The HAPTIC TOUCH Toolkit: Enabling Exploration of Haptic Interactions

Ledo, David; Nacenta, Miguel A.; Marquardt, Nicolai; Boring, Sebastian; Greenberg, Saul

The HAPTICTOUCH Toolkit: Enabling Exploration of Haptic Interactions

David Ledo[1], Miguel A. Nacenta[1], Nicolai Marquardt[1], Sebastian Boring[1] and Saul Greenberg[1]
[1]Department of Computer Science
University of Calgary, 2500 University Drive NW
Calgary, AB, T2N 1N4, Canada
[2]Department of Computer Science
University of St. Andrews, College Gate,
St Andrews, Fife KY16 9AJ, Scotland, UK
[dledomai, nicolai.marquardt, sebastian.boring, saul.greenberg]@ucalgary.ca, mans@st-andrews.ac.uk

ABSTRACT
In the real world, touch based interaction relies on haptic feedback (e.g., grasping objects, feeling textures). Unfortunately, such feedback is absent in current tabletop systems. The previously developed Haptic Tabletop Puck (HTP) aims at supporting experimentation with and development of inexpensive tabletop haptic interfaces. The problem is that programming the HTP is difficult due to interactions when coding its multiple hardware components. To address this problem, we contribute the HAPTICTOUCH toolkit, which allows developers to rapidly prototype haptic tabletop applications. Our toolkit is structured in three layers that enable programmers to: (1) directly control the device, (2) create customized combinable haptic behaviors (e.g. softness, oscillation), and (3) use visuals (e.g., shapes, images, buttons) to quickly make use of the aforementioned behaviors. Our preliminary study found that programmers could use the HAPTICTOUCH toolkit to create haptic tabletop applications in a short amount of time.

Author Keywords
Haptics, Tabletop, Touch Interface, Haptic Tabletop Puck, API, Toolkit, Rapid Prototyping, Enabling Technologies.

ACM Classification Keywords
H5.2 [Information interfaces and presentation]: User Interfaces – haptic I/O, input devices and strategies; D.2.2 [Software Engineering]: Design Tools and Techniques.

General Terms
Design, Human Factors.

INTRODUCTION
Touch based interfaces let users interact with computers through touch. In the vast majority of these systems, the communication back from the computer happens exclusively through visual and auditory channels. This represents a lost opportunity in human-computer interaction, as we know that haptics can provide a rich bi-directional channel that goes beyond the homogeneous and unchanging rigid plane offered by most interactive touch surfaces. Unfortunately, and despite the enormous development of the field of haptics, developing hardware and software that provides a compelling haptic experience is still expensive and requires specialized knowledge making programming this kind of interfaces difficult.

Figure 1. Behavior Lab lets developers explore, combine and feel diverse haptic behaviors before writing code.

As one promising approach for exploring haptic feedback within tabletop interaction, Marquardt et al. introduced an open source haptic device platform called the *Haptic Tabletop Puck* (or HTP) [13]. It uses inexpensive do-it-yourself (DIY) hardware and low-level software. The HTP offered (1) a single-point rod providing haptic output via rod height, (2) the same rod reacting to finger pressure for input, and (3) controllable friction as the puck was moved across the surface. Collectively, the HTP enables users and programmers to experience and experiment with a rich haptic channel in a multi-user tabletop environment, where it provides an opportunity for researching haptics in a broad range of applications.

The problem is that programming even this simple haptic device requires the programmer to learn complex haptics models (e.g., the interaction between input pressure and output force). In addition, programmers have to understand low-level details of the multiple underlying hardware components (i.e. pressure sensor and servo motors).

To encourage research in tabletop haptics, we contribute the HAPTICTOUCH toolkit that simplifies the development of haptic-enabled applications for surface-based interfaces via a multi-level API and an interactive Behaviour Lab. Our work offers four contributions:

Cite as:
Ledo, D., Nacenta, M., Marquardt, N., Boring, S., and Greenberg, S. (2011) The HapticTouch Toolkit: Enabling Exploration of Haptic Interactions. *Research Report 2011-1012-24* Department of Computer Science, University of Calgary, Calgary, AB, Canada T2N 1N4, July.

- a working downloadable toolkit that simplifies programming on the HTP platform;
- a series of abstractions and application programming interface (API) organization to enable programming surface-based (2D) haptic interfaces (Figure 3), some of which may be generalizable to other haptic devices;
- the Behavior Lab (Figures 1 and 3) that lets developers explore, combine and feel diverse haptic behaviors before writing code, and
- a preliminary exploration on the usability of the API and its abstractions

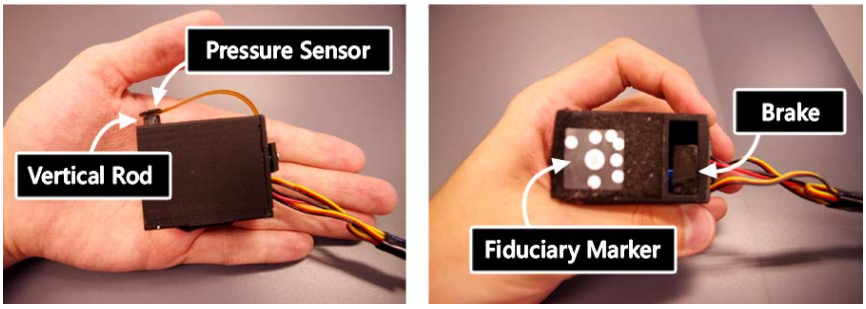

Figure 2. The HTP's main components [13].

