Toolkit support for vision-based tangible interfaces

Scott R. Klemmer and James A. Landay

Group for User Interface Research
Computer Science Division
University of California
Berkeley, CA 94720-1776, USA
{srk, landay}@cs.berkeley.edu

Abstract. Studies of office workers, web designers, and oral historians have found that even in the digital age (and sometimes because of it), we are using paper more and more. The paperless office is a myth. The paper-saturated office is not a failing of digital technology; it is a validation of our expertise with the physical world. We use paper, and writing surfaces more generally, in their myriad forms: books, notepads, whiteboards, Post-it notes, and diagrams. We use these physical artifacts to read, take notes, design, edit, and plan. Here, we present a toolkit for building vision-based tangible interfaces. We also present the user-centered design methods we employed to build this toolkit.

1 Introduction: Augmenting the Physical World

Beginning with Wellner's Digital Desk [33], researchers have explored how to better integrate the physical and electronic worlds. In 1997, Ishii and Ullmer [16] proposed the Tangible User Interfaces (TUIs) research agenda, giving the area of tangible computing a more concretely shaped vision. Currently, researchers around the world are building physical interfaces (e.g., [18, 20, 22, 23, 32]). TUIs are an important part of ubiquitous computing research, as they provide the means for truly embedding interaction in the user's environment.

The Myth of the Paperless Office [28] describes field research the authors undertook over several years at a variety of office companies. The central thesis of the book is that paper is often viewed as inefficient and passé, when in actuality it is a nuanced, efficient, highly effective technology. The authors are not asserting that "paper is better than digital" or vice versa, but that our naïve utopia of the paperless office is mistaken. Digital technologies certainly change paper practices, but they rarely make paper irrelevant.

There are excellent reasons for researchers to embrace, not abandon, our interactions with everyday objects in the physical world. Paper and other everyday objects:

- Allow users to continue their familiar work practices, yielding safer interfaces [20].
- Are persistent when technology fails, and thereby more robust [22].
- Enable more lightweight interaction [13].

– Afford for fluid collocated collaboration.
– Are higher resolution, and easier to read than current electronic displays.

However, "tangible computing is of interest precisely because it is not purely physical" [8]. Researchers have electronically augmented paper and other everyday objects to offer:

– An interactive history of an evolving physical artifact [19].
– Collaboration among physically distributed groups.
– Enhanced reading.
– Associative physical links to electronic resources.
– Physical handles for fluid editing of electronic media.
– Automated workflow actions.

There are difficulties in employing paper and everyday objects as a user interface. When paper is used as an interactive dialog, the update cycle (printing) is much slower than with electronic displays. When multiple actors (computational or human) control the physical data (e.g. Post-it notes [19]), the application needs to reconcile the physical objects representing an inconsistent view. This can be handled by either prohibiting such actions, or by electronically mediating them. Physical sensors (especially computer vision) require substantial technological expertise to be built robustly. Many developers have excellent ideas about how physical computing can better support a task, but lack this technological expertise.

The difficulties involved in building tangible interfaces today echo the experiences of the GUI community of twenty years ago. In 1990, Myers and Rosson found that 48% of code and 50% of development time was devoted to the user interface. One of the earliest GUI toolkits, MacApp, reduced Apple's development time by a factor of four or five [25]. We believe that similar reductions in development time, with corresponding increase in software reliability and technology portability, can be achieved by a toolkit supporting tangible interaction.

While the research community has shown the substantial benefits of tangible interaction, these UIs are currently very difficult and time consuming to build, and the required technology expertise limits the development community. The difficulty of technology development and lack of appropriate interaction abstractions make designing different variations of an application and performing comparative evaluations unrealistic. In each of the twenty-four research systems we have studied [17], at least one member of the project team was an expert in the sensing technology used. Contrast this with GUIs, where developers are generally experts in the domain of the application, not in raster-graphics manipulation.

GUI tools have been so successful because, "tools help reduce the amount of code that programmers need to produce when creating a user interface, and they allow user interfaces to be created more quickly. This, in turn, enables more rapid prototyping and, therefore, more iterations of iterative design that is a crucial component of achieving high quality user interfaces" [24].

