Portico: Tangible Interaction on and around a Tablet

Daniel Avrahami[1,2], Jacob O. Wobbrock[2] and Shahram Izadi[3]

[1]Intel
2200 Mission College Blvd.
Santa Clara, CA 95054-1549

[2]The Information School | DUB Group
University of Washington
Seattle, WA 98195-2840

[3]Microsoft Research
7 J J Thomson Avenue
Cambridge, UK

daniel.avrahami@intel.com, wobbrock@uw.edu, shahrami@microsoft.com

ABSTRACT

We present *Portico*, a portable system for enabling tangible interaction on and around tablet computers. Two cameras on small foldable arms are positioned above the display to recognize a variety of physical objects placed on or around the tablet. These cameras have a larger field-of-view than the screen, allowing Portico to extend interaction significantly beyond the tablet itself. Our prototype, which uses a 12" tablet, delivers an interaction space six times the size of the tablet screen. Portico thus allows tablets to extend both their sensing capabilities and interaction space without sacrificing portability. We describe the design of our system and present a number of applications that demonstrate Portico's unique capability to track objects. We focus on a number of fun applications that demonstrate how such a device can be used as a low-cost way to create personal surface computing experiences. Finally, we discuss the challenges in supporting tangible interaction beyond the screen and describe possible mechanisms for overcoming them.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces—*graphical user interfaces*.

General terms: Design, Human Factors.

Keywords: Tangible, TUI, surface, tablet, portable.

INTRODUCTION

Surface and tabletop computing has been an important area in HCI research for over two decades. Interacting with surface computers allows users to directly manipulate digital elements through touch, and often allows interaction with and through physical objects set directly on the display. Surface computers, however, are typically large, expensive, and neither personal nor portable. In this paper we present *Portico*, a system that enables tangible interaction in a new inexpensive portable form-factor, while still delivering a large interaction space to support physical interaction with objects, touch, and gesture. Portico presents a possibility of enabling low-cost personal surface computers for many users.

UIST'11, October 16–19, 2011, Santa Barbara, California, USA.

Figure 1. The Portico system in use. Two cameras track objects on the screen and surrounding surface. In this application, a toy zebra is tracked.

The majority of tabletop research and development has focused on the use of surface computing to support co-located groups, a natural outcome of existing tabletop computers resembling everyday tables. However, tangible interaction for personal or individual use is not yet widespread, largely due to the size and cost of today's tabletop computers. As presented by Beckwith *et al.* [3], affordable personal tangible interaction may have particular promise for education. Recent products such as Apple's iPad have started a convergence of portable tablet computers with tabletop computing through the introduction of multi-touch technology to tablets. Tablet computers, being portable and flat and supporting touch input, are prime candidates for enabling personal tangible interaction. However, with the exception of ThinSight [8], which, through the use of sensors embedded in the back of the display is able to recognize hands and objects placed on the screen, current tablet computers lack support for interaction with physical objects. A key challenge is overcoming the constraints imposed by the boundaries of the tablet screen, since, for many tangible applications, small screen real-estate proves prohibitive.

Portico overcomes the challenge of a limited interaction space by using two cameras on small foldable arms that provide a large field-of-view. Portico is thus able to recognize and react to objects manipulated not only on the tablet screen, but also on the surface beyond the tablet screen. In our prototype, Portico provides an interactive

space six times the size of a 12" tablet screen, equivalent to the surface area of a 28" screen—a considerable interaction space for a small, lightweight, low-cost device. Designed to be *portable* rather than *mobile*, we envision that Portico will be carried around like a regular tablet but used for tangible interaction when it is on a tabletop, countertop, classroom desk, or the even the living room floor.

In the remainder of this paper, we describe related work, our system's hardware and software, and a number of example applications that take advantage of Portico's capabilities. We then discuss the benefits and limitations of Portico's design.

RELATED WORK

Connecting the physical and virtual has been a long sought-after goal in computing research. The earliest major work in this area was the DigitalDesk by Wellner in 1993 [23]. Ulmer and Ishii [10,22] carried this vision forward, inspiring a long and creative line of research. Much of the work on tangible interaction focused on the manipulation of tangible objects as controls for digital elements, often referred to as Tangible User Interfaces, or TUIs. A nice review of work on TUIs is provided in the literature [18]. With its strong link to the physical world, tangible interaction is typically done in the context of a tabletop or other surface computer that is large, often expensive, and fixed in its environment.

With Portico, we demonstrate how tangible computing can be supported in an affordable and portable form-factor without severely limiting the space available for physical interaction. This vision of a simple, portable device that can be carried around and that can quickly turn into a surface computer is similar to PlayAnywhere [24] and Bonfire [12], both of which create portable tabletop systems with the use of standalone projectors and embedded projectors. While Bonfire allows the laptop to "spill over" to the tabletop through the use of a projector, Portico achieves much of the same benefits but without requiring a projector, making it substantially cheaper, and requiring significantly less power. Portico can also use the tablet's touch-screen for touch and gesture, which is more robust than vision. Also, with Portico, the primary display is horizontal in the same plane as the tangible objects, whereas Bonfire's primary display was vertical and above its peripheral interactive surfaces. This difference allows Portico to take advantage of physical objects interacting with high-resolution output for the user's primary tasks.

While the key characteristic of Portico—a large interaction space in a portable form-factor—is a novel contribution of our work, the basic use of a tablet as a horizontal display for interaction with objects on top of the screen has been demonstrated by prior work. Edge and Blackwell [5], for example, used a tablet computer on which users could interact with objects as a fixed peripheral display to a workstation. As they point out, however, in their case, the tablet easily could have been replaced with a screen embedded in the desk on which the workstation is located.

The Collaborative Slate (C-Slate) from Izadi *et al*. [11] was designed to support remote collaboration. Not designed to be portable, C-slate uses a horizontally mounted 21" tablet combined with a down-facing stereo camera, a vertical display and webcam, and supports tangible and multi-touch interaction on the tablet screen. C-Slate does, however, include abilities to view a collaborator's face on the vertical display, and to view the collaborator's hands on the tablet screen.

