AN AUDIOVISUAL WORKSPACE FOR PHYSICAL MODELS

Benjamin Schroeder, Marc Ainger, Richard Parent
The Ohio State University
benschroeder@acm.org, ainger.1@osu.edu, parent@cse.ohio-state.edu

ABSTRACT

We present an experimental environment for working with physically based sound models. We situate physical models in an interactive multi-modal space. Users may interact with the models through touch, using tangible controllers, or by setting up procedurally animated physical machines. The system responds with both real-time sound and graphics. A built-in strongly-timed scripting language allows for a different kind of exploration. The scripting language may be used to play the models with precise timing, to change their relation, and to create new behaviors. This environment gives direct, concrete ways for users to learn about how physical models work and begin to explore new musical ideas.

1. INTRODUCTION

Physically based sound synthesis is, for the user, both familiar and richly expressive. Physical models correspond to real-world objects, and variations in instrument design and playing style may be specified using real-world concepts like shape, forces, and material properties.

In this paper we describe an experimental environment for working with physical models. Interaction in our environment is direct and concrete; it corresponds to real-world experiences but goes beyond the strictly physical.

There are several ways the model can be affected. The models are situated in a virtual space and may be played using touch. Tangible controllers may be used to influence the model or the space they reside in. Small "machines" based on procedural animation and physics give another way to explore relations between space, time, and rhythm. A textual scripting language provides for more precise timing, deeper exploration, and extension of the system's behavior.

Our environment is multi-modal: models respond both aurally and visually in real time. We support touch input (as well as conventional mouse input) and input from different kinds of tangible controllers. We intend for the users' different senses - auditory, visual, and kinesthetic - to work together to create a fuller interaction experience.

This work is certainly inspired by other sound systems which use interactive touch, most notably the Reactable [1]. The Reactable situates sound objects in spatial relation and uses space to create rhythmic patterns. Its objects are based on signal processing and abstract notation; our research uses physically based models and a more concrete, physical notion of space.

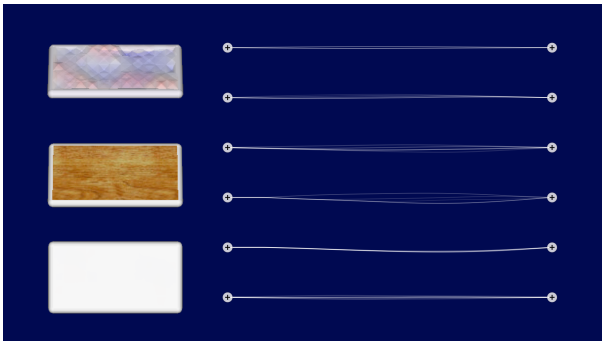

Figure 1. Several strings and plates in the direct-manipulation environment.

In what follows, we introduce the different kinds of interaction supported by our system: direct manipulation, tangible controllers, procedural animation, and textual scripting. We then conclude with a brief discussion of questions to be addressed in future research.

1.1 Video and audio examples

Video examples (with sound) demonstrating several features of our system are available on the web at *http://www.youtube.com/user/avworkspacesmc2010*.

2. SPATIAL, MULTI-MODAL INTERACTION

Physical models, like their real-world counterparts, are situated in time and space. They make sound when things interact with them in physical ways: plucking, bowing, striking, fretting. This suggests a graphical setting in which models may be arranged spatially and interacted with using direct manipulation.

Our environment includes strings and plates as sounding primitives. Figure 1 shows several strings and metal plates. A user can play the strings by "plucking" them with the mouse. Moving across several strings at once produces a strum. Tapping on a plate makes a clanging sound.

Since the strings and the plate are implemented using physical models, they respond realistically to differing input. For example, real strings and plates produce different sounds depending on where they are plucked or struck due to the excitation of different vibrational modes. This is true

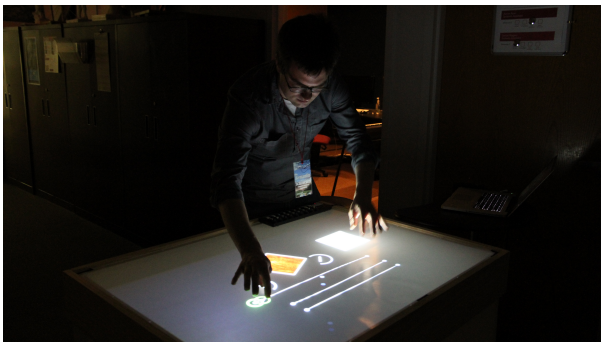

Figure 2. The system running on a diffuse-illumination multitouch table.

of our models as well.