BACKGROUND

The toolkit and its API described in this paper are extensions of the Haptic Tabletop Puck [13]. In this section, we revisit this device and its functionality. We then review other relevant haptic platforms and devices, and APIs.

The Haptic Tabletop Puck

The HTP is an active tangible device that works on a surface that recognizes fiduciary markers, e.g., the Microsoft Surface. It contains the following elements (see Figure 2).

a) ***Haptic output via a movable vertical rod.*** A movable cylindrical rod comes out of a small brick-shaped casing. A small servo motor hidden inside the case controls the up and down movement of the rod.
b) ***Haptic input, also via the rod.*** A pressure sensor atop the rod measures a user's finger pressure on it.
c) ***Friction.*** A friction brake at the bottom of the HTP is implemented by a small rubber plate whose pressure against the surface is controlled by another servo motor.
d) ***Location.*** The location of the HTP on the table is tracked through a fiducial marker on its bottom.

Collectively, the HTP enables three main sources of haptic information:

- ***Height.*** The vertical movement of the rod can represent irregular surfaces and different heights.
- ***Malleability.*** The feedback loop between applied pressure and the rod's height can simulate the dynamic force-feedback of different materials.
- ***Friction.*** The brake can modify the puck's resistance to movement in the horizontal plane.

These information sources can be controlled dynamically according to different parameters, such as the puck's position and orientation on the table, time, and so on.

Haptic Platforms and Devices

Haptics and tactile interfaces are very active research areas. In this section we focus on reviewing work that relates to the facilitation of programming and prototyping of haptic interfaces. Correspondingly, we do not cover comprehensively tangible development APIs (e.g., [11]), since they focus mostly on passive tactile experiences which do not have programmable tactile behaviors. We also disregard hardware prototyping APIs and toolkits (e.g., Phidgets [6]), which are not focused on creating tactile experiences even though they can be used to develop haptic hardware.

Haptic interactive devices that have application building and prototyping support available can be separated into two largely distinct groups: *one-dimensional haptic output devices*, and *multi-dimensional haptic input-output devices*. The most common representatives of the first class are vibrotactile interfaces built into consumer devices such as game controllers and mobile phones. These haptic devices are relatively easy to program, since the haptic channel is unidirectional (i.e., they serve only as output—it is a haptic open-loop). Examples of output with this kind of haptic channel are haptic icons [2], which can be designed with the help of specific tools [4,24]. Further, the MOTIV SDK [7] facilitates using vibrotactile feedback in mobile phones and tablets. It provides functions translating sound and visual feedback into vibrotactile signals on touch-based mobile interface platforms (e.g., Android phones).

In comparison, the HTP provides a *bi-directional* haptic channel that is richer than the one present in these devices and tools. The extended interactivity puts different requirements on the design of the HAPTICTOUCH toolkit. However, our work resembles the work above, as we adopt a simple and more familiar GUI paradigm (as in MOTIV), and provide a tool – the Behavior Lab – which is similar to existing haptic icon and waveform design tools [4,24].

The second category of interactive haptic devices are multi-dimensional (usually > 3 DOF) closed-loop devices such as Immersion's PHANToM [14] or Novint's Falcon [17]. Programming these haptic interfaces can require significant expertise [15], and its difficulty has often been identified as an obstacle in the popularization of this type of interfaces [20,22]. Multiple APIs exist that enable higher-level programming of these haptic devices. Many are low-level and provide access to specific haptic devices (e.g., Novint's HDAL [18]), although there are also ongoing efforts to provide generalized haptic libraries that work on several devices (e.g., H3D [23], ProtoHaptic [5], and OpenHaptics [8]; also see survey in [10]). Some introduced drag-and-drop interfaces for allowing non-programmers to customize haptic systems (e.g., [22]). In general, these APIs assume a

model of interaction based on the simulation of physical objects and surfaces, often directly linked to 3D virtual reality models.

The HTP is closer to these devices than to the first group because it enables bi-directional communication and provides multi-dimensional haptic channels. However, the HAPTICTOUCH toolkit was designed to avoid the programming complexity inherent to 3D models and the simulation of physical models. Our 2D GUI-based programming model also enables a more flexible relationship between haptic interaction and graphical components because it is not constrained to physical behavior such as objects moving in 3D and colliding with each other. This corresponds to the needs of flexible prototyping tools advocated in [15,16]. Our approach is most similar to tools such as HITPROTO [20], which make use of haptics to represent abstract data or *haptifications*. In a way, our approach takes relevant features from both groups of haptic devices, yet avoiding high development complexity.