The Papier-Mâché research project seeks to provide toolkit level support for physical input. We believe that handling physical input at the toolkit level will enable developers to (1) quickly build paper-based tangible user interfaces and (2) change

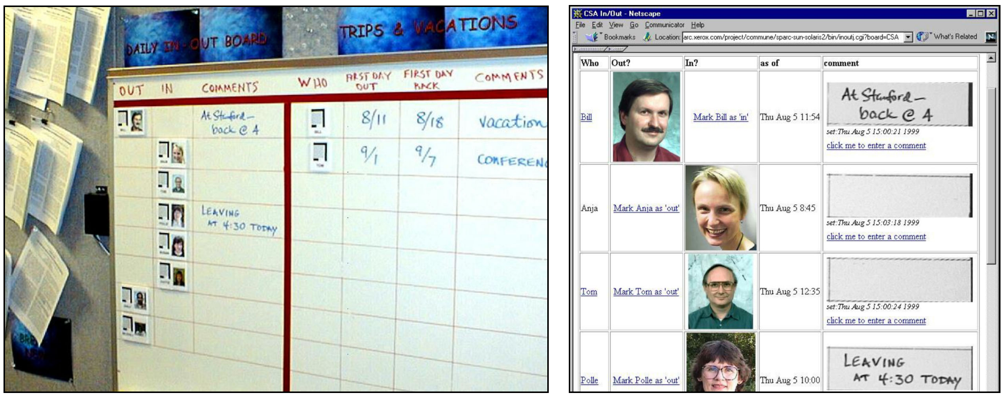


Figure 1. Collaborage. *Left:* Hallway with In/Out and Away boards. *Right:* Web browser view of part of the In/Out board.

the underlying sensing technologies with minimal code changes. Papier-Mâché also enables further tangible interface research by providing an open-source platform for researchers, students, and commercial developers.

In its full form, Papier-Mâché will support vision, bar-code, and RFID tag input. Vision is the most flexible, powerful, and unwieldy of these technologies. For these reasons, we have researched toolkit support for vision-based UIs first. Tool support in this domain is minimal: while software libraries such as JAI [1] and OpenCV [4] aid vision developers in image processing tasks, there are no tools that enable developers to work with vision-based UIs at the application/event level. The primary contribution of this paper is the introduction of a high-level event mechanism for vision-based TUIs.

2 Application Space

We conducted a literature survey of existing tangible user interfaces, looking specifically for examples of systems employing paper and other "everyday" objects (as opposed to mechatronic interfaces).

The twenty-four representative applications fall into four broad categories: *spatial, topological, associative,* and *forms.*

Spatial applications include augmented walls, whiteboards, and tables, used for collaboratively creating or interacting with information in a Cartesian plane. Collaborage (see Figure 1) is a spatial application: it is "a collaborative collage of physically represented information on a surface that is connected with electronic information, such as a physical In/Out board connected to a people-locator database" [23].

Topological applications employ physical objects as avatars (e.g., for airplanes, media files, and PowerPoint slides). Arranging these objects determines the behavior of the corresponding electronic system. Paper Flight Strips [20] is a topological application: the system augments the flight controllers current work practice of using paper flight strips by capturing and displaying information to the controllers.

With *associative* applications, physical objects serve as an index or "physical hyperlink" to digital media. Durrell Bishop's Marble Answering Machine [6] is an associative application. An answering machine deposits a physical marble (with an embedded RFID tag) each time a message is left. To play a message, one picks up the marble and drops it into an indentation in the machine.

Forms applications, such as the Paper PDA [13], provide batch processing of paper interactions. Users work with paper in a traditional manner, then scan or fax it to initiate electronic behaviors. The Paper PDA is a set of paper templates for a day-planner. Users work with the Paper PDA as with any other paper day-planner. They can then scan the pages in, and information will be electronically captured.