Finally, Hodges *et al*.'s ThinSight [8] uses optical IR sensors embedded behind an LCD to detect fingers and hands on the surface. Although the authors mention it primarily as a future possibility, their system can also detect the base of objects placed on the screen, with potential uses for tangible interaction. Although ThinSight does not go beyond the confines of its screen and sees objects only from underneath, our work was nonetheless inspired by ThinSight and its exploration of tangible interaction on small form-factor screens.

Interacting Beyond the Screen

A primary goal of Portico is to provide a large interaction space despite the limited screen area provided by a tablet computer. The physical design of our system with two cameras enables it to view and respond to users' interactions with objects on the surface *surrounding* the tablet. Thus, our system's input space (what the system sees) is significantly larger than its output space (limited to the screen boundaries). Prior work on mobile interaction explored the ability to manipulate on-screen (digital) content by interacting in the space around the screen. With mobile devices, screen real-estate often makes it impossible to accommodate a user's hands or even fingers without obstructing the digital content on the screen. SideSight [4] was designed for mobile devices and uses infrared (IR) proximity sensors embedded on each side of a mobile device that detect the position of fingers on the surface around the device. Abracadabra [7] was designed to allow finger interaction with graphical interfaces on very small displays using a combination of a magnetometer in the device and a magnet on a ring worn by the user to detect the finger's position. Portico relates to these prior efforts and uses vision to track objects and users' hands outside the device screen area.

THE DESIGN OF PORTICO

Our system comprises both hardware and software components: a custom designed hardware attachment, a vision system used for object detection, and an output system allowing Portico to visualize objects that are on the screen or on the table around it.

Hardware Design

Portico comprises a tablet computer and a pair of standard cameras that are attached to the tablet via custom designed fixtures. These cameras look down at the tablet screen and its surrounding surface. We modeled the fixtures in CAD such that they clasp the sides of the tablet screen (rather than the base, so they do not block any ports). For our

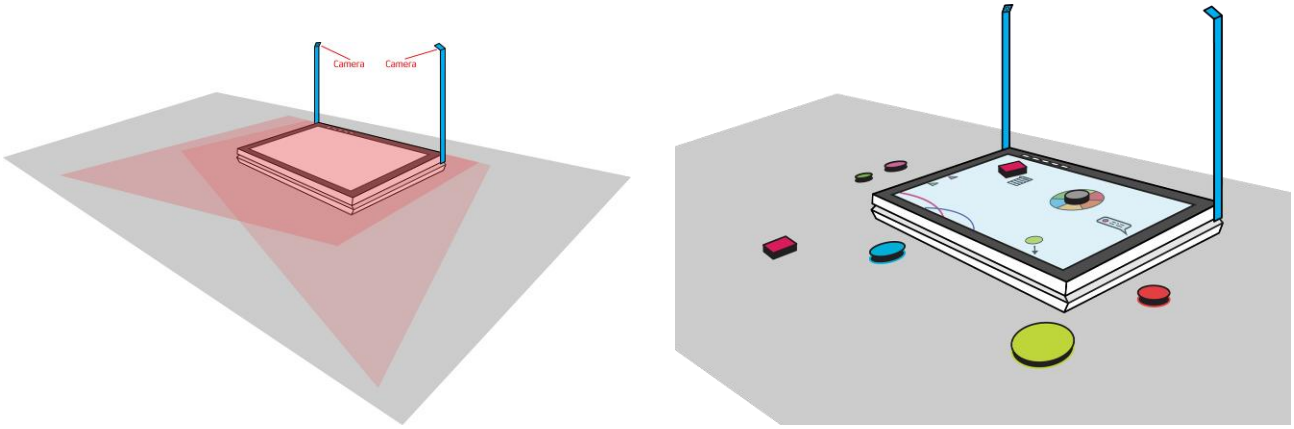

Figure 2. Portico system illustrations: The unified area viewable by the two cameras (left), and the system reacting to object on and around the tablet (right).

prototype, we printed the fixtures out of plastic using a Dimension Elite 3D printer. While printed plastic was satisfactory for prototyping the fixtures, we believe that stronger plastic or even aluminum is preferred. To provide additional robustness to breaking, flexible hinges similar to those used in eye glasses could allow Portico's arms to be bent outwards. Such a design is particularly important when Portico is used on the floor, or used by children playing a game, *e.g.*, our Penalty Shootout application (described below). For our prototype, we used a 12" Dell Latitude XT convertible tablet that supports both pen and finger multi-touch input. We also used a pair of Logitech Webcam Pro 9000 cameras extracted from their original housings. As shown in Figure 1, each camera is mounted at the end of a foldable arm attached to each side of the screen. We designed our system such that when folded down, the arms and cameras are flush against the tablet and do not interfere with normal use.

When the arms are raised, the cameras are bent down using a single rotation hinge to allow them to see the tablet screen and the tabletop. Figure 2a shows an illustration of the unified area viewable by the two cameras. Figure 2b illustrates the system reacting to objects on and around the screen. Finally, to avoid the system confusing digital elements drawn on the screen with physical objects, we applied linear polarizing filters to the screen and cameras. (Note that while LCDs are linearly polarized, in many tablets the touch-sensitive element diffuses the light, requiring another polarizing layer.)

Computer Vision System

We implemented a computer vision system to allow applications to support interactions with objects on the tablet and the surface around it. The vision system was written predominantly in Python and uses Intel's OpenCV library. We now describe the vision system in some detail to enable readers to replicate our system.

As shown in Figure 3 (next page), the vision system was implemented in a threaded, hierarchical structure. At the core of the vision system is the *Camera* Module, which is responsible for retrieving frames from a single camera, performing an optional background subtraction step, and gathering detected objects from different *View* Modules. Each View Module is responsible for manipulating raw camera frames to produce a specific simulated view, and contains a set of object recognition classifiers that operate on the simulated view. These views are useful since, as can be seen in Figure 4a, in order for the camera to see the tablet screen and the surrounding surface (and still be folded down when not in use), the cameras must view the world from a very oblique angle. Our system uses two Camera Modules, one for each camera. Both Camera Modules and all View Modules are threaded to allow the vision system elements to operate concurrently.