Sound in a physical model comes from the simulation of an object's vibration. The models in our system respond visually as well as sonically, giving users a way to build intuition about how changes in shape produce changes in sound. (The visual response may be exaggerated in scale far beyond what is realistic.) Time in the system may be stretched into slow motion, allowing for close examination of how waves progress through an object. Most of the graphics in our system are shown in a 2D diagrammatic perspective, but plates may be tilted in 3D to provide a better view of their complex vibrations.

Models may also be changed through simple direct manipulation. Strings and plates may be stretched to change their size and hence the sound they produce. String tension may be set by manipulating a string's end as though using a tuning peg. All of these changes are made interactively; changes can even be made while an object is sounding.

A multitouch surface is a natural setting for this kind of interaction, and we have experimented with running our system on a diffuse-illumination multitouch table. (Our table is shown in Figure 2; a brief description of the technology is given in Appendix B.) Although touch is not strictly necessary for our system, the availability of multiple touches gives rise to additional kinds of interaction, such as fretting a string to produce different notes. Furthermore, the table setting allows multiple users to play at once.

2.1 Physical model implementation

Our string and plate primitives are implemented using finite differences [2]. In principle, any physical modeling technique could be used, as long as it accepts input in terms of forces and positions and provides output in these same terms.

Finite difference models work especially well in this regard; physical quantities are calculated for each point on a finite-difference grid at every time step, making it trivial to use a model's output for such things as real-time visualization. Input is in terms of these same physical quantities. By contrast, frequency-based techniques such as modal synthesis accept input in terms of forces, but require more computation in order to provide output in terms of physical quantities rather than fully synthesized waveforms.

We use a string model described by Chaigne and Askenfelt [3] and a plate model given by Bilbao [4]. Our string plucking model is a partial implementation of that described by Cuzzucoli and Lombardo [5]. The main equations for the string and plate are reproduced here for convenient reference and in order that we might mention the available parameters; for full details, please see the appropriate papers.

The string model is given by the following equation, which describes a string's basic motion as well as internal and radiative damping. The equation also models dispersion due to stiffness, as occurs, for example, in piano strings.

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial^2 y}{\partial x^2} - \epsilon c^2 L^2 \frac{\partial^4 y}{\partial x^2} - 2b_1 \frac{\partial y}{\partial t} + 2b_3 \frac{\partial^3 y}{\partial t^3} + f(x, x_0, t). \quad (1)$$

In this equation, c is the speed of sound on the string, which incorporates tension and the string's mass density; L is the string's length; b_1 and b_3 are damping constants. The coefficient ϵ describes the string's stiffness. The function f accounts for force interaction over time.

The plate model is similar but is given in two dimensions.

$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2 \nabla^4 u + c^2 \nabla^2 u - 2\sigma \frac{\partial u}{\partial t} + b_3 \frac{\partial}{\partial t} \nabla^2 u + f(x, y, t). \quad (2)$$

Here, κ describes the plate's stiffness; c is again the speed of sound due to any applied tension. The coefficients σ and b_3 are damping constants [1]. The function f again represents force interaction.

3. TANGIBLE CONTROLLERS

We have discussed one way to influence the system of sounding objects: using touch and spatial motion. Another way is to use tangible physical controllers and sensors. These expand the expressivity of the system, giving us ways to map things like 3D motion, shape, pressure, or temperature into our environment. In addition, such controllers have a satisfying physical presence; even simple controls like sliders and buttons can seem more "real" when made out of actual plastic and steel rather than pixels.

Our environment can receive messages directly from MIDI controllers and Wii remotes. It can also receive Jitter network messages, allowing for the use of Max/MSP/Jitter as a sort of pre-processing frontend. Messages of this sort can easily be made to control the parameters of a model, such as its tension, length, or position, or to induce actions such as plucks.