These twenty-four applications share much functionality with each other, including:

– Physical input for arranging electronic content
– Physical input for invoking actions (e.g., media access)
– Electronic capture of physical structures
– Coordinating physical input and graphical output in a geo-referenced manner.
– An Add, Update, Remove event structure.

In the interest of brevity, this taxonomy broadly omits several classes of interfaces: haptic (e.g., wheels [29]) and mechatronic (e.g., inTouch [5]) UIs, and TUIs whose only input is a tethered 3D tracker (e.g., Doll's head [14]).

3 Motivating Applications

As part of our user-centered design process, we conducted interview surveys with nine researchers who have built tangible interfaces. These researchers employed a variety of sensing techniques including vision, various RF technologies, capacitive field sensing, and bar-codes. We will report on the details of this study elsewhere. Here, we summarize the findings from the three researchers that used vision.

Researcher #1 has a PhD in computer vision, and was the vision expert on an interdisciplinary research team. His team built a wall-scale, *spatial* TUI. Their driving user experience beliefs were:

– "People don't want to learn or deal with formidable technology."
– "They're torn between their physical and electronic lives, and constantly trying work-arounds."
– "Technology should make things more calm, not more daunting."

They used vision because, "it gives you information at a distance without a lot of hassle, wires, and instrumentation all over the place. It puts all the smarts in one device and instrumentation is limited. It also is possible to retrofit existing spaces." His main frustration with using vision was that "getting down and dirty with the pixels" was difficult and time-consuming.

Researcher #2 built a wall-scale, *spatial* TUI augmented with speech and gesture recognition. For rapid implementation, the system was originally implemented with a SMART Board as the sensor. Later, this was replaced with vision for two reasons: 1) SMART Boards expensive and bulky, while cameras are inexpensive and small. 2) SMART Boards provide single-input of (*x, y*). Vision offers a much richer input space. This vision task is exactly the kind of task that Papier-Mâché can support.

Researcher #6 built a desktop *forms* UI incorporating image capture. His main frustration was that, "The real-time aspects of camera interfacing were probably the hardest." This system was designed iteratively over a number of years. At each iteration, user feedback encouraged making the interaction techniques more light-weight and calmer. This echoed the experiences of the other two researchers, as well as our own group's research.

All three researchers mentioned the difficulty of working with cameras. #2 avoided them initially. #1 plowed through anyway, lamenting "it's not always worth it to live at the bleeding edge of technology. … Make sure you have a very good reason if you choose to work on a problem whose solution requires pushing more than one envelope at most."

Myers, Hudson, and Pausch [24] point to rapid prototyping as a central advantage of tool support. Vision is an excellent technology for rapid prototyping of interactive systems. It is a highly flexible, completely software configurable sensor. There are many applications where the final system may be built using custom hardware, but the prototypes are built with vision. An example application built with Papier-Mâché is the Physical Macros class project [7]. As discussed in section 7.1, vision enabled these students to rapidly prototype their system in roughly three weeks. Neither of the students had experience with computer vision. Given the tight time schedule of class projects, this system would not have been possible otherwise.

4 Related Work

We present related work in two areas: GUI input models and TUI input models.

4.1 GUI Input Models

Model, View, Controller (MVC) was one of the earliest attempts at providing guiding software abstractions for the developers of GUIs. MVC separates code into three parts: "the model which embodies the application semantics, the view which handles the output graphics that show the model, and the controller which handles input and interaction. Unfortunately, programmers have found that the code for the controller and view are often tightly interlinked; creating a new view usually requires creating a corresponding new controller" [26]. In practice, most developers write their code in two pieces: a model and a view-controller. While the view-controller distinction makes some historical sense (mice are different than displays), the two are necessarily blurred in practice. An Interactor-like approach [26] is much more common.