A single pass of the vision system consists of these steps:

1. *Grabbing camera frames*. Each Camera Module is connected to a single camera device. Upon request from the Perception Manager, a Camera Module retrieves a new frame from the camera and passes it to each of its View Modules.

2. *Optional background-subtraction*. Portico provides a background subtraction capability and offers it as an optional step in the perception process prior to view generation and object recognition. We use a Gaussian Mixture Models approach for background subtraction [20], with a dedicated model for each camera. When applied, a background/foreground mask is produced and is passed to each View Module along with the raw camera frame.

3. *Producing simulated camera views*. Using calibration homography computed during system setup, a View Module can produce one of a set of simulated views to be used for object detection. Each Camera Module also contains a Raw View Module responsible for object recognition done on the raw camera view (see Figure 4a-b). We have implemented support for a number of simulated views. Our proof-of-concept applications, described below, make use of the following two simulated views. The *Screen and Bezel View* (Figure 4c) produces a rectified view of the tablet screen and bezel and is useful for performing more precise detection of objects placed on the tablet. The *Table View* (Figure 4d-e) produces a rectified view of the tablet

screen and surrounding surface and is computed such that the tablet screen occupies one of the top quadrants of the rectified view depending on whether the left or right camera is used. Finally, a *Screen View* is also available, producing a rectified view of the tablet screen only. If background subtraction is used, a View Module will distort the background/foreground mask received with the raw camera frame such that it matches the simulated view.

We implemented the View Module class such that the resolution of each of the views (Raw, Screen and Bezel, Table, and Screen) is independent and can be changed depending primarily on the details of objects that need to be detected. Note that while the resolution of a rectified view has performance implications in our prototype (rectification is a pixel-wise multiplication operation), such operations can be accelerated in hardware to reduce this performance hit.

4. *Object classification*. Each View Module contains a list of classifiers responsible for vision-based object recognition. After its view is produced, a View Module gives its simulated view to each of these classifiers. Since some classifiers (*e.g.*, Color-Histogram) make use of foreground/background segmentation, while other classifiers do not (*e.g.*, HaarCascade or Template Matching), our implementation allows each classifier to request that images are returned as-is, or that background subtraction operations take place first. Each classifier then returns a list of detected objects, which can be empty.

Our system supports an extensible set of classifiers. Each classifier must implement a set of basic functions, including `classify()`, `trainNegative()`, `trainPositive()`, and `reset()`. Our system already includes a small set of implemented classifiers such as Template Matching, Compound Template Matching, HaarCascade, and a Color-Histogram. We plan to add a 2D marker classifier as well as a SIFT-based classifier, which is robust to rotation and scaling, in the future.

5. *Aligning object coordinates*. Since different views have different resolutions and coordinate systems in relation to the tablet screen, the coordinates of each object must be transformed into a uniform coordinate system prior to passing the objects to the Camera Module. We chose to transform the coordinates of each detected object to conform to the tablets' coordinate system, such that an object placed at the top-left of the tablet screen will have a coordinate of (0,0), and an object placed at the bottom-right corner of the tablet screen will have a coordinate of (1280, 800) in our prototype. At the end of this step, objects to the left of the tablet have negative *x*-coordinates, objects in front of the tablet have *y*-coordinates greater than 800, and objects to the right of the tablet have *x*-coordinates greater than 1280. Converging on this single coordinate system allows our output system to easily tell whether an object is on or off the screen, and choose one or more visualizations appropriately. The list of objects with updated coordinates is passed to the Camera Module.

6. *Removing redundant objects across views*. When entering this step, a Camera Module may hold more than one set of objects classified on different views with potential redundancies. However, each object must be reported at most *once* to the output system. We thus iterate over the lists returned from the different views and remove duplicates of objects that occur in multiple lists. We consider two objects with the same name and an overlap greater than 75% to be duplicates. The single list of objects is then passed from each Camera Module to the Perception Manager for processing.

7. *Unifying objects across cameras*. Similar to the previous step, the final step in the process is to merge objects returned from the two cameras, which is a step relevant to objects within the overlapping area between the two cameras. Unlike merging objects from different views of the same camera, however, we cannot assume that a single object detected by both cameras will have perfect overlap between the views. In fact, for any 3D object, we can assume that this will *not* be the case. Our system uses the overlapping area for an object seen by both cameras as the possible base of the object. For overlapping objects, only the intersecting area is thus reported to the output subsystem to approximate an object's touch point with the

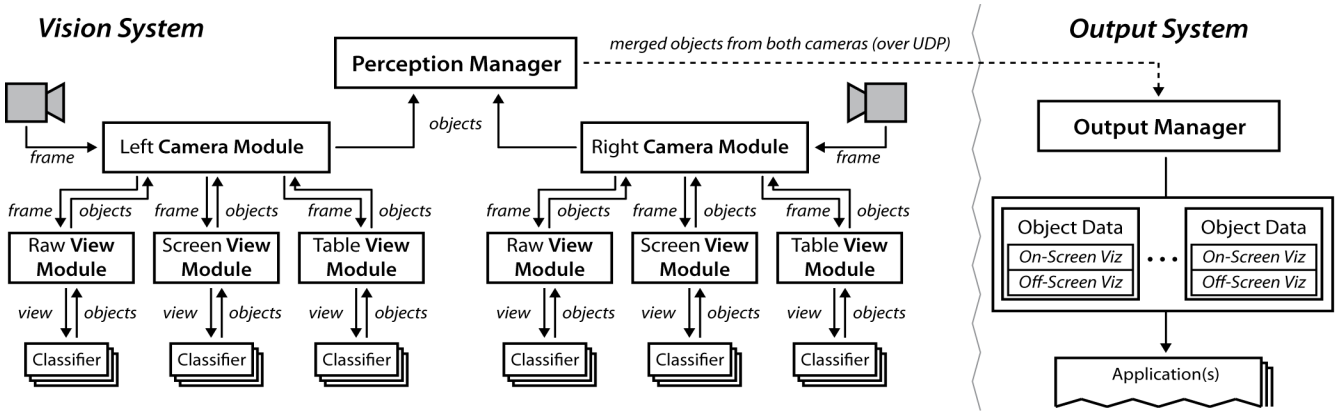

Figure 3. System diagram. Vision system with two camera feeds communicates detected objects to the output system over UDP sockets. Digital representations for each object are created and passed to subscribed applications.