More interestingly, controllers might be used to affect the physical environment in broader ways. Figure 3 shows a breath controller given a virtual presence. It may be moved and turned like any other object via direct manipulation. Blowing into the controller causes a "wind" of particles to enter the space. The particles pluck the strings and

[1] Bilbao gives this coefficient as b_1, but we have here used b_3 for consistency with the string model.

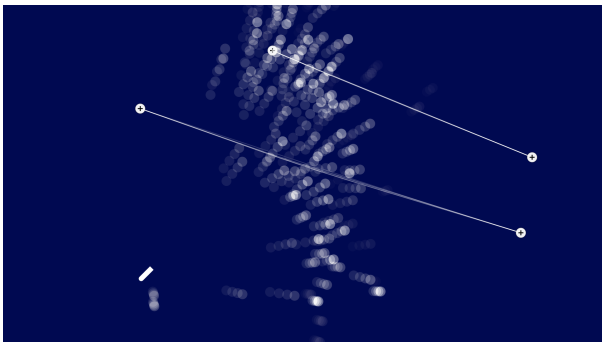

Figure 3. A "wind" of particles from a breath controller.

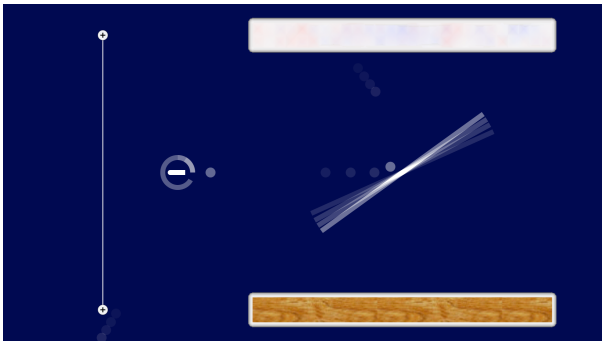

Figure 4. A basic procedural machine.

bounce off the plates. We might now consider using a camera system and digital compass to map a person's physical location and orientation into the system, letting them in a sense move through the simulated space, blowing wind as they go.

4. PROCEDURAL MACHINES

Procedural animation techniques introduce more variety into the graphical environment and give ways to explore the role of algorithms in sound generation. We have already seen one example of this: the particle system of the breath controller's "wind". As encapsulated algorithms, procedural animations also allow for actions to be repeated in more precise ways than direct interaction does, and for the actions to be edited over time. This supports an iterative style of design.

Figure 4 shows a basic procedural machine. In this system, there is an emitter on the left that sends particles into the environment at a regular rate, like a metronome. The slab to its right rotates back and forth, bouncing the particles either to the top plate, made of metal, or to the bottom plate, made of wood. Occasionally a particle ricochets off the slab head-on and plucks the string behind the emitter. By controlling the rate and angle of particle emission, the user can make different rhythmic patterns and learn about relationships between organized space, velocity, and time.

Other kinds of emitters are possible. For example, we have built objects that split incoming particles into two. Gravity (in any direction and strength) adds another twist

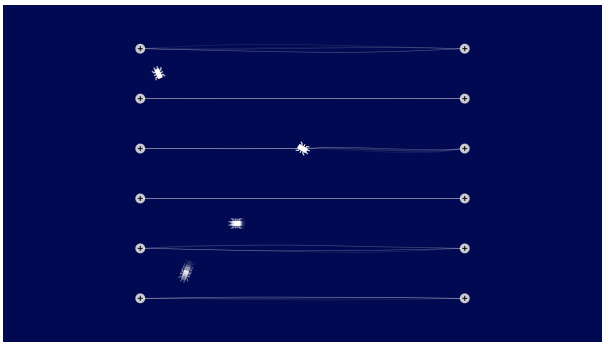

Figure 5. Procedural crawlers which fret strings.

to the possibilities of particle motion.

Physics is not the only way to interact algorithmically with the system. In Figure 5, small robots crawl around the space, moving to random locations on a grid. When a crawler moves to a string, it latches on, fretting the string for a time, before moving off in a random direction again.

5. INTERACTIVE PROGRAMMING

So far, we have described several high-level ways to interact with sound models. Our environment also includes a textual language which provides for low-level access to the models. This has two purposes. First, textual scripting is a flexible way to set up direct-manipulation environments like those described above and to describe new procedural elements. Scripts, like procedural animations, support iterative design by allowing for the reuse and careful editing of algorithms.