Few devices exist that enable haptic feedback on tabletop surfaces; most notably, shape displays (e.g., Lumen [21] and Relief [12]), tangible actuators (e.g., Madgets [25]), or ferro-magnetic tactile feedback devices (e.g., MudPad [9]). As none of these haptic devices provide any development support or prototyping tools yet, our concepts of the HAPTICTOUCH toolkit design might be applied these and similar platforms in order to facilitate haptics exploration.

API DESIGN OBJECTIVES AND DESIGN DECISIONS

When designing the HAPTICTOUCH API we set out to achieve two major goals:

(1) we want to enable the *creation of a wide range of application prototypes and sketches* for haptic tabletop environments (currently the HTP) without requiring programmers to understand 3D models or the physics of objects and materials, and

(2) we want to provide a *simple programming interface* for haptics development.

These goals can be considered as specific instantiations of Olsen's general goals for interface systems research [19].

Based on these goals we designed an API architecture that is different from existing haptic API approaches reviewed in the previous section. Our design principles are summarized as follows:

GUI Model. We base our architecture on a GUI model rather than the simulation of three dimensional physical behaviors. This leads to a more abstract and generic paradigm with higher flexibility, which corresponds to our first generic goal. Furthermore, not only are programmers more familiar with the GUI paradigm, but it also seems to better fit the two-dimensional constraints of surface based systems.

3-Layered Architecture. We structure our architectural design in three abstraction levels. The *raw layer* provides access to physical properties of the device. The *behavior layer* enables haptic interactions packaged as behaviors. The *graphical layer* allows creating two dimensional objects (e.g., shapes, images) extending traditional GUI concepts to the control of haptics. Programmers can use select layers to access functionality required for different projects, but each layer can be learnt separately and used independently.

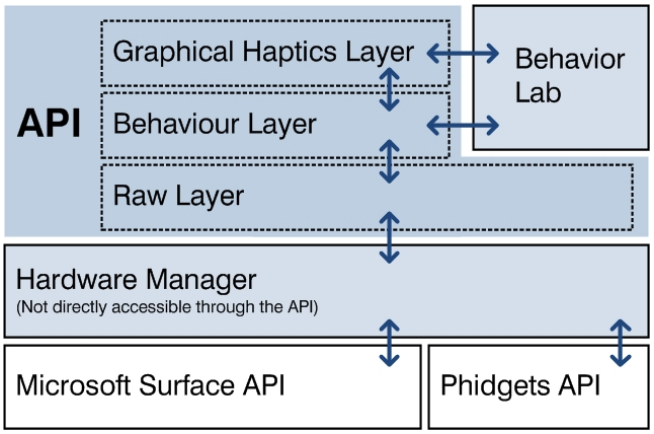

Figure 3. The HAPTICTOUCH toolkit's architecture.

Although layering is not uncommon in the design of SDKs and APIs, many tools decide to hide the details related to the device (e.g., HITPROTO [20]). This has two consequences. First, due to the higher level of abstraction, the creation of new haptic interaction techniques can be constrained. Second, different programmers take different approaches to a problem initially: bottom up and top down. We divided our API design into three levels of abstraction to enable both styles.

ARCHITECTURE AND PROGRAMMING INTERFACE

Each layer of the HAPTICTOUCH Toolkit is successively closer to the underlying hardware. We expect the needs of most programmers require programming only at the highest abstract level, as this will ease programming effort while still being highly expressive [19]. For flexibility, programmers can work at lower layers when more direct or nuanced control is required with only slightly more effort [19].

Figure 3 shows the toolkit's structure. It contains a *hardware manager* layer which accesses underlying commercial APIs, three *API layers* accessible to programmers, and the *behavior lab* utility. The toolkit's layers build on top of each other, with higher levels using lower level components. In this section, we explain the architecture by reviewing the function and main components of each layer.

Hardware Manager

The core of the HAPTICTOUCH toolkit is the *hardware manager*. It merges higher-level services from commercial APIs. Specifically, it manages the Phidget [6] servos and sensors that comprise the HTP, and the interaction between the Microsoft Surface SDK and the HTP. The manager takes care of hardware calibrations of each individual HTP's, and abstracts the HTP's built-in Phidget components to control its *height*, *friction* and *pressure*, where values are normalized in a scale from 0 to 1. We use the Mi-

crosoft Surface tag events to gather information about a HTP's location and orientation and detect when a puck is placed down, moved or lifted up. Combining all abstracted information, an HTP is characterized by *pressure*, *friction*, *height*, *location*, *orientation* and *id* (i.e., the tag's id). The hardware manager contains all HTP information and their states at any time. The hardware manager is not accessible to programmers directly from the API, but several attributes of the hardware can be modified through human-readable XML configuration files.