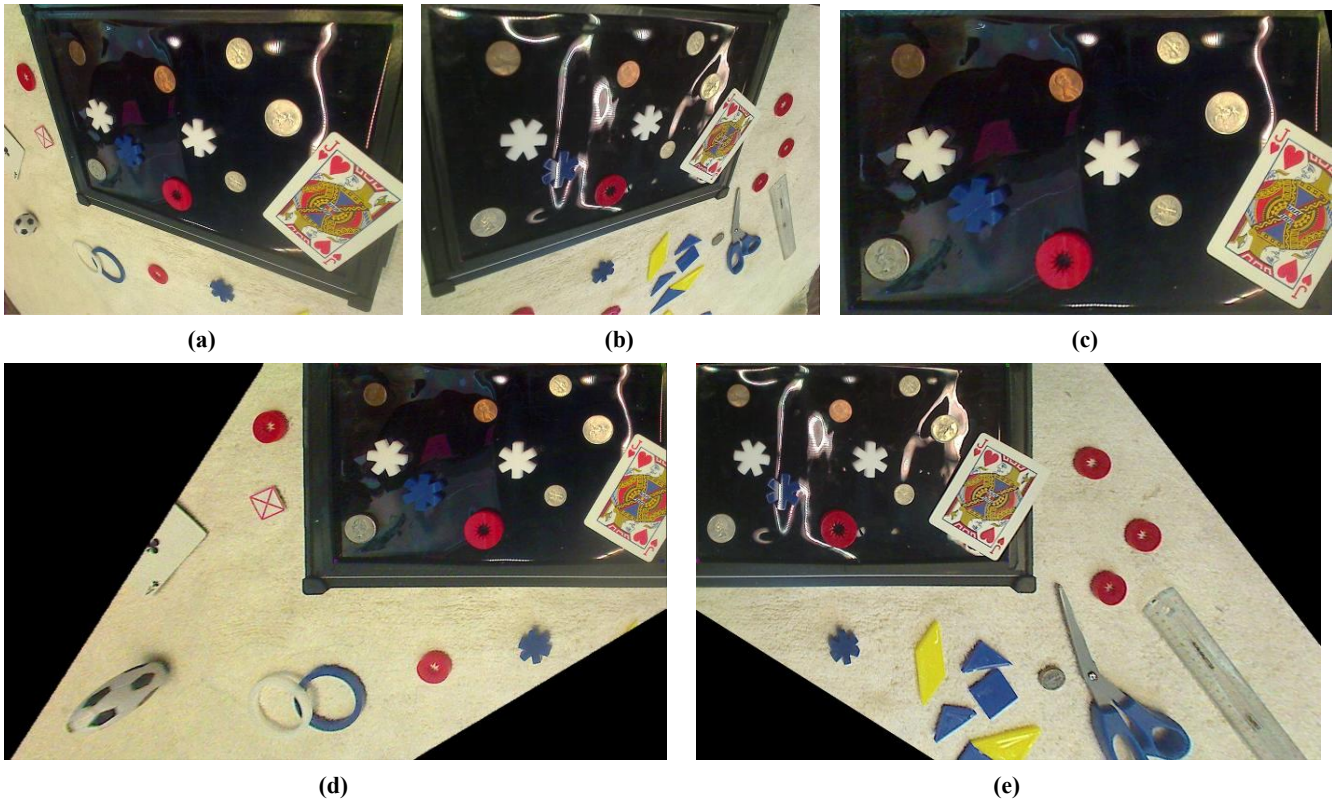

Figure 4. Multiple camera views: The raw camera view of (a) the right, and (b) the left cameras. (c) A rectified view of the screen and bezel (from the left camera), and the rectified view of screen and tabletop surface from (d) the right and (e) the left cameras.

surface and provide feedback at the appropriate position.

8. *Passing Objects to the Output System*. Finally, the unified list of objects from both cameras is passed over UDP sockets to the output system.

System Calibration

Prior to first use (~~in~~ the factory"), the system is calibrated to generate the different camera views and to deliver correct object positions. System calibration consists of three steps. Note that because the camera and screen are orthogonally polarized, these steps cannot use virtual on-screen markers. In the first step, the four corners of the screens are located, once per camera. The transformation homographies are stored on file and are loaded whenever the system starts. In our prototype, the corners are manually selected with the stylus; however, in the future, this step should be performed automatically, *e.g.,* by embedding permanent physical markers at the screen corners.

At this point, an object placed on the screen will correctly receive the same coordinates from both cameras. However, because the plane of the tablet screen is elevated *above* the plane of the table, an object placed on the table surface will receive different coordinates from each camera. Thus, to correct for the difference in planes between the screen and tabletop, in the second calibration step, a single physical marker is placed on the table at the overlap between the two cameras (at this point, the system will report seeing two markers on the table). The system computes an offset that aligns the marker's coordinates, stores it, and later applies it to the coordinates of all off-screen objects. Since the thickness of the tablet is fixed, this step needs to be performed only once.

Finally, in order for physical objects to correspond to meaningful coordinates in application space, the calibration marker is placed at the four corners of a calibration application and a transformation is computed based on the coordinates returned by the vision system. This transform is stored, and later used by our various applications.

Output System

Our output system is responsible for listening for objects delivered over UDP from the vision system, and for providing mechanisms to allow an application to visualize and represent physical objects. By using sockets for the communication between the vision system and output system, we were able to support applications written in different languages. Currently we have implemented two versions of the output system, one for writing applications in C# using .NET 2.0 and the other for Java.