Interactors [26] introduced multiple widgets (view-controller encapsulations) with equivalent APIs; a highly influential abstraction that has influenced modern GUI toolkit design (e.g., Java Swing). The original six interactors were: *Menu selection*, *Move-grow*, *New-point*, *Angle* (the angle that the mouse moves around some point), *Text entry*, and *Trace* (for free-hand drawing). By providing a higher-level API, Interactors give application developers a level of independence from implementation details such as windowing systems. Other researchers have continued this model-based direction (see [30] for an excellent summary), enabling application developers to ask the toolkit to execute tasks without being concerned with the actual widget (e.g., theoretically, a menu interaction could be handled via a drop-down menu, radio buttons, a textbox, or even speech input).

4.2 Tangible Computing Toolkits and Interaction Models

Phidgets. The work most related to Papier-Mâché is Phidgets [12]. Phidgets are physical widgets: programmable ActiveX controls that encapsulate communication with USB-attached physical devices, such as a switch or motor. Phidgets are a great step towards toolkits for tangible interfaces. The digital ActiveX controls, like our Java event system, provide an electronic representation of physical state. However, Phidgets and Papier-Mâché address different classes of tangible interfaces. Phidgets primarily support tethered, mechatronic tangible interfaces that can be composed of wired sensors (e.g., a pressure sensor) and actuators (e.g., a motor). Papier-Mâché primarily supports untethered TUIs employing everyday objects.

Papier-Mâché provides stronger support for the "insides of the application" than Phidgets. Phidgets facilitates the development of widget-like physical controls (such as buttons and sliders), but provide no support for the creation, editing, capture, and analysis of physical data. This data is the insides of the application that Papier-Mâché supports. With Phidgets, all servo-motors are equivalent; the physical artifacts are generic. With Papier-Mâché, the physical artifacts are non-generic; the application can have different behavior depending on the content or identity of the physical objects.

One reason for Phidgets' success is that the authors' original research agenda was building applications, not a toolkit. The difficulty and frustration of building and evaluating one-off applications led the authors to the Phidgets concept. Experiential knowledge is very powerful—toolkit designers with prior experience building relevant applications are in a much better position to design truly useful abstractions. Our research group has a similar vantage point: we have spent four years building physical interfaces. Our motivation for building a toolkit also comes through this important experiential knowledge.

Emerging frameworks for tangible user interfaces. In [31], Ullmer and Ishii provide an excellent taxonomy of existing tangible interfaces. We have drawn heavily on both this taxonomy and the innovative ideas of their Tangible Media Group in creating our list of inspirational applications. They also propose MCRpd as analogue to MVC for the physical world. M is the model and C is the controller. The difference is that the view is split into two components: Rp, the physical representation, and Rd, the digital representation. We believe this interaction model is flawed for three reasons. First, MVC is not an appropriate starting point because real application developers rarely write idealized MVC code; Interactors are a more commonly used abstraction. Second, from an implementation standpoint, it is unclear whether explicitly separating physical and digital outputs is beneficial. In fact, for reasons of application portability, it is important that the event layer is agnostic to whether the implementation is physical or digital (e.g., for usability studies, it would be great to be able to create and compare physical and electronic versions of an application with minimal code changes). Third, the approach is untested; no toolkit or applications were ever built explicitly using the MCRpd approach.

iStuff. iStuff [3] provides some compelling extensions to the Phidgets concept, primarily that it supports wireless devices. iStuff provides fast remapping of input devices into their iRoom framework, enabling standard GUIs to be controlled by novel input technologies. By providing a multi-user, multi-device event-based software API, "the technology does not matter to those who access the device in their software applications" [3]. iStuff and Papier-Mâché are highly complementary; it very well might be possible to use iStuff to drive distributed Papier-Mâché applications.

There are two main differences in our research agenda: First, while Papier-Mâché offers no explicit support for retargeting existing applications, iStuff offers novel control of existing applications. To the extent that applications already have a correct input model, iStuff is a huge win. However, the tangible interfaces Papier-Mâché supports do not operate on the GUI input model. Second, like Phidgets, iStuff targets the mechatronic subspace of tangible interfaces, rather than the augmented paper subspace of tangible interfaces. (For example, it would not be possible to build The Designers' Outpost [18] using iStuff.)