The hardware manager module can be substituted to recognize other hardware. For example, we are replacing Phidgets with Arduino [1] so that our HTPs can work wirelessly, which only requires updating the hardware manager.

Raw Layer: Access to Device-Independent Hardware

This first layer is intended for experts: it allows direct control of the HTP physical properties and actuators. To avoid conflict with higher layers, users have to enable *manual control* in the `HTPManager`. Programmers can then access the applied *pressure* returned by the pressure sensor, and determine the *height* and *friction* values of the HTP. This is all done in a device-independent manner; no knowledge of the underlying Phidget hardware is required. As mentioned before, friction, height and pressure are represented within a range from 0 to 1 rather than (say) the angle of the servo motor as implemented in the Phidget API.

As an example, we illustrate how a programmer can create a basic haptic loop that simulates malleability. It changes the height of the HTP's rod to vary with the pressure the person applies to it. That is, the more pressure applied, the lower the position (height) of the rod. We first initialize the HTP manager (line 1), and then retrieve a specific HTP identified by the id of the fiduciary marker attached to it (here: 0xEF; line 3). Then we enable manual control (line 4). The remaining lines implement an infinite loop in which the height of the puck drops linearly according to the applied pressure (lines 5-6).

```
1. HTPManager manager = HTPManager.Instance;
2. // Retrieve HTP
3. HTP htp = manager.GetHTP(0xEF);
4. manager.ManualControl = true;
5. while(true)
6.     htp.Height = 1.0 – htp.Pressure;
```

Behavior Layer: Adding Haptic Behaviors

The Raw Layer requires low-level programming and does not minimize the complexity of creating and managing haptic loops. To facilitate haptics programming, the Behavior Layer adds a higher level abstraction as a set of pre-defined haptic behaviors. We define a *haptic behavior* as the change in height or friction as a function and combination of: the current height, friction, pressure, and external factors, such as the puck's location and orientation, and time.

Using Basic Haptic Behaviors

As a starting point for application developers, we included a pre-defined set of behaviors that proved worthy in practice. Each behavior has a specific parameter ranging from 0 to 1 that modifies the behaviors further. These behaviors are:

a) ***Softness:*** change of height or friction depending on applied pressure. Adjusting this behavior makes an 'object' under the puck feel softer (closer to 1) or harder (closer to 0). That is, a hard object will offer considerable resistance when a person presses on the HTP's rod, while a soft object will feel mushy and yielding.
b) ***Breakiness:*** a non-linear relationship between pressure and height produces an irregular tactile sensation. A configurable number of breakpoints are placed along a linear softness response. At these points, the height does not change within a certain range of pressure, contributing to a tactile sensation that resembles: a dual-press button (as in photo-camera shutter buttons – with a single break point), poking a finger on sand (with many breaking points), and coarse salt (with fewer points).
c) ***Oscillation:*** change of height or friction depending on time. The behavior follows a sinusoidal change of height with an adjustable frequency (with 1 being maximum frequency). Oscillation can be used to notify the user, or to simulate tactile noise.
d) ***Static:*** change of height or friction to a specific predefined value (1 representing maximum height).

These behaviors represent only a small set of the possibilities. For example, each behavior is also invertible, resulting in interesting alternatives. Inverted softness provides a "resistant" behavior where the height of the puck increases with the pressure applied.

These behaviors can be used individually, or can be easily combined (i.e., stacked) to allow for even more expressiveness. If combined behaviors are used, programmers can specify *weights* for each behavior that defines the proportion of the movement of the rod used for that behavior relative to others. The weight of each haptic behavior is initially set to 1. Different weights and the order in which the behaviors are added create a wide variety of expressive outputs. Haptic behaviors are assigned to a list, either to the set of behaviors belonging to the height rod, or the ones belonging to the brake. The following example shows how to add two haptic behaviors, each with different weights.

```
1. HTPManager manager = HTPManager.Instance;
2. HTP htp = manager.GetHTP(0xEF);
3. // Add the first behavior
4. h.AddHeightBehavior(new SoftnessBehavior(0.5));
5. OscillationBehavior osc
       = new OscillationBehavior(0.03);
6. // Give twice the weight for the oscillation
7. osc.Weight = 2;
8. h.AddHeightBehavior(osc);
```

In the example, we set up the HTP as in the previous example (lines 1-2), add a softness behavior in line 4 and an oscillation with a weight of 2 in lines 5 and 7. This creates 3 partitions of the rod range, with 2 assigned to oscillation and 1 assigned to softness. Both behaviours also need to be