As shown in Figure 3 (right side), the output system contains a UDP client that parses incoming messages for physical objects. Our system contains a basic representation of a physical object (the VisionObject class) which can be subclassed to support specific objects. Each VisionObject (or subclass) possesses the knowledge of how to draw itself when it represents an object on the tablet or when that object lies beyond the tablet's boundaries. The output system uses an object's coordinates to determine whether

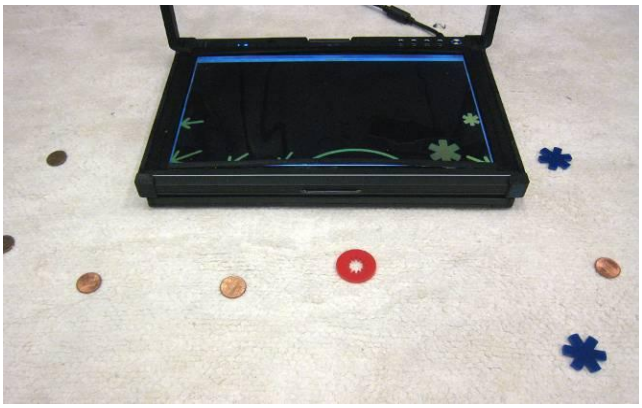

Figure 5. Some of the off-screen visualizations: arrows, icons, and halos.

the object is on- or off-screen, and objects know how to draw themselves based on where they are found in relation to the screen. An Output Manager then passes the parsed objects to any subscribed application, which allows more than one running application to interact with physical objects simultaneously.

For easier application development, we created a Vision Simulator in C# that allows for using simple drag-and-drop operations to send simulated on-screen and off-screen objects to any application. It has been our experience that this decoupling of an application's computer vision requirements and output capabilities greatly increases the ability to iterate and debug applications.

Optional Feedback for Off-Screen Objects

With an input area larger than the screen, conveying to users the system's perception of objects outside its output space could be useful. For example, in Bonfire [12], information about coffee consumption is presented next to the user's coffee cup. However, a user is unlikely to want to place a beverage on their tablet. Similarly, in Classmate Assist [3], a student is guided through a sequence of actions with math manipulatives. Because of the screen size, in Portico, many of the objects will necessarily be off the tablet. In *Halo* [2], Baudisch and Rosenholtz presented a technique for visualizing off-screen objects using contorted partial ellipses. Originally designed for PDAs and phones, Halo demonstrated the ability to convey the location of landmarks on a large virtual map that are off the current display. In Wedge [6], Gustafson *et al.* modified Halo to convey distance and direction, while reducing screen clutter. Inspired by this work, Portico enables application developers to provide users with optional feedback about objects that are off the screen using a library of visualizations that can be easily modified and extended. Figure 5 shows a number of the off-screen visualizations in action. This library includes visualizations for reflecting the presence of an off-screen object and manipulators that can be applied for conveying the distance of the object from the tablet. The following visualizations are implemented:

Line. A line, or ray, is drawn from the center of the tablet screen in the direction of the object that is on the table. The line can be used with thickness or an alpha-blend to indicate an object's distance from the tablet.

Icon. An icon representing the object is drawn at the edge of the screen. The position of the icon is computed such that the icon is placed at the edge of the screen on the imaginary line connecting the object's and screen's centers.

Arrow. Similar to the icon, an arrow is drawn at the edge of the screen pointing in the direction of the object. Arrows have stems by default, but those can be turned off. In our prototype, thinner, longer arrows are used for objects that are farther away from the tablet screen.

Callout. A callout looks similar to a speech bubble with the tail pointing towards the off-screen object. A callout includes a label and an optional icon. It can be sized and alpha-blended to reflect an object's distance.

Halo. Inspired by Baudisch and Rosenholtz's [2] technique for visualizing virtual off-screen elements, we support halos to visualize physical elements off the tablet screen. Halos are arcs at the screen edge that are part of a virtual circle centered at the physical object. Thus, a halo with a larger radius conveys that an object is further away from the screen.

Alpha Blend. This visualization manipulator changes the alpha value of a visualization based on the object's distance from the screen. When the object is touching the screen, the alpha value is 100%. We set the alpha value for an object at the edge of the outer interaction space to 20%.

Size. This manipulator changes the size of the visualization. By default, greater size indicates that objects are further away; however, this can be easily reversed by altering options in a settings file.

Length. For some elements, a longer representation can be useful to conveying that the object is further away. In our system, the stem of an arrow becomes longer the farther away an object is.

Figure 6. Using physical tokens to play *Tic Tac Toe* on the tablet. The game is supervised by Portico and any illegal moves are flagged (and audibly "buzzed").

Figure 7. *Penalty Shootout*. The player rolls the soccer ball on the table or floor towards the screen. After the toy ball hits the tablet chassis, a virtual ball continues the physical ball's trajectory and speed towards the goal.

Line-Thickness. Portico can change the line thickness used by some visualizations, for example, making lines thicker for objects nearer to the screen.

PORTICO'S PROOF-OF-CONCEPT APPLICATIONS

To put Portico through its paces, we created a number of fully functional applications designed to highlight Portico's capabilities to respond to objects on and off the screen. Our examples highlight some playful uses of Portico.

Tic Tac Toe: Tracking On-Screen Objects

Tic Tac Toe is a basic demonstration of our system's ability to detect objects on the tablet, and to make use of the tablet's screen and touch capabilities. In this application, physical tokens are used by one or two players to play the classic game (Figure 6). Portico recognizes tokens' positions on the screen to determine the state of the game-board and highlight tokens from underneath.

After placing a piece, a player touches the NEXT button on the screen to indicate the completion of their turn. At this point, the application checks the state of the game board to ensure that no "mistakes" or "cheats" were made. If one is found, the application plays an audible buzzer, and visually flags the illegal pieces. Although simple, Tic Tac Toe conveys Portico's ability to oversee physical transactions and augment the experience, in this case, by providing a referee or, if desired, a computer opponent.

Penalty Shootout: Dynamic Off-Screen Interaction

Our second application is called *Penalty Shootout*. In this application, designed to be played on the floor or carpet, the tablet screen shows a soccer goal and goalie. The player physically rolls a small soccer ball-shaped foosball on the carpet toward the virtual goal on the screen. When the physical ball hits the tablet's chassis, a virtual ball appears and flies toward the goal along the same trajectory as the incoming physical ball (Figure 7) and the goalie tries to block the shot.

