# Auraglyph

## Handwriting Input for Computer-based Music Composition and Design

Spencer Salazar
spencer@ccrma.stanford.edu

Ge Wang
ge@ccrma.stanford.edu

Center for Computer Research in Music and Acoustics (CCRMA)
Stanford University
Stanford, CA 94305

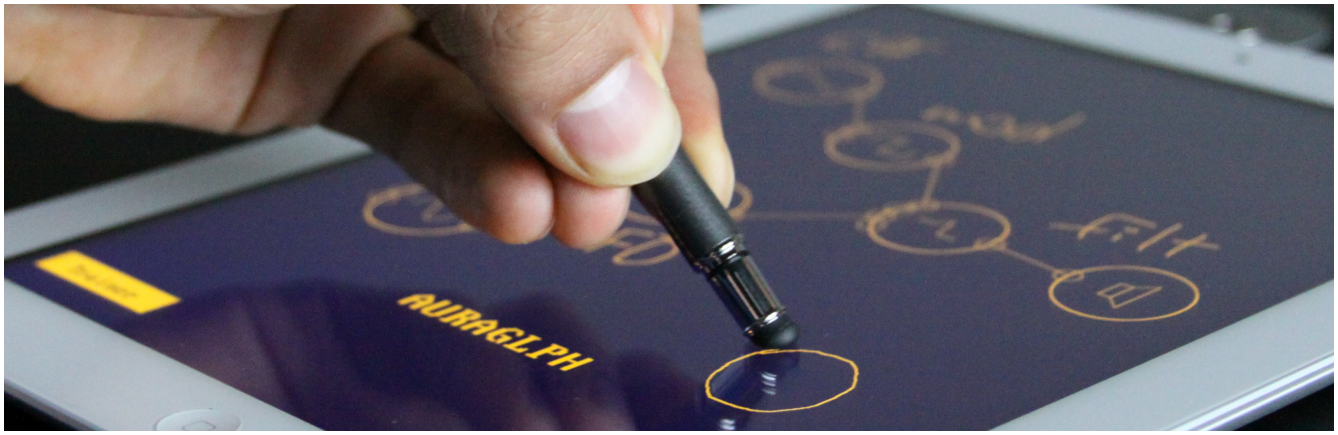


**Figure 1: Auraglyph in use.**

## ABSTRACT

Effective software interaction design must consider all of the capabilities and limitations of the platform for which it is developed. To this end, we propose a new model for computer music design on touchscreen devices, combining both pen/stylus input and multitouch gestures. Such a model surpasses the barrier of touchscreen-based keyboard input, preserving the primary interaction of touch and direct manipulation throughout the development of a complex musical program. We have implemented an iPad software application applying these principles, called "Auraglyph." Auraglyph offers a number of fundamental audio processing and control operators, as well as facilities for structured input and output. All of these software objects are created, parameterized, and interconnected via stylus and touch input. To enable these interactions, we have employed a pre-existing handwriting recognition framework, LipiTk, which is capable of recognizing both alphanumeric characters and arbitrary figures, shapes, and patterns.

## Keywords

handwriting input, computer music systems, handwriting recognition, touchscreen, tablet computing


*NIME'14,* June 30 – July 03, 2014, Goldsmiths, University of London, UK.


## 1. INTRODUCTION

Touch interaction 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, numerous touchscreen devices have entered the popular consciousness – mobile phones, tablet computers, smart 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.

We propose a new model for touchscreen interaction with musical systems: the combined use of stylus-based handwriting input and direct touch manipulation. This system provides a number of advantages over existing touchscreen paradigms for music. Stylus input, complemented by modern digital handwriting recognition techniques, provides a number of advantages in this scheme. Firstly, it replaces the traditional role of keyboard-based text/numeric entry with handwritten letters and numerals. It additionally allows for modal entry of generic shapes and glyphs, for example, 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. 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 we discuss the design, prototyping, and evaluation of such a system, which we have named "Auraglyph."

## 2. RELATED WORK

In our research, handwriting recognition has previously found limited use in interactive computer music. Most notably, Garcia et al. studied the use of handwritten sketches and outlines as a preliminary exercise in electronic composition [5]. This study was then used as the basis for mapping the strokes of an camera-augmented pen to real-time musical synthesis, aiding the compositional process. Graphical manipulation of computational data via pen or stylus, augmented by computer intelligence, goes back as far as Sutherland's Sketchpad [13] and the RAND Tablet by Davis et al. [3], which incorporated the Graphical Input Language (GRAIL) by Ellis et al. [4]. GRAIL is the first system to our knowledge in which computer programs are created via stylus input. In GRAIL, hand-drawn figures are automatically recognized and converted to meaningful symbols within the system, which may then be further modified with stylus gestures, such as drawing connections between nodes and scratching out symbols to erase them.