Second, while direct manipulation is approachable and concrete, text is a good choice for describing more sophisticated interactions. In particular, it is more apparent how to use parameterization and abstraction in this context, and timing may be made more precise.

The textual language is interactive and runs in conjunction with the direct manipulation environment. This allows a user to go back and forth between the two as desired.

A complete reference is beyond the scope of this paper. However, the examples that follow are intended to give an idea of how the language works. We introduce the examples with a short discussion of the textual language's general structure and philosophy.

5.1 Language structure and philosophy

The textual language is a prototype-based object-oriented language; it is similar in many ways to Self [6] or Smalltalk. Our concern in this environment is mainly for ad-hoc and on-the-fly programming, rather than the construction of larger software systems. We have therefore emphasized programming facilities that support experimentation, rather than ones that support longer-term efforts. Although the underlying language is capable of representing complex abstractions, we have mainly concentrated on its use in an interactive context.

Figure 6. Scripting takes place through an interactive workspace.

Most interaction with the language happens through a text window called a "Workspace" (Figure 6). This is similar to the interactive command line or read-eval-print loop provided by some environments, but it retains more context from step to step. Code in a Workspace can be executed line-by-line; results may be printed; old code can be revisited. In keeping with the philosophy of supporting experimentation, variables set at the top level of our Workspaces are defined automatically, allowing users to define names as they go.

The textual environment runs concurrently with the direct-manipulation environment and with audio output. In this way, results from code execution can be seen immediately, and changes in the output can be investigated more closely using code. Users can move freely between the two levels as needed; in a multi-user setting, one person might even be playing an instrument using touch while the other modifies it using code.

Our language follows ChucK [7] in being *strongly timed*. The simulation time for a given thread proceeds (and audio samples are calculated) only when the programmer explicitly asks for it to do so. Code between such statements is considered, from the perspective of the simulation, to execute instantaneously. In this way, the programmer is given full control over timing and coordination.

In our environment, it is possible that no user-level code is running at some particular time, but instead that all interaction is taking place through the direct-manipulation environment. In that case, simulation time proceeds along with real time.

5.2 A simple example

Imagine that you wanted to try playing some notes on plucked strings. You might start by making a string and tuning it.

```
stringD: stringModel make.
stringD frequency: 146.80.
```

This puts a string, tuned to D, on the screen, and assigns it the variable name "stringD". You can pluck the string by hand or using code.

```
stringD pluckAtFraction: 0.7.
```

We can add a second string, positioning it a little below the first. Plucking the strings at timed intervals while changing their notes plays a familiar tune. (Quotes are used below to add comments.)

```
stringA: stringModel make.
stringA frequency: 110.
stringA center: 0 @ -0.01.

stringD fretAtIndex: 2. "E"
stringD pluckAtFraction: 0.7.
simTime advance: 0.25 seconds.

stringD fretAtIndex: 0. "D"
stringD pluckAtFraction: 0.7.
simTime advance: 0.25 seconds.

stringA fretAtIndex: 3. "C"
stringA pluckAtFraction: 0.7.
simTime advance: 0.25 seconds.

stringD pluckAtFraction: 0.7.
simTime advance: 0.25 seconds.

stringD fretAtIndex: 2. "E"
3 timesRepeat:
  [stringD pluckAtFraction: 0.7.
  simTime advance: 0.25 seconds].
```

5.3 Making physical changes

One of the strengths of many physical models is their ability to represent different kinds of material. Strings are created by default as nylon strings. We can change the strings above to be steel strings by assigning new material properties.

```
{stringA. stringD} do:
  [:each |
  each
    changeMaterialDensity: 7800.0
    youngsModulus: 200e9].
```

For convenience, this keeps the strings at the same frequency as before by adjusting their tension to match the new material. The strings will sound slightly different due to their new material properties and tension. Other properties of the models, such as damping coefficients, may be changed as well.

5.4 Adding new behaviors

The lengths of the strings in the direct manipulation environment can be changed; this also changes their frequency. The tune in the code above is written in terms of the built-in frets, which are proportional to the length of the string (they are set at half-steps, like guitar frets). Therefore, changing the length of the string would transpose the tune up and down; this could be done while the tune was playing to act as a sort of simple (if not strictly realistic) tremolo arm (or "whammy bar").