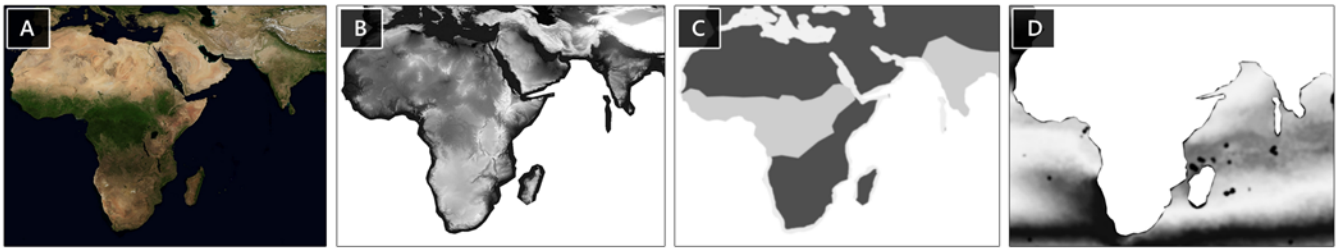

Figure 4. Haptic Image and the corresponding Behavior Mappings: a) visible haptic image, b) image mapping for static behavior, c) image mapping for softness behavior, and d) image mapping for oscillation behavior.

created (with a parameter indicating their strength – lines 4 and 5), and added to the height rod (lines 4-8).

The Behavior Lab

To let developers explore various combinations of behaviors and weights, we created the Behavior Lab tool (see Figure 1). It serves as an experimental test bed for interactively examining haptic behaviors without requiring any programming. Users can experiment with the previously mentioned basic set of haptic behaviors. They can change haptic settings (e.g., softness, oscillation speed, number of breakpoints, static values), and combine behaviors with different weights. The behaviors can be tested and felt in real time as they are created, and then its settings saved for use in an application via code.

Graphical Haptics Layer

The highest level of abstraction defines haptic shapes, images, and widgets. These allow for rapid programming of haptics in a GUI-like fashion. The general idea is to complement existing shapes, images and graphical widgets with a set of haptic behaviors assigned by the programmer. This level also incorporates events within visuals (fired when an HTP is *down*, *changed* or *up*), which allows further customization of currently embedded visuals. We provide three haptic visual classes: *haptic shapes*, *haptic images* and *haptic buttons*, which we describe in the following subsections together. We then introduce *haptic transparency*, which allows combining them.

Haptic Shapes

Haptic shapes are basic graphical shapes (e.g., rectangles, ellipses, or paths). The programmer can associate one or more behaviors to the shape. When the shape is on the screen, an HTP anywhere within its bounds will reproduce the shape's assigned haptic behavior. For example, a programmer can create an ellipse that produces oscillations when the HTP is within its bounds, and a line that causes friction when the HTP is moved across.

To illustrate how haptic shapes are used consider a developer that wants a rectangle on the screen to cause the puck to resist horizontal movement in an oscillatory manner equivalent to shuddering. The developer creates a *rectangle* and adds it to the window (not shown), creates a *haptic shape* that encapsulates the rectangle (line 1), and registers that shape with the `HTPManager` (line 2). He then assigns the oscillation and the resistance (softness) to friction within that shape (lines 4-5). The rectangle will now produce those responses when the HTP passes over it, even if that rectangle is scaled, rotated or translated at runtime.

```
1. HTPShape shape = new HTPShape(rectangle);
2. this.manager.RegisterWidget(shape);
3. // Add behaviors to the shape
4. hapticShape.AddFrictionBehavior(
       new OscillationBehavior(0.5));
5. hapticShape.AddFrictionBehavior(
       new SoftnessBehavior(1.0));
```

Haptic Images

Haptic images contain one or several image maps, some of which can be invisible to the user, that defines haptic behaviors. The main difference between shapes and images is that, in images, the haptic behavior parameter can change according to the particular grey-level pixel values of the image where the HTP points. We refer to this as *spatial behavior mapping*. The programmer constructs a grey-scale image (e.g., using a bitmap editor), where the level of each pixel determines the parameter of the haptic behavior produced from it. Using these pixel-based haptic image maps allows creating more sophisticated and complex mappings of haptic behaviors. For example, Figure 4 illustrates using image maps for haptic behavior mapping. Image 4a (a satellite image) is normal graphics visible to users. The other images, which are invisible to the user, represent actual haptic mappings. Image 4b encodes a relief as a static behavior. 4c represents the softness of different terrains, and 4d the ocean depth as different oscillation speeds. Because images are stacked, behaviors are automatically combined in a location-dependent manner.