Penalty Shootout takes advantage of the tablet being raised from the table so that the physical ball can impact the tablet and then return, more or less, to the user. Velocity and acceleration in the physical world are converted directly into virtual motion on the screen. The velocity of the physical ball is translated into the height of the virtual soccer ball on the screen. Thus, if the physical ball is rolled too fast, the shot will appear to travel over the goal.

The direction of the physical ball is computed by performing linear regression on the observed locations of the incoming ball samples. The centroid of the template-matched ball is used. For recognition purposes, the vision system was trained on "blurry" images of a ball being rolled at the tablet as well as stationary soccer balls. Linear regression provided some robustness to noisy samples.

Penalty Shootout highlights our system's ability to detect objects off the tablet, track their velocity and trajectory, and react to them on the screen. Our use of physics calculations was inspired by Wilson *et al.*'s [25] work on bringing physics to interactions with virtual objects on a Microsoft Surface. Penalty Shootout is just one in a large class of possible games that make use of the trajectory and velocity of tracked physical objects around the surface. Others include "eagle-eye" golf, bowling, or marbles.

Portico Arcade: Off-Screen Physical Controls

Portico Arcade pays tribute to Atari's classic 1979 arcade game *Asteroids*. In Portico Arcade, physical interaction with a custom toy we've created takes place entirely off the tablet screen. Similar to SideSight [4], user's actions do not block their view of the screen. In this game, the angle of a plastic spaceship on the table is used to control the rotation of a virtual spaceship on the screen and aim its lasers (Figure 8). As in the original Asteroids game, the player's goal is to shoot floating asteroids while avoiding being hit by them. We also created physical add-on weapons that can be snapped onto the plastic ship for added firepower. In our example, snap-on cannons enable the player to fire three lasers at a time instead of just one. To determine the angle of the spaceship, two circular visual tags, one on each wing, are tracked and the angle between them is calculated. Other visual tags are used to detect the presence of add-on

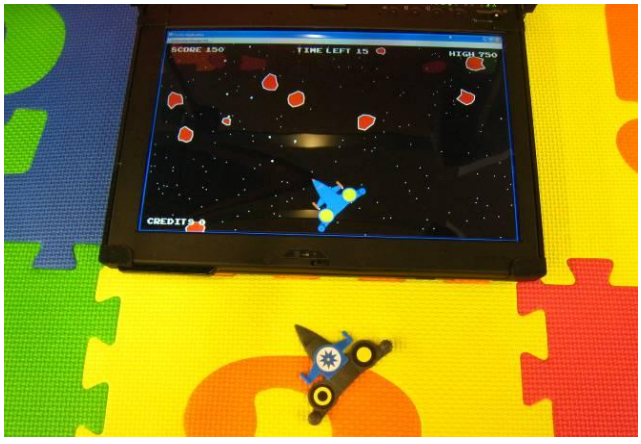

Figure 8. *Portico Arcade* uses a physical spaceship on the table to control the angle of an on-screen virtual spaceship. Physical add-on weapons provide increased firepower.

weapons. If a tag on either wing is not detected, an indicator on the corresponding wing of the on-screen virtual spaceship turns from orange to gray, and the ship does not rotate. This feedback is meant to help the user realize that the system's view of the ship is obstructed, or that the ship is outside the system's interactive area.

Tabletgotchi: Virtual Worlds for Physical Pets

Our fourth proof-of-concept application, *Tabletgotchi*, allows a child to play with an uninstrumented physical toy pet (*e.g.*, a plush zebra), and have the system react to the toy's state on, or around, the tablet (see Figures 1 and 9). Tabletgotchi was inspired by Webkinz[1] and the original Tamagotchi Japanese virtual pets. With Webkinz, a child links a physical plush pet with its virtual instantiation in the digital world on the Webkinz's web site. The child can then play with the virtual toy in its virtual world or with the physical toy in the physical world. However, the physical and virtual remain disconnected. With Tabletgotchi, we establish a link to bring the two worlds directly together.

In Tabletgotchi, a physical toy zebra can ~~"e~~at" food shown on the screen, or ~~"d~~rink" from a virtual pool of water. (A timer periodically replenishes the supply.) A sand area provides a place for the zebra to use the ~~"p~~otty." The physical orientation of the zebra is also relevant in Tabletgotchi. Specifically, the zebra can ~~"n~~ap," as our system distinguishes between a standing zebra and a zebra lying down. For example, laying the zebra on its side below the tablet triggers a ~~"d~~ream" where a nature video of zebras in the wild plays above the zebra's head on the screen. If the zebra stands up mid-dream, the state change is immediately recognized and the dream promptly stops. Future versions could add 3D gesture tracking to enable actions such as ~~"p~~etting" to be recognized and rewarded.

We believe that allowing the physical toy to participate in play, rather than substituting it as with Webkinz, will encourage young children to interact with objects around them, and reduce the extent to which they focus their attention entirely on digital content.

DISCUSSION

Over the course of this work, we obtained insights into the capabilities and limitations of Portico. We believe that Portico represents a new approach to enabling personal and portable tangible surface interaction. We also believe that the interaction area provided by Portico, larger than the screen of the tablet, is necessary for meaningful and extensible portable tangible applications, and we consider this to be a primary contribution of Portico. Furthermore, with the vision and output systems in place, developing applications is easy. For example, *Tabletgotchi* and *Pentaly Shootout* were 600 and 552 lines of C# code respectively, and each used template matching of prototypical views of the objects with approximately 20 negative examples to train the classifier threshold.

[1] Webkinz: http://www.webkinz.com/

Figure 9. *Tabletgotchi* uses an uninstrumented zebra that "eats" carrots, "drinks" water (right side), uses the sand area (left side), or lays below the tablet and dreams (above). If the zebra stands up, the dream video stops, responding to the change in the zebra's physical state.