Recent developments in sketch-oriented computing include Landay's SILK, a software prototyping system in which user interfaces are hand-sketched with a digital tablet [7]. Where appropriate, sketched graphical widgets (buttons, sliders, etc.) are recognized as such, and converted into actual interactive widgets. Interestingly, recognized widgets are not transformed into any canonical visual representation; rather they retain the idiosyncrasies and hafnd-drawn characteristics of the original sketched figures. The SILK system further employs stylus gestures to describe transitions between different parts of the interface, offering an extent of programmability for GUI interactions.

Object graph-based programming languages such as Puredata [12] and Max/MSP [19] have tremendously influenced the space of visual computer music design and composition. More recently, the Kronos programming language extended functional programming models to a block diagram-based visual space in the context of real-time music composition and performance [11]. Mira, an iPad application, dynamically replicates the interface of desktop-based Max/MSP programs, joining conventional music software development with touch interaction [14]. The Reactable [6] and the work of Davidson and Han [2] have both provided significant foundational work in interacting with digital music via touch and direct manipulation.

Languages designed (or repurposed) for live-coding, such as SuperCollider [9] and ChucK [15], suggest interesting possibilities for the expressive use of natural input for music programming. Such systems also constitute the modern incarnation of unit generator-based sound design and synthesis, to which this work owes much of its technical underpinnings. Live-coding practice in general is discussed by Collins et al. [1] and Wang and Cook [17].

Work from the commercial iPhone developer Smule has explored the space of possibilities for musically enabling mobile touchscreen devices [18]. Smule's Ocarina is both an iPhone musical instrument and a musical/social experience, in which performers tune in to each others' musical renditions around the world [16]. Ocarina's feature set (four finger-sized tone-holes on the touchscreen, breath input for dynamic control, tilt-based vibrato, and location-based social networking) is tailored specifically to the technical capabilities, physical dimensions, and limitations of the iPhone. This approach to design — the deliberate consideration of what interactions leverage the characteristic features of a technological platform, and of what interactions are distinctly unsuitable — has directly influenced the development of Auraglyph.

## 3. A CASE FOR HANDWRITING INPUT IN COMPUTER MUSIC DESIGN

This work 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 musical 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 work 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. We soon realized that we could use handwriting recognition to analyze 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 by the underlying software application in real-time.

Evidenced by the previously mentioned work of Garcia et al., handwritten planning is a vital component of many compositional processes, whether the resulting musical product is electronic, acoustic, or both. More generally, writing and drawing with a pen or pencil on a flat surface is a foundational expressive interaction for an incredible number of individuals; this activity is continuously inculcated from early childhood around the world. Therefore sketching, as a natural interaction for expressing ideas in the real world, might be apt for realizing them in a virtual world. The original work described herein seeks to apply this to the context of computer music and audio design. By shortening the distance between a users' abstract musical thought and the audible realization of that thought, handwriting input might arm its users to more effectively express those thoughts.

Another distinct advantage of handwriting input in this context is the ability to run and evaluate musical constructs in real-time. As in Landay's SILK, sketches in Auraglyph are reified into entities specific to the system (e.g., a drawn object might be converted to a sine wave generator, or a timer). These entities can then present controls and interfaces for direct touch manipulation or stylus gestures, customized to that object type or mode. This level of real-time creation and manipulation affords a composer or programmer expressive control similar to live-coding.

## 4. INTERACTING WITH AURAGLYPH

As a framework to prototype and evaluate the concepts discussed above, and for use in composition and sound design, we developed Auraglyph, a software application for iPad. A user interacts with Auraglyph using an iPad-compatible stylus (many models of which are widely available) and traditional touch interaction.

The basic environment of Auraglyph is an open, scrollable canvas in which the user freely draws. 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 against the set of base object glyphs available in the main canvas, via a handwriting recognition algorithm (discussed in Section 4.2). Main canvas objects whose glyphs can be matched include an audio rate processor (unit generator), control rate processor, input, or output (see Section 4.1). If the stroke matches an available glyph, the user's stroke is replaced by

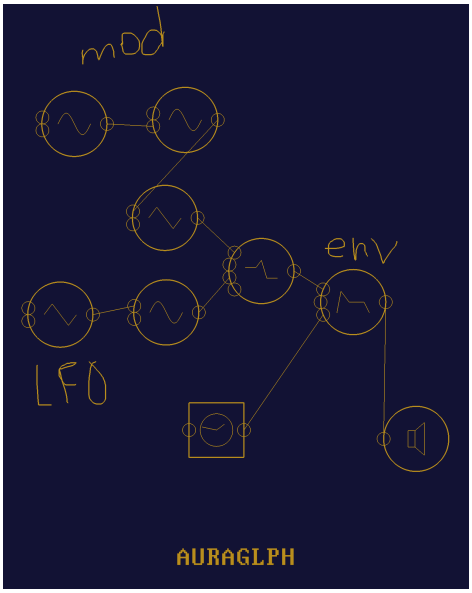


**Figure 2: A full Auraglyph patch, with annotations.**

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 3). 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.