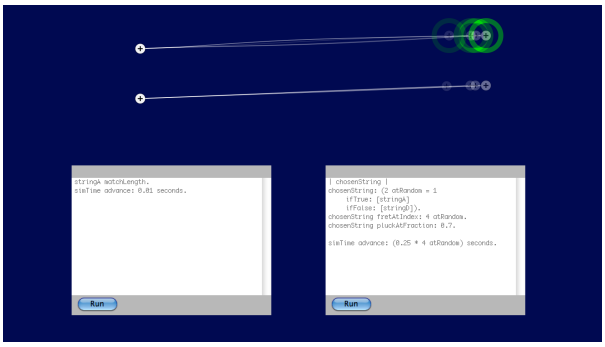

Figure 7. Facilities for running code in continuous loops.

However, the tune is written on two strings. It would be difficult to change these in exactly the same way simply by dragging. We can keep the second string up to date with the first using the following code.

```
stringA learn:
  (|
  partner <- nil.
  matchLength = (| delta |
    delta: partner endPoint
      - partner startPoint.
    endPoint: startPoint + delta).
  |).
stringA partner: stringD.

[[true] whileTrue:
  [stringA matchLength.
  simTime advance: 0.01 seconds]] fork.
```

This code first teaches the A string a new behavior, how to match length with a partner. It then creates a new independent thread which asks the string to match length every hundredth of a second.

Note that this acts like a constraint, but it is only an approximation of one. In particular, it is not updated continuously, and the custom matching code could be expensive to run frequently. A facility for constraints is a topic for future research.

Forking new continuously-running threads is useful for many things. When working with threads, it is important to be able to stop and start them at will, while retaining their code. (In the example above, the thread never stops.) The graphical environment therefore has a facility for running continuously-looping code, as shown in Figure 7.

One could envision a way to express the length-matching behavior via direct manipulation, such as attaching all of the strings to a rigid bar and then moving the bar: a kind of virtual tremolo arm to match the behavior. The scripting environment lets us prototype new ideas before adding them at higher levels.

5.5 New procedural objects

Code may also be used to create new kinds of objects for the procedural animation environment. Figure 8 shows a simple but complete example of an emitter that sends

```
firework: proceduralElement make.
firework learn:
  (|
  emit = (| newBall. theta |
    newBall: ball make.
    newBall radius: 0.01.
    newBall dragFactor: 0.01.
    newBall beFading.
    newBall position: position.

    theta: ((24 atRandom) * 15.0)
      asRadians.
    newBall impulse:
      (theta cos @ theta sin)
      * (0.1 @ 0.1)).

  burst = (||
    40 timesRepeat:
      [| waitTime |
      emit.

      waitTime: (5 + 5 atRandom)
        milliseconds.
      simTime advance: waitTime]).
  |).
```

Figure 8. A "fireworks" animation object.

a burst of particles into the space at random angles and speeds. It may be activated by executing `fireworks burst`.

6. CONCLUSION AND FUTURE WORK

We have described our interactive environment for experimenting with physical models in space and in code. The environment supports several different kinds of interaction with the models, from direct playing to the use of procedural animation and physics and exploration through code.

Our goal with this environment has been to provide for easy, high-level experimentation with physical models. However, the range of expressiveness of physical models is deep, and we have only scratched the surface.

For example, we might want facilities for creating new models of a player's interaction with a string: different kinds of plucks, bowing, fretting models, playing string harmonics, rasgueado, and more. We might consider physical models that change over time in unrealizable but physically plausible ways, such as a web of strings that shifts around, splitting and reconnecting over time. Models might be connected to one another and resonate. Input from more sophisticated tangible controllers or even other instruments might be used to drive models. In addition, several musically important concepts addressed in existing languages go unaddressed here.

In our future research, we hope to address deeper questions such as these. We expect that new abstractions, methods of interaction, and language facilities will help make the full power of physical models truly accessible.

7. REFERENCES

[1] M. Kaltenbrunner, S. Jorda, G. Geiger, and M. Alonso, "The reactable*: A collaborative musical instrument," *Enabling Technologies, IEEE International Workshops on*, pp. 406–411, 2006.

[2] S. Bilbao, *Numerical Sound Synthesis*. Chichester: John Wiley & Sons, Ltd., 2009.