In the example below, the programmer uses the images shown in Figure 4a (i.e., the visible image), and Figure 4b (i.e., relief image map) to map the height of the HTP's rod as a function of the terrain relief. The developer uses a set of pre-existing or self-designed images and adds them to the canvas. Three main objects are created, a *HTPImage*, which receives the visible image as its construction parameter (line 1), a *StaticBehavior* (line 4), and a *SpatialBehaviorMapping*, which belongs to the static behavior, and is passed the *reliefImage* as its constructor parameter (line 5). The image is finally associated to the height produced by the mapping (line 6). As with all graphical haptics, registration with the HTP Manager is required (line 2). Similarly,

the programmer can add more behaviors to an image for height or friction (e.g. Figures 4c and d).

```
1. HTPImage hapticImage = new HTPImage(mapImage);
2. this.manager.RegisterWidget(hapticImage);
3. // Create, modify, and add the behavior
4. StaticBehavior behavior = new IntensityBehavior();
5. behavior.IntensityMapping =
       new SpatialBehaviorMapping(reliefImage);
6. image.AddHeightMapping(behavior,
       behavior.IntensityMapping);
```

Haptic Buttons

Haptic buttons are graphical widgets with pre-assigned haptic behaviors. While we foresee many haptic widgets, we have currently only programmed a haptic button whose haptics depends on how it is pressed. A button implements three haptic images representing the three basic states (i.e., *inactive*, *hover* and *pressed*) with each of them having individual haptic behavior mappings. The button state transitions from *normal* to *highlighted* (and vice versa) occur when the HTP enters or leaves the button. The pressure sensor is then used to determine whether a button has been pressed or released. By applying certain haptic behaviors, a button may need more pressure to activate it (e.g., signaling the user of a critical operation). Likewise, if the action is impossible at the moment, the button would not allow itself to be pressed. Buttons allow developers to subscribe to events when a button is pressed or released. A pressure value to go from *pressed* to *released* and vice versa is initially set to the default value of 0.5, but can be adjusted during runtime.

To create a customizable haptic button, the developer first creates three haptic images for each of the button's states and their corresponding behaviors (here: *inactive*, *hover* and *pressed*) as described previously. She then creates the haptic button with these as parameters (line 1), and registers it with the `HTPManager` (line 2). Finally, she registers event handlers to the button (lines 4-5). Of course, pre-packaged versions of button behaviors can be supplied, where the programmer does not have to supply these image maps.

```
1. HTPButton button =
       new HTPButton(inactive, hover, pressed);
2. this.manager.RegisterWidget(button);
3. // add events
4. button.ButtonPressed +=
       new BeingPressedHandler(BecomesPressed);
5. button.ButtonReleased +=
       new BeingPressedHandler(BecomesReleased);
```

Haptic Transparency

Overlapping graphical widgets can be simultaneously visible on a given area through transparency. Similarly, several graphical haptics can share the same space on the tabletop surface and still contribute to the haptic behavior through what we call *haptic transparency*. This mechanism allows programming complex behaviors by using a combination of haptic shapes, hatpic images and haptic buttons atop each other. Haptic transparency is implemented as an extension of the haptic behavior stacking mechanism in the behavior layer.

PRELIMINARY EXPLORATION

We conducted a preliminary exploration with three developers (one undergraduate and one graduate student, plus one developer from industry) that used our toolkit. While this does not represent a formal evaluation, it allowed us to see whether programmers understand the abstractions in our toolkit, and if they are able to create simple haptic applications in a short amount of time.

Participants attended a one-hour tutorial workshop. None had worked with the toolkit before. We demonstrated the toolkit, and illustrated code samples required to access each level. We then asked them to perform a series of pre-defined tasks. Later, participants were asked to create an application of their own design using the HAPTICTOUCH toolkit entirely on their own, where we limited them to 3 hours of programming. This section summarizes our results: the findings from the pre-defined tasks, the applications they created, and overall observations we made.

Simple Tasks to Explore the Abstraction Levels

Participants were given six pre-defined tasks, each an exercise to explore the higher two layers. We did not test raw layer programming, as we consider that to be appropriate for only experts. In the first task, participants were asked to use the HTP's *up*, *down* and *changed* events to make the HTP's softness vary linearly as a function of moving left or right across the screen (e.g., soft at the left, hard at the right side of the screen). On average, participants needed ~15 minutes to complete this first task, most which we ascribe to the programmers familiarizing themselves with the programming environment and the HTP in particular.

For the second task, participants were asked to create two haptic shapes (the top one with opacity of 0.5): one shape with a breakiness behavior, the other one with an oscillation behavior. Shapes would intersect at a certain point to combine these behaviors. This task took participants on average ~7 minutes. In task three, participants were provided with two images: a visual image and one representing a haptic mapping for that image. Their task was to program these to create a haptic image. They did this in ~ 5.3 minutes.

The next three tasks were somewhat more complex. They had to simulate an invisible cloth on the table which contained a hidden cube. They were then asked to recreate this case using the HTP through three different methods. We observed the different levels they chose, how long they took and what their preference is. All participants chose the same methods, but in a different order according to their preferences: a shape with opacity 0 (average: 6.2 min); an invisible image (average: 3.3 minutes); and haptic events (average: 3.1 minutes). We observed that the reasons for the rather long time for shapes were that developers forgot (1) to give a fill to the shape causing it not to fire any events, or (2) that the shape can have a transparent fill. These issues

Figure 5. Applications created in our preliminary exploration: a) Haptic Vet, b) Ouija Board, and c) Haptic Music.

relate more to unfamiliarity with the nuances of graphical programming.