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 has 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.

## 4.1 Objects

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 4). Unit generators currently available include basic oscillators (sine, sawtooth, square, and triangle waves, plus hand-drawn wavetables), an audio file player, envelopes (linear and ADSR), filters (resonant low-


**Figure 3: Modifying the "freq" parameter of a unit generator with handwritten numeric input.**

pass, high-pass, band-pass, band-reject, parametric EQ, or hand-drawn transfer function), and a reverberator. Control-rate processors include timers, counters, mathematical formulae, discrete time-step sequencers, a pitch-to-frequency converter, and hand-drawn parameter control curves. Inputs consist of on-screen keyboard controllers, graphical sliders, and buttons. Outputs consist of numeric displays and binary "LED" indicators.

**Figure 4: Base objects (left to right): unit generator, output, input, control-rate processor.**

After creating a base object, a drawing area and menu open to allow the user to select an object sub-type. In some cases, a glyph for the sub-type can be directly drawn by the user to select that object. For example, basic oscillators and filter types lend themselves to simple glyph patterns. A user can create a sine wave oscillator by simply drawing a circle (creating a generic unit generator object) and then drawing a sine wave within it.

Some object sub-types do not have straightforward glyph mnemonics, for example a sequencer or a reverberator. To support such objects, a menu appears after creating a base object, displaying a list of subtypes to select from. For completeness, objects that can be drawn as corresponding glyphs are also listed in this menu.

Some objects have more advanced needs for modifying values beyond the standard parameter editor. For example, the wavetable oscillator's `table` parameter brings up a large input space for precisely drawing the desired waveform, as does the `xferFn` (transfer function) parameter of the hand-drawn filter. The formula object eschews the standard parameter editor entirely, instead directly soliciting handwriting input of a sequence of mathematical operations (currently addition, subtraction, multiplication, and division are supported, with one input and one output).

## 4.2 Implementation of 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 [8], 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) [10] and nearest-neighbor classification (k-NN) to match pen strokes to a pre-existing training set of pos-

sible 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.

## 5. CONCLUSIONS

While effective in replacing the keyboard as a textual/numeric input device, touchscreen stylus input also affords a broader set of interactions apt for computer music. We have developed an iPad software application, Auraglyph, to leverage these principles into an interactive music environment. We intend to release Auraglyph under an open source license when it is reasonably mature and bug-free, via the project website. [1]

This system has fulfilled our goals of expressivity and versatility, but we believe this conceptual framework provides many more opportunities for computer music. Stylus input might be extended to traditional music notation, more complex mathematical formulae, or even Turing-complete programming code, whose digital representations can then be refined, augmented, and extended with direct manipulation and touch input. We believe that this is only the beginning for handwritten computer music.

## 6. REFERENCES


[1] N. Collins, A. McLean, J. Rohrhuber, and A. Ward. Live coding in laptop performance. *Organised Sound*, 8(3):321–330, 2003.

[2] P. L. Davidson and J. Y. Han. Synthesis and control on large scale multi-touch sensing displays. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 216–219. IRCAM/Centre Pompidou, 2006.

[3] 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.

[4] T. O. Ellis, J. F. Heafner, and W. Sibley. The GRAIL language and operations. Technical report, DTIC Document, 1969.

[5] J. Garcia, T. Tsandilas, C. Agon, W. Mackay, et al. InkSplorer: Exploring musical ideas on paper and computer. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2011.

[6] S. Jordà, G. Geiger, M. Alonso, and M. Kaltenbrunner. The reacTable: Exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, TEI '07, pages 139–146, New York, NY, USA, 2007. ACM.

[7] J. A. Landay. *Interactive sketching for the early stages of user interface design*. PhD thesis, Carnegie Mellon University, 1996.

[8] 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.

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

[10] 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.

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

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

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

[14] 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.

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

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

[17] G. Wang and P. R. Cook. On-the-fly programming: Using code as an expressive musical instrument. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 138–143. National University of Singapore, 2004.

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


[1] `https://ccrma.stanford.edu/~spencer/auraglyph/`