[3] A. Chaigne and A. Askenfelt, "Numerical simulations of piano strings. I.," *Journal of the Acoustical Society of America*, vol. 95, no. 2, pp. 1112–1118, 1994.

[4] S. Bilbao, "A finite difference plate model," in *Proc. of Intl Comp. Music Conf. (ICMC 2005)*, 2005.

[5] G. Cuzzucoli and V. Lombardo, "A physical model of the classical guitar, including the player's touch," *Comput. Music J.*, vol. 23, no. 2, pp. 52–69, 1999.

[6] D. Ungar and R. B. Smith, "Self: The power of simplicity," *SIGPLAN Not.*, vol. 22, no. 12, pp. 227–242, 1987.

[7] G. Wang and P. R. Cook, "ChucK: a concurrent, on-the-fly audio programming language," in *Proc. ICMC*, pp. 219–226, 2003.

[8] E. Catto, "Box2D," 2010. http://www.box2d.org. Accessed April 22, 2010.

[9] NUI Group Community, "Community core vision," 2010. http://ccv.nuigroup.com/. Accessed April 27, 2010.

[10] Hiroaki *et al.*, "DarwiinRemote," 2008. http://sourceforge.net/projects/darwiin-remote/. Accessed April 22, 2010.

[11] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza, "Tuio: A protocol for table-top tangible user interfaces," in *6th International Gesture Workshop*, 2005.

[12] NUI Group Authors, "Multi-touch technologies," 2009. http://nuicode.com/projects/wiki-book. Accessed April 27, 2010.

[13] G. P. Scavone and P. R. Cook, "RTMidi, RTAudio, and a Synthesis Toolkit (STK) update," in *In Proceedings of the International Computer Music Conference*, 2005.

A. IMPLEMENTATION NOTES

Our system runs in real time on commodity Macintosh hardware. We make use of several open-source libraries: RtMidi and RtAudio [13], DarwiinRemote [10], and the Box2D physics engine [8]. Our multi-touch implementation uses the CCV tracking system [9] and the TUIO multitouch protocol [11].

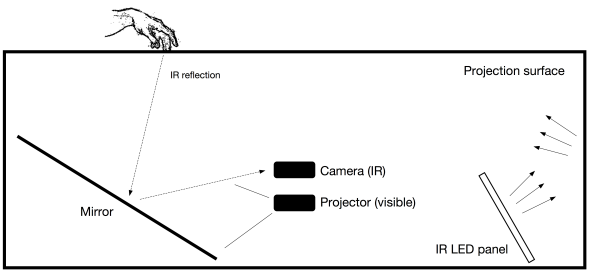

Figure 9. A diffuse-illumination multitouch table.

B. DIFFUSE-ILLUMINATION MULTITOUCH

One setting for our system has been a diffuse-illumination multitouch table. This appendix contains a brief discussion of how such tables may be implemented. A more complete description of this and other multitouch implementation techniques may be found in an excellent online book by the NUI Group [12].

Multitouch tables allow for direct interaction with displayed images. A major question in the implementation of such tables is therefore how to detect touches. Diffuse-illumination tables detect touches via a camera system; to distinguish between actual touches and changes in the display, they rely on the difference between visible and infrared light.

As shown in Figure 9, both a projector and a camera are mounted below the table surface. A mirror may be used (as shown here) to allow for larger projections without requiring the table to be very deep.

The projector displays a video image on a projection surface at the top of the table. This image is made up only of visible light; commodity projectors typically do not emit any infrared light. A number of infrared LED panels also illuminate the projection surface. Since these only emit light in the infrared spectrum, they are invisible to users of the table. The illumination is meant to be spread evenly across the surface. To some extent, the light is diffused by the surface itself, which is semi-opaque; to further avoid "hot spots" of illumination, light is often bounced off of a wall or other surface before reaching the top of the table.

The camera is equipped with an infrared-passing filter; it only "sees" infrared light and does not react to changes in the visible part of the spectrum. If nothing is touching the table, then, the camera sees a static image, even if the video image displayed by the projector is changing.

If a user touches the table, light reflects more strongly from the location of the touch. Techniques from computer vision are used to isolate the area of the touch from the surrounding static background. Multiple touches are seen as separate areas in the resulting image. These touches are tracked over time; their locations and shapes are passed to higher-level software such as our system for further processing.