Overall, we noticed (as expected) that participants completed their tasks more quickly when working with pre-defined shapes and images *vs.* working at the lower layer.

Example Applications

Participants created diverse and rich haptic applications. By using the toolkit, developers were able spending a significant amount of time on the visual appearance of their application rather than coding haptic behaviors from scratch.

Haptic Vet

In the Haptic Vet application (Figure 5a), the goal is to find the one or more areas where a dog is injured within a 30 second period. People can "scan" the displayed image of a dog by moving the HTP over it, where injured parts makes them feel an oscillation haptic feedback. When an injury is discovered, players press on the HTP to heal the dog, after which the dog's facial expression changes. Healing injuries increases the score of the player until all injuries are healed.

To create this application, the participant made use of a haptic image with assigned behavior mappings. These mappings simulated the dog's texture, the oscillation behavior of injured areas, and an additional soft texture for the injury. The participant used events to determine the pressure applied in different locations, as well as to start the game (triggered by placing down an HTP). The participant stated that the idea came from a similar childhood board game.

Ouija Board

In the Ouija Board application (Figure 5b), a person can reveal messages of the board (i.e., a sequence of letters). As the user moves over the characters of the board, certain letters vibrate (via changing the oscillation speed) when the HTP is moved over them. The player then presses the HTP to select the letter; and moves on to discover the next letter. Once all letters are correctly selected, he or she can spell the answer to the question by rearranging the letters.

The developer of this application made use of the behavior layer events to derive the HTP's location and set the oscillation frequency, as well as the softness values of the HTP's height rod. Alternatively, it could have been implemented as a haptic shape, e.g., by moving a haptic shape with an oscillating behavior under the next level. Similar to the developer of the first application, the participant based his idea on an existing board game.

Haptic Music

Haptic Music is an application that used a variety of textures to create a new approach to music (Figure 5c). It introduces various musical instruments, each of them having a different haptic texture to best represent that particular instrument. Cymbals and tuba are represented by softness, and require different amounts of pressure to play them. Softness corresponds to how easy the instruments are to be played in real life. The piano had varying pressure levels, which depended on the key underneath the HTP. Maracas used friction and the natural sound made by the servo motors when being pressed. A center box serves as a haptic metronome: when the player places the HTP atop different tempo values, the rod's height is varied to let the player feel that tempo.

The participant creating this application only used haptic shapes. Each shape was filled with a transparent brush and had associated haptic behaviors matched to the various instruments and/or parts of that instrument. The participant stated that he preferred haptic shapes as they are easy to track in the program.

Observed Problems

While participants were able to create rich applications in a short amount of time and with very little training, we did note several places that caused confusion:

1. Haptics and haptic behaviors are a new domain for most programmers. Some of our participants initially struggled to understand haptic behaviors, and how they are controlled by their 0 to 1 parameter range.
2. Individuals did not fully understand the distribution of weights for combined behaviors, nor the importance of the order when combining them. They interpreted weights as percentages, rather than proportions of the rod height used to implement a particular behavior.

Both problems likely occur because people are trying to associate a physical phenomenon (haptics) with an abstraction (programming). We believe our Behavior Lab will mitigate both of these problems, where people can directly experiment with and feel the effects of adjusting parameters as well as how they are combined.

CONCLUSIONS AND FUTURE WORK

Haptics can be useful in many situations as they introduce an additional sensorial layer to interaction. They can reduce the cognitive load and the amount of visual and auditory cues [3]. They enhance the interaction experience by making it more realistic and close to the physical world. They can be used to make current computer technology accessible to people with visual disabilities.

With the haptic puck and our HAPTICTOUCH toolkit, we facilitate programming of simple haptic interfaces for surfaces by providing a DIY haptic device [13] and a set of pre-programmed building blocks (this paper). We introduced meaningful abstractions and concepts to make haptics accessible to developers. Based on our initial exploration with three developers, we assume that our toolkit can help in the exploration of the design space of haptics as it allows a new level of expressiveness for developers.

We layered the toolkit to promote flexibility and expressivity while still minimizing the programming burden for common tasks. The raw layer allows unconstrained hardware access. Our behavior layer introduces contact events for HTP devices, and contributes pre-defined haptic behaviors. Our graphical haptics layer uses shapes, images and widgets to associate haptic behaviors to graphical objects. The three layers give developers the possibility to choose the layer most suitable for the particular haptics programming task at hand. These layers are augmented with our Behaviour Lab, which lets developers interactively explore, combine and feel diverse haptic behaviors before writing any code.