Beyond interaction with physical objects for tangible manipulation, prior work, such as DigitalDesk 23] and Bonfire [12], proposed incorporating everyday objects to enrich a person's computing interaction. In the work on Bonfire, for example, a coffee cup is tracked for personal logging. In order to support such interaction with a personal tabletop system, however, a large class of objects (*e.g.*, liquids and other consumables) must be recognized without the need for the user to place them on the screen. Our system provides a natural extension to these systems in its ability to recognize objects around the tablet. A cup placed next to (rather than on) the tablet would be detected and tracked. While the absence of a projector makes it impossible to provide labels right next to objects, off-screen visualizations can be used instead.

System-in-Use

We informally tested our system with one 4 year-old and two 3-year old boys. Our applications clearly targeted a young crowd and the use of familiar physical objects was very appealing to them. The children were keen to use the system and the objects right away and greatly enjoyed their experiences. In one case, one of the 3-year old children showed that he personally identified with the zebra's actions and drew analogies to his own experiences. (~~"H~~e [the zebra] uses the potty like I use the potty!")

One noteworthy behavior that we did not expect was that, once the children were done using an application as we intended, they attempted to use objects with applications not designed for those objects. For example, one child tried using the tic-tac-toe pieces in the Tabletgotchi application and was confused when the system did not react to these objects. This observation suggests that our classifiers should support the option to recognize the presence of unknown objects, and suggests that applications still act on these unknown objects in some possibly playful or meaningful way.

System Performance

To give a sense of Portico's performance, we examined the number of frames-per-second (fps) under different

conditions. As a baseline, with both cameras set to a resolution of 800×600 and without any additional processing, the vision system runs at 30 fps. Next, the system yields 24 fps for rectification of each camera once (*e.g.*, to generate the Table view) and 18 fps to rectify each camera twice (*e.g.*, to generate both the Table and Bezel views). Such frequent matrix manipulations for rectification could take advantage of hardware acceleration. The compute time required for classification is dependent on the type of classifiers used and the number of objects possible in the scene. In Portico Arcade and Tabletgotchi, for example, the system runs at 14 fps, which we found sufficient for our explorations.

System Constraints

Contour of the Interaction Space

One possible limitation of our system mentioned earlier is that the cameras' coverage to the right and left of the tablet screen is angled. As a result, the contour of the area viewed by the cameras resembles a wedge rather than a rectangle, which may make perceiving the bounds of the interaction space more difficult for users. The shape of the interaction space (see Figures 4d-e) is determined by the type of cameras used and their placement. In designing our prototype, for example, we limited the height of the cameras (specifically, the arms on which the cameras are mounted) to the depth of the screen such that, when folded, the cameras are flush with the screen and allow the tablet to be used normally. We also opted to utilize cheap web cameras, as they provide the benefits of being lightweight, low-cost, and widely available. While the current shape of the interaction space does not pose a limitation for all applications, for other applications, growing or widening the interaction space may be desirable. Fish-eye lens cameras, for example, could provide a wider view of the tabletop around the screen than our current cameras, although they would sacrifice image fidelity at the periphery. Similarly, different placements of the cameras, likely with different mounting arms, will produce different interaction spaces. Finally, on-screen visualizations, for example, a ~~"r~~adar" view of the interaction space that, at a glance, can highlight all objects recognized by the system and convey the extent of the interaction space, may also help.

Sensitivity to Illumination Changes and Obstruction

Like any top-view camera-based system, Portico is sensitive to changes in illumination and obstruction. IR-based sensing, especially when done from behind the screen, can robustly detect the placement of objects on the screen. Top-down camera systems, however, can observe more of the object's shape than merely its contact points with the surface, as well as see objects from the user's point of view. While Portico's top-down cameras were adequate for a proof-of-concept, a system that combines our design with the optical sensors used in ThinSight [4] could benefit from both approaches. IR-based *depth sensing* cameras can provide robustness to illumination, together with added 3D capabilities; however, many of the objects used in tangible educational and playful applications (*e.g.*, coins, Cuisenaire rods, Tangram and jigsaw puzzle pieces) are thinner than the depth resolution provided by current 3D cameras. Another constraint of our system is that cameras may fail to see objects that are very close to or touching the tablet if the tablet is thick. As new tablets are made thinner (the recent iPad's thickness is 8.8 mm compared to our prototype's 35 mm), this problem will be significantly reduced.

Potential for Use in Education

Tangible and surface computing can present significant benefits to education. The use of manipulatives in mathematics education dates back to the 19th century with Swiss educator Johann Heinrich Pestalozzi [16]. Manipulatives are concrete objects that can be viewed and physically handled by students in order to demonstrate or model abstract concepts. In a meta-analysis of 60 studies, Sowell [19] affirms the effectiveness of manipulative material for students' achievement and attitudes in math education, particularly over long-term use. Studying the use of tangible computing for education, Antle *et al.* [1] compared how children solve a digital puzzle using a mouse, a tangible augmented puzzle, and a standard physical puzzle. Their results show that children were more successful and faster at solving puzzles using a tangible-based approach. Related results by Tuddenham *et al.* [21] suggest that when using tangibles, users are both quicker to acquire, and more accurate in manipulating, interface control objects, compared to using multi-touch or mouse-and-puck. Patten *et al.* [15] proposed using Sensetable for chemistry and system dynamics simulation applications. Zuckerman *et al.* [26] presented a system that uses computationally enhanced manipulatives. They show that their manipulatives are accessible to young children and encourage learning of abstract concepts through an iterative hands-on process. Scarlatos [17] used multimedia and visually tagged tangible objects to guide children in collaborative problem solving with the TICLE system. Horn *et al.* [9] demonstrated the use of tangible interaction for instruction of computer programming concepts.

As this prior research shows, the promise of tangible interaction to improve the learning experience is great. Similar to [3], we believe, however, that in order for this promise to be realized at a large scale, tangible interaction must be supported by personal systems, not only by traditional fixed tabletop computers. With Portico, we seek to deliver the benefits of tangible interaction to children in the classroom, in their home, or even on the living room carpet, all in a low-cost portable and personal form.

FUTURE WORK

As mentioned above, our system currently includes only a handful of object classifiers and we intend to extend this set to support a greater range of objects. We also intend to support tracking of objects across frames, as shown in [24]. The use of IR-based cameras could mitigate the sensitivity to illumination changes, while higher resolution cameras will allow for scanning of physical objects, similar to [13].

