing is not conclusive as to any given application's utility or expressiveness, but does allow a quantitative approach that might reasonably suggest its overall quality.

A related issue is the audience of the software. Auraglyph is not necessarily intended for beginners or amateurs (though it would not intentionally discourage them from using it either). A primary goal of Auraglyph is to offer the depth of functionality that one might find in popular music software environments.

6 First Draft

I have already developed an initial draft of Auraglyph, thus far embodying a number of the principles stated above. A description of this system and the concept of handwritten computer music was submitted to the International Conference on New Interfaces for Musical Expression (NIME) 2014.

The basic environment of this version of Auraglyph is an open, scrollable canvas in which the user freely draws with a stylus. Using a variety of pen strokes, a user creates interactive *objects* (such as unit generators, control rate processors, and input/output controls), sets parameters of these objects, and forms connections between them. Collectively, these objects and their interconnections form a *patch*. After a user completes a pen stroke (a single contour between touching the pen to the screen and lifting it off the screen), it is matched

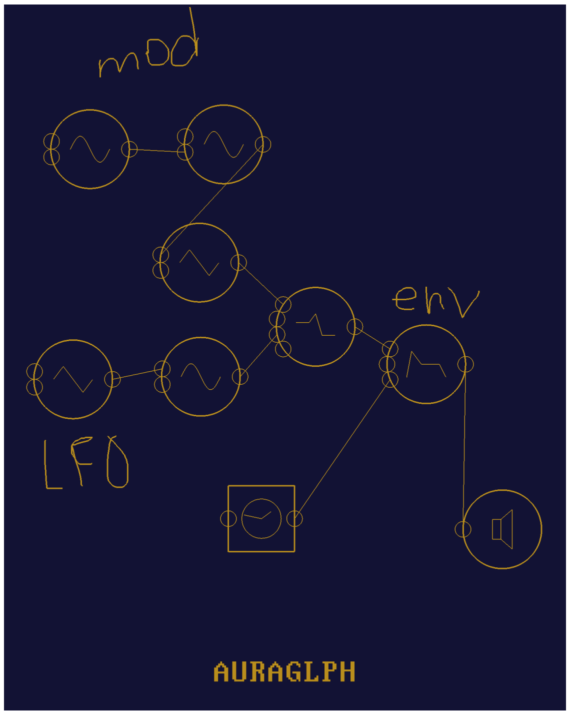

Figure 5: A full Auraglyph patch, with annotations.

against the set of base object glyphs available in the main canvas, via a handwriting recognition algorithm. Main canvas objects whose glyphs can be matched include an audio rate processor (unit generator), control rate processor, input, or output. If the stroke matches an available glyph, the user's stroke is replaced by the actual object. Unmatched strokes remain on the canvas, allowing the user to embellish the canvas with freehand drawings.

Tapping and holding an object will open up a list of parameters for that object (Figure 6). Selecting a parameter from this list opens a control into which writing a number will set the value. This value can then be accepted or discarded, or the user can cancel setting the parameter entirely.

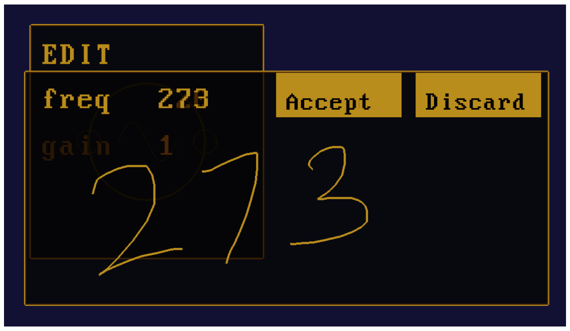

Figure 6: Modifying the "freq" parameter of a unit generator with handwritten numeric input.

Every base object may have inputs, an output, or both. These appear visually as small circles, or *nodes*, on the perimeter of the object. Drawing a stroke from an input node to an output node, or vice-versa, forms a connection between those two objects. For example, connecting a sawtooth object's output to the `freq` input of a sine object creates a simple FM (frequency modulation) patch, with the sine as the carrier wave and the sawtooth as the modulator. Most objects only have one output source, but an input node may have several destinations within that object (e.g. frequency, amplitude, or phase of a given oscillator). In such cases, a pop-up menu appears from the node to display the options a user may have for the input destination.