Image Processing With Crayons. Fails and Olsen have implemented a highly compelling system for end-user training of vision recognizers, "Image Processing with Crayons" [10]. It enables end-users to draw on training images, selecting the areas of the images (e.g., hands or note-cards) that they would like the vision system to recognize. They employ decision trees as their classification algorithm, using Integral Images as the feature set. The resulting recognizers can be exported as serialized Java objects for incorporation into standard Java software. Crayons complements our work well, offering a compelling interaction technique for designating objects of interest. Papier-Mâché's recognition methods (e.g., edge detection and perspective correction) are higher-level than the pixel-level processing employed by Crayons. We also offer higher-level object information (e.g., orientation), and most importantly, an event mechanism for fluidly integrating vision events into applications. Papier-Mâché also supports ambiguity [21], an important feature unavailable in Crayons.

5 High Level Design

The primary contribution of this paper is the introduction of high-level events for vision-based TUIs. In contemporary vision-based applications, the information that the vision system provides to the application tends to be ad-hoc and written in technology-centered terms. Papier-Mâché introduces an event mechanism that makes much more sense from an application developer's perspective.

A well-specified event API enables a separation of concerns between algorithms development and interface development. Of concern to the application developer is, "When a user places a light-bulb on the work surface, display visual output based on the bulb's location and orientation." A savvy application developer is also likely interested in mediation techniques [21] if an object was recognized with low confidence. These issues live above the event layer. The details of the recognition algorithms are hidden by the event layer. The techniques for object detection can change completely, and the event API (and thereby, application code) does not need to be changed.

Papier-Mâché is Java software written using the Java Media Framework (JMF) [2] and Java Advanced Imaging (JAI) [1] APIs. JMF supports any camera with a standard camera driver, including high-quality cameras such as the Sony 1394 cameras (we use the VL-500) as well as inexpensive webcams. Papier-Mâché is open source software available on the web at *anonymized URL*.

We remind the reader: the contribution of this paper is not in the domain of vision algorithms. Our contribution is a novel set of APIs for building interactive systems and a toolkit that employs well-known algorithms that are effective for this task.

5.1 Motivating scenario: building The Designers' Outpost with Papier-Mâché

We introduce the software architecture with a scenario of how Papier-Mâché would help a developer build The Designers' Outpost [18], a tool that supports information design for the web. Web designers use pens, paper, walls, and tables for explaining, developing, and communicating ideas during the early phases of design [27]. Outpost embraces and extends this paper-based practice through a large electronic wall with a tangible user interface (see Figure 2).

Users have the same fundamental capabilities in the Outpost system as in a paper and whiteboard system. One can create new pages by writing on new Post-it notes, add them to the electronic wall and organize a site by physically moving Post-it notes around on the board. Paper in the physical world becomes an input device for the electronic world. A camera mounted inside the board captures the location of notes, detecting when notes are added, removed, or moved.