While the haptic puck is a limited input and output device, it serves as a good starting point that will hopefully allow a wider set of researchers to explore haptics until more affordable and richer technology becomes available. The toolkit itself points the way to the future. While designed for the haptic puck, its notion of encapsulating haptics as combinable behaviors is a new contribution. Also new is the use of shapes and image maps to specify very fine-grained haptic behaviors. Our Behavior Lab also points the way for how we can let programmers explore and 'feel' available forms of haptic feedback even before they write a single line of code. These ideas can be applied to other haptic devices that go beyond the haptic puck.

ACKNOWLEDGMENTS

This research is partially funded by the AITF/NSERC/ SMART Chair in Interactive Technologies, Alberta Innovates Technology Futures, NSERC, and SMART Technologies Inc.

REFERENCES

1. Arduino. Open-source Electronics Prototyping Platform. http://www.arduino.cc (access 9/2/2011).
2. Brewster, S. and Brown, L.M. Tactons: structured tactile messages for non-visual information display. *Proc. of AIUC '04*, Australian Comp. Soc., (2004), 15–23.
3. Chang, A., Gouldstone, J., Zigelbaum, J., and Ishii, H. Pragmatic haptics. *Proc. of TEI*, ACM (2008).
4. Enriquez, M.J. and MacLean, K.E. The hapticon editor: a tool in support of haptic communication research. *Proc. of HAPTICS '03*. IEEE (2003).
5. Forrest, N. and Wall, S. ProtoHaptic: Facilitating Rapid Interactive Prototyping of Haptic Environments. *Proc. of HAID '06*, Springer (2006), 18–21.
6. Greenberg, S. and Fitchett, C. Phidgets: easy development of physical interfaces through physical widgets. *Proc. of UIST '01*, ACM (2001), 209–218.
7. Immersion. MOTIV Development Platform. http://www.immersion.com/products/motiv (access 9/2/2011).
8. Itkowitz, B., Handley, J., and Zhu, W. The OpenHaptics Toolkit: A Library for Adding 3D Touch Navigation and Haptics to Graphics Applications. *Proc. of WHC'05*, (2005).
9. Jansen, Y., Karrer, T., and Borchers, J. MudPad: localized tactile feedback on touch surfaces. *Adj. proc. of UIST '10*, ACM (2010), 385–386.
10. Kadlecek, P. A Practical Survey of Haptic APIs. *BSc Thesis*, Charles University in Prague (2010).
11. Klemmer, S.R., Li, J., Lin, J., and Landay, J.A. Papier-Mache. *Proc. of CHI '04*, ACM (2004), 399-406.
12. Leithinger, D. and Ishii, H. Relief: a scalable actuated shape display. *Proc. of TEI*, ACM (2010), 221–222.
13. Marquardt, N., Nacenta, M.A., Young, J.E., Carpendale, S., Greenberg, S., and Sharlin, E. The Haptic Tabletop Puck: tactile feedback for interactive tabletops. *Proc. of ITS '09*, ACM (2009), 85–92.
14. Massie, T. and Salisbury, J.K. The PHANTOM Haptic Interface: A Device for Probing Virtual Objects. *Proc. of ASME Symp. on Haptic Interfaces for Virt. Environments and Teleoperator Systems*, (1994).
15. Moussette, C. and Banks, R. Designing through making: exploring the simple haptic design space. *Proc. of TEI '11*, ACM (2011), 279–282.
16. Moussette, C. Feeling it: sketching haptic interfaces. *Proc. of SIDeR '09*, (2009), 63.
17. Novint Technol. Inc. Falcon. http://www.novint.com/~ index.php/products/novintfalcon (access 9/2/2011).
18. Novint Technol. Inc. Haptic Device Abstraction Layer (HDAL). http://www.novint.com/index.php/support/~ downloads (access 9/2/2011).
19. Olsen, J. Evaluating user interface systems research. *Proc. of UIST '07*, ACM (2007), 251–258.
20. Panëels, S.A., Roberts, J.C., and Rodgers, P.J. HITPROTO: a tool for the rapid prototyping of haptic interactions for haptic data visualization. *Haptics Symposium*, IEEE (2010), 261–268.
21. Poupyrev, I., Nashida, T., Maruyama, S., Rekimoto, J., and Yamaji, Y. Lumen. *SIGGRAPH '04 Emerging technologies*, ACM (2004), 17.
22. Rossi, M., Tuer, K., and Wang, D. A new design paradigm for the rapid development of haptic and telehaptic applications. *Proc. of CCA 2005*, IEEE (2005).

23. SenseGraphics AB. H3DAPI. http://www.h3d.org (access 9/2/2011).
24. Swindells, C., Maksakov, E., and MacLean, K.E. The Role of Prototyping Tools for Haptic Behavior Design. *Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, IEEE (2006), 25.
25. Weiss, M., Schwarz, F., Jakubowski, S., and Borchers, J. Madgets: actuating widgets on interactive tabletops. *Proc. of UIST '10*, ACM (2010), 293–302.