Objects and freehand drawings can be moved around on the canvas by touching and dragging them, a familiar gesture in the touchscreen software ecosystem. While dragging an object, moving the pen over a delete icon in the corner of the screen will remove that object, along with destroying any connections between it and other objects. Connections can be removed by grabbing them with a touch and then dragging them until they "break." The entire canvas may be scrolled through using a two-finger touch, allowing for patches that extend well beyond the space of the tablet's screen.

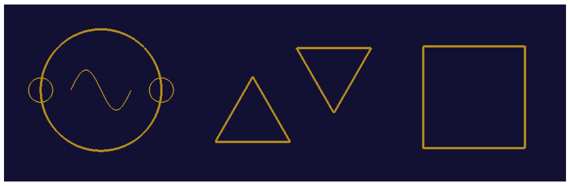

Figure 7: Base objects (left to right): unit generator, output, input, control-rate processor.

Four base types of objects can be drawn to the main canvas: audio-rate processors (unit generators; represented by a circle), control-rate processors (represented by a square), inputs (a downward-pointing triangle), and outputs (an upward triangle) (Figure 7). Unit generators currently available include basic oscillators (sine, sawtooth, square, and triangle waves) and filters (resonant low-pass, high-pass, band-pass). Control-rate processors include a timer. Inputs and outputs are not currently implemented. After creating a base object, a menu opens to allow the user to select an object sub-type (see Figure 8). Selecting a subtype from this menu creates an instance of that object, which may be then parameterized and connected to other objects.

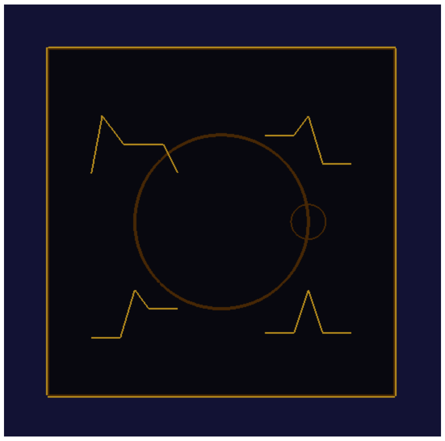

Figure 8: Menu for selecting an object sub-type. Here, we see a menu for a unit generator object, showing ADSR, low-pass filter, high-pass filter, and band-pass-filter sub-types. Scrolling the menu will reveal additional sub-types.

6.1 Handwriting Recognition

The concepts presented herein are generally invariant to the underlying algorithms and framework for handwriting recognition, but these topics merit discussion with regards to our own implementation in

Auraglyph. The handwriting recognition engine used by Auraglyph is LipiTk [5], a comprehensive open-source project for handwriting recognition research. LipiTk is not natively designed to function with iPad applications, but we extended it to do so with straightforward code changes and additions.

LipiTk's default configuration uses dynamic time warping (DTW) [8] and nearest-neighbor classification (k-NN) to match pen strokes to a pre-existing training set of possible figures. The result of this procedure is one or more "most likely" matches along with confidence ratings for each match. We have found the speed and accuracy of LipiTk in this configuration to be satisfactory for real-time usage, though a slight, noticeable delay exists between finishing a stroke and the successful recognition of that stroke.

Before they can be used to classify figures of unknown types, the recognition algorithms incorporated into LipiTk must be primed with a set of "training examples" for each possible figure to be matched. This training set is typically created by test users before the software is released, who draw multiple renditions of each figure into a specialized training program. This training program serializes the salient features of each figure into a database, which is distributed with the application itself.

In our experience, LipiTk's recognition accuracy is highly linked to the quality, size, and diversity of the training set. For instance, a version of our handwriting database trained solely by right-handed users suffered reduced accuracy when used by a left-handed user. A comprehensive training set would need to encompass strokes from a range of individuals of varying handedness and writing style. Interestingly, though, LipiTk's algorithms are able to adapt dynamically to new training examples. An advanced system might gradually adjust to a particular user's handwriting eccentricities over time, forming an organically personalized software interaction. Auraglyph takes advantage of this feature to a small degree, allowing a user to add new training strokes via a separate training interface.

However, as Auraglyph's interactions grow in sophistication and depth, the requirements of the handwriting recognition system may extend beyond what LipiTk innately provides. It is foreseeable that LipiTk's functionality may need to be wrapped into a higher level handwriting parser, aggregating individual characters into larger structures. For example, additional logic would need to classify the geometric arrangement of written text characters if they are to be used to represent programming constructs. Similarly, it is not clear if LipiTk's classifiers would be able to recognize "filled-in" areas in figures, such as the head of a quarter note. Additional research may be necessary to determine an effective solution to these and other issues.