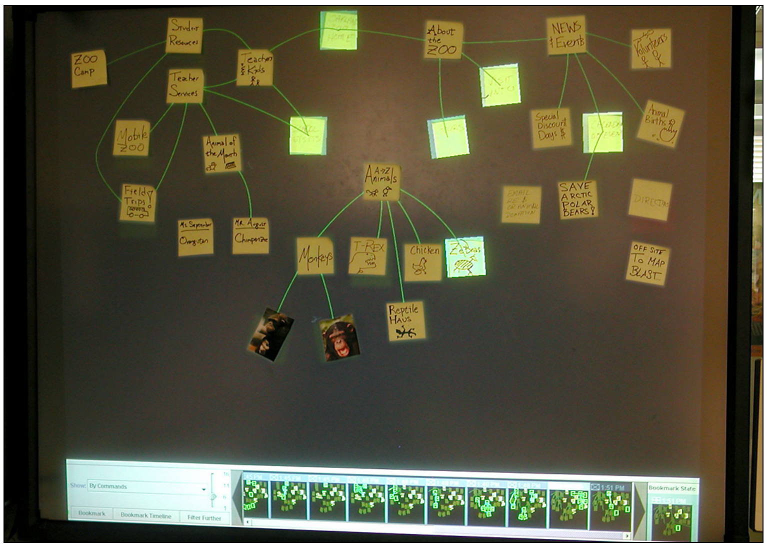
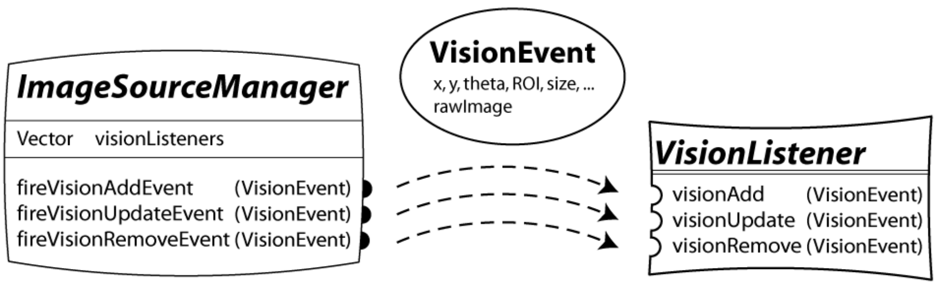


Figure 2. The Designers' Outpost, a system for collaborative web design.

Outpost is a *spatial* TUI. To build Outpost, a developer would begin by instantiating a camera source for the internal camera. She would then instantiate a `PMWindow` (a visual component capable of partaking in camera calibration) as the main canvas for the board. She would add an `EventFilter` that filtered for Post-it note sized objects as a listener to the camera's event stream. She would then instantiate a `VisualAnalogueFactory` as a listener to the note filter, and perhaps add a `MotionlessTranslator` to filter out hand motions. She would have the factory create visual forms with a faint yellow shadow. With just this code, the developer has built a system that tracks Post-it notes placed on the screen and presents visual feedback. To support users tapping on notes, she could add standard Java mouse listeners to the visual forms the factory creates. To extend the system with a remote awareness shadow [9], she would add a second `EventFilter` to the camera. This would filter for person-sized objects. This filter would have a corresponding factory that created the outline shadow using the outline pixel data from the events source.

5.2 Vision Events

The central piece of the toolkit is the `VisionEvent` class. Applications receive `VisionEvents` from an `ImageSourceManager` by registering `VisionListeners` (see Figure 3). A `VisionListener` receives events about all objects larger than a specified minimum size. (This minimum size constraint is solely for performance; it avoids an inappropriately large number of events from being generated.) `VisionEvents` have a similar API to Java's `MouseEvents`, a descendant of the Interactors research. There are several important differences, however.



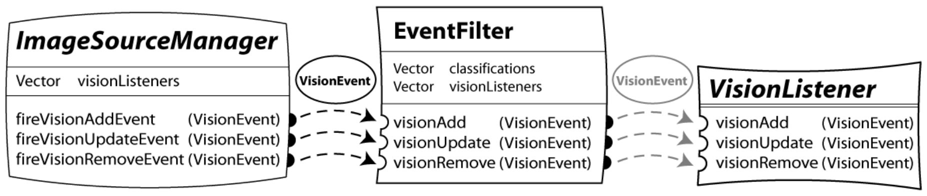
ImageSourceManager
Vector visionListeners
fireVisionAddEvent (VisionEvent)
fireVisionUpdateEvent (VisionEvent)
fireVisionRemoveEvent (VisionEvent)
VisionEvent
x, y, theta, ROI, size, ...
rawImage
VisionListener
visionAdd (VisionEvent)
visionUpdate (VisionEvent)
visionRemove (VisionEvent)

Figure 3. Event dispatching in Papier-Mâché. There can be multiple vision listeners.