Our current prototype, which used a 12" tablet, was able to produce an interaction space six times larger. In our desire to support our system's use for education, we have since created a prototype using a small 10" tablet, a platform more suitable for use within schools. We plan to explore the use of our system to support math-manipulative activities in the future.

CONCLUSION

This paper presents Portico, a low-cost surface computing system that supports tangible interaction. Using its mounted cameras, Portico greatly extends a tablet's interaction space to include objects manipulated on the tabletop surface around it. We presented four proof-of-concept applications—*Tic Tac Toe*, *Penalty Shootout*, *Portico Arcade*, and *Tabletgotchi*—each of which highlights an important aspect of Portico's capabilities. Unlike most prior surface computing systems, Portico is based on a *personal* and *portable* form factor, namely a tablet computer, and as such, Portico brings surface computing one step closer to everyday use.

ACKNOWLEDGEMENTS

We thank Shaun K. Kane, Anthony LaMarca, Matthai Philipose, Richard Beckwith, Noam Avrahami, Caelan D. Wobbrock, and Malcolm Greenstein.

REFERENCES

1. Antle, A. N., Droumeva, M., and Ha, D. 2009. Hands on what?: Comparing children's mouse-based and tangible-based interaction. In *Proc. IDC '09*, 80-88.
2. Baudisch, P. and Rosenholtz, R. 2003. Halo: A technique for visualizing off-screen objects. In *Proc. CHI'03*, ACM Press, 481-488.
3. Beckwith, R., Theocharous, G., Avrahami, D. and Philipose M. 2010. Tabletop ESP: Everyday sensing and perception in the classroom. Intel Technology Journal *14* (1), 16-31.
4. Butler, A., Izadi, S., and Hodges, S. 2008. SideSight: Multi-"touch" interaction around small devices. In *Proc UIST '08*, ACM Press, 201-204.
5. Edge, D. and Blackwell, A.F. 2009. Peripheral tangible interaction by analytic design. In *Proc. TEI '09*, 69-76.
6. Gustafson, S., Baudisch, P., Gutwin, C., and Irani, P. 2008. Wedge: Clutter-free visualization of off-screen locations. In *Proc. CHI'08*, ACM Press, 787-796.
7. Harrison, C. and Hudson, S. E. 2009. Abracadabra: Wireless, high-precision, and unpowered finger input for very small mobile devices. In *Proc. UIST'09*. 121-124.
8. Hodges, S., Izadi, S., Butler, A., Rrustemi, A., and Buxton, B. 2007. ThinSight: Versatile multi-touch sensing for thin form-factor displays. In *Proc. UIST'07*, ACM Press, 259-268.
9. Horn, M.S., Solovey, E.T., Jacob, R.J.K. 2008. Tangible programming and informal science learning: Making TUIs work for museums. In *Proc. IDC'08*, 194-201.
10. Ishii , H. and Ullmer, B. 1997. Tangible Bits: Towards seamless interfaces between people, bits and atoms. In *Proc. CHI'97*, ACM Press, 234-241.
11. Izadi, S., Agarwal, A., Criminisi, A., Winn, J., Blake, A., & Fitzgibbon, A. 2007. C-Slate: A multi-touch and object recognition system for remote collaboration using horizontal surfaces. In *Proc. Tabletop'07*. 3-10.
12. Kane, S. K., Avrahami, D., Wobbrock, J. O., Harrison, B., Rea, A. D., Philipose, M., and LaMarca, A. 2009. Bonfire: a nomadic system for hybrid laptop-tabletop interaction. In *Proc. UIST'09*, ACM Press, 129-138.
13. Kirk, D., Izadi, S., Sellen, A., Taylor, S., Banks, R. & Hilliges, O. 2010. Opening up the family archive. In *Proc. CSCW'10*, ACM Press, 261-270.
14. Kratz, S. and Rohs, M. 2009. HoverFlow: Expanding the design space of around-device interaction. In *Proc. MobileHCI '09*, ACM Press, 1-8.
15. Patten, J., Ishii, H., Hines, J., and Pangaro, G. 2001. Sensetable: A wireless object tracking platform for tangible user interfaces. In *Proc. CHI'01*, 253-260.
16. Saettler, P. *A History of Instructional Technology*. New York: McGraw-Hill, 1968.
17. Scarlatos L.L. 2002. TICLE: Using multimedia multimodal guidance to enhance learning. *Information Sciences 140* (1-2), 85-103.
18. Shaer, O. and Hornecker, E. 2009. Tangible user interfaces: Past, present and future directions. *Foundations and Trends in Human-Computer Interaction 3* (1-2), 1-137.
19. Sowell, E. 1989. Effects of manipulative materials in mathematics instruction. *Journal for Research in Mathematics Education*, 20: 498–505.
20. Stauffer, C., and Grimson, W. 1999. Adaptive background mixture models for real time tracking. In *Proc. CVPR'99*, IEEE, 246-252.
21. Tuddenham, P., Kirk, D., and Izadi, S. 2010. Graspables Revisited: multi-touch vs. tangible input for tabletop displays in acquisition and manipulation tasks. In *Proc. CHI '10*, ACM Press, 2223-2232.
22. Ullmer, B. and Ishii, H. 1997. The metaDESK: Models and prototypes for tangible user interfaces. In *Proc. UIST'97*, ACM Press, 223-232.
23. Wellner, P. 1991. The DigitalDesk calculator: Tangible manipulation on a desk top display. In *Proc. UIST'91*, ACM Press, 27 - 33.
24. Wilson, A. 2005. PlayAnywhere: A compact interactive tabletop projection-vision system. In *Proc. UIST'05*, ACM Press, 83-92.
25. Wilson, A. D., Izadi, S., Hilliges, O., Garcia-Mendoza, A., and Kirk, D. 2008. Bringing physics to the surface. In *Proc. UIST'08*, ACM Press, 67-76.
26. Zuckerman O., Arida S., and Resnick M. 2005. Extending tangible interfaces for education: Digital Montessori-inspired manipulatives. In *Proc. CHI'05*, ACM Press, 859-868.