7 Conclusions

I believe that using stylus control and multitouch manual input on touchscreen devices presents a number of novel opportunities for developing music computing interfaces. I have proposed a number of interactions to explore as part of an initial mapping of this space, within the context of an iPad application, code-named Auraglyph. I have further proposed methods for evaluating the success of individual interactions and the system as a whole. While building this system will require overcoming a number of design and technical obstacles, I am confident my experience at CCRMA and elsewhere has more than sufficiently prepared me for undertaking these tasks.

References

[1] M. R. Davis and T. Ellis. The RAND tablet: A man-machine graphical communication device. In *Proceedings of the October 27-29, 1964, fall joint computer conference, part I*, AFIPS '64 (Fall, part I), pages 325–331, New York, NY, USA, 1964. ACM.

[2] I. Fujinaga. Exemplar-based learning in adaptive optical music recognition system. In *Proceedings of the International Computer Music Conference*, pages 55–56, 1996.

[3] I. Fujinaga, B. Pennycook, and B. Alphonce. Computer recognition of musical notation. In *Proceedings of the First International Conference on Music Perception and Cognition*, pages 87–90, 1989.

[4] HarryVanHaaren. SquareWaveExample (Faust online examples). `http://faust.grame.fr/index.php/online-examples`.

[5] S. Madhvanath, D. Vijayasenan, and T. M. Kadiresan. LipiTk: A generic toolkit for online handwriting recognition. In *ACM SIGGRAPH 2007 courses*, page 13. ACM, 2007.

[6] J. McCartney. Rethinking the computer music language: Supercollider. *Computer Music Journal*, 26(4):61–68, 2002.

[7] H. Miyao and M. Maruyama. An online handwritten music symbol recognition system. *International Journal of Document Analysis and Recognition (IJDAR)*, 9(1):49–58, 2007.

[8] R. Niels, L. Vuurpijl, et al. Using dynamic time warping for intuitive handwriting recognition. In *Advances in Graphonomics, Proceedings of the 12th Conference of the International Graphonomics Society*, pages 217–221, 2005.

[9] V. Norilo. Visualization of signals and algorithms in Kronos. In *Proceedings of the 15th International Conference on Digital Audio Effects*, York, U.K., 2012.

[10] Y. Orlarey, D. Fober, and S. Letz. Faust: an efficient functional approach to dsp programming. *New Computational Paradigms for Computer Music*, 2009.

[11] M. Puckette et al. Pure Data: Another integrated computer music environment. *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41, 1996.

[12] S. Salazar, G. Wang, and P. Cook. miniAudicle and ChucK Shell: New interfaces for ChucK development and performance. In *Proceedings of the International Computer Music Conference*, pages 63–66, 2006.

[13] J. O. Smith. *Physical Audio Signal Processing: for Virtual Musical Instruments and Audio Effects*. W3K Publishing, 2010.

[14] I. E. Sutherland. Sketch Pad: A man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop*, pages 6–329. ACM, 1964.

[15] S. Tarakajian, D. Zicarelli, and J. K. Clayton. Mira: Liveness in iPad controllers for Max/MSP. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Daejeon, Korea, 2013.

[16] G. Wang. *The ChucK Audio Programming Language: A Strongly-timed and On-the-fly Environ/Mentality*. PhD thesis, Princeton University, Princeton, NJ, USA, 2008. AAI3323202.

[17] G. Wang. Ocarina: Designing the iPhone's magic flute. *Computer Music Journal*, 38(2), 2014.

[18] G. Wang, G. Essl, J. Smith, S. Salazar, P. Cook, R. Hamilton, R. Fiebrink, J. Berger, D. Zhu, M. Ljungstrom, et al. Smule= sonic media: An intersection of the mobile, musical, and social. In *Proceedings of the International Computer Music Conference*, pages 16–21, 2009.

[19] G. Wang, J. Oh, and T. Lieber. Designing for the iPad: Magic Fiddle. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 197–202, 2011.

[20] G. Wang, J. Oh, S. Salazar, and R. Hamilton. World Stage: A crowdsourcing paradigm for social/mobile music. In *Proceedings of the International Computer Music Conference*, 2011.

[21] M. Wright and A. Freed. Open Sound Control: A new protocol for communicating with sound synthesizers. In *Proceedings of the International Computer Music Conference*, pages 101–104. International Computer Music Association San Francisco, 1997.

[22] D. Zicarelli. An extensible real-time signal processing environment for MAX. In *Proceedings of the International Computer Music Conference*, 1998.