1) A mouse is a heavyweight (i.e., "always present"), generic input device; the meaning of its input is constructed entirely through the graphical display. In traditional GUIs there is always exactly 1 mouse. (Some research systems offer multiple mice.) In contrast, tangible interfaces nearly always employ multiple input devices. In many TUIs, the meaning is constructed entirely by the physical and visual form of the object. In other TUIs, the form and a graphical display both determine behavior. In these systems, the input devices are lightweight; multiple objects appear and disappear frequently at runtime. So while `MouseEvents` offer only position updates, `VisionEvents` offer `Add`, `Update`, and `Remove` methods.
2) With a traditional mouse, the only input is (*x, y*) position and button presses. With physical objects on a plane, we have (*x, y*) position, orientation (θ), size, shape information, and visual appearance. Currently, Papier-Mâché also provides bounding box, edge pixel set, and major and minor axis lengths as shape information. We currently provide the mean color, as well as access to the source image data, for visual appearance.
3) Papier-Mâché provides a lightweight form of classification ambiguity. This is discussed in section 5.3.
4) Because of the substantial interpretation involved in moving from camera data to high-level events, events provide a method to get the raw source image from which the event was generated. Having access to the original pixel data, and the region of interest (ROI) for the event provides a lightweight way for developers to extend the toolkit for their own needs. For example, a developer might want to extend the image processing, or capture the raw image data for subsequent display.

5.3 Event filtering, classification, and translation

Nearly all applications are interested in only certain classes of events. For example, Outpost is only looking for Post-it notes and user shadows. Events can be filtered using `EventFilters` (see Figure 4). Filters are both event listeners and producers. They listen to input events, filter according to a specified criteria, and pass along relevant events to the `VisionListeners` registered with the filter. Most filters filter events that meet a certain classification criteria. The three currently imple-



ImageSourceManager
Vector visionListeners
fireVisionAddEvent (VisionEvent)
fireVisionUpdateEvent (VisionEvent)
fireVisionRemoveEvent (VisionEvent)
VisionEvent
EventFilter
Vector classifications
Vector visionListeners
visionAdd (VisionEvent)
visionUpdate (VisionEvent)
visionRemove (VisionEvent)
VisionEvent
VisionListener
visionAdd (VisionEvent)
visionUpdate (VisionEvent)
visionRemove (VisionEvent)

Figure 4. Filters can serve as intermediary listeners.

mented classifiers are `MeanColorClassifier`, `ROIClassifier`, and `SizeClassifier`.

The `MeanColorClassifier` passes along events about objects whose color is within distance ε of an ideal color. The `ROIClassifier` passes along events about objects in a particular region of interest of the camera view. And the `SizeClassifier` passes along events for objects whose size is within a Euclidean distance ε of an ideal size. We plan to implement more classifiers, most notably shape-based classifiers. In our Outpost example, the developer would 1) filter for Post-it note sized objects and 2) filter for person-sized objects.

Additionally, we provide a `MotionlessTranslator`. Many interactive systems optionally employ human body/hand input along with inanimate physical objects. The `MotionlessTranslator` filters out human body/hand input by ignoring moving objects; the remaining events are inanimate. Each of these classifiers returns a confidence value from the classification. These confidence values provide the application with more than just a simple "yes/no."

5.4 Visual Analogues

In all of the interactive vision-based UIs in our taxonomy, camera input drives some form of graphical output. In some cases, physical input and visual output are visually combined, yielding augmented direct manipulation interaction. In other cases, graphical presentation of the physical objects occurs on another collocated or remote display. To support this, we offer a `VisualAnalogueFactory`, a factory class that creates graphical components representing physical objects. It does so by registering itself as a `VisionListener`. This simplifies three important tasks: graphical feedback/augmentation and event handling. In our Outpost example, physical notes cast a graphical shadow on the board. The shadow provides lightweight feedback, informing the user the note has been recognized. Additionally, users can tap on a note to invoke a graphical context menu. By having a graphical component geo-referenced with the physical object, handling a tap on the SMART Board is as simple as adding a mouse listener to the component.

5.5 Wizard of Oz simulation of camera input

We provide a Wizard of Oz simulation mode for times when a camera is not readily available, or more importantly when the developer needs to be able to guarantee a concrete input stream for testing purposes. Wizard of Oz prototyping has a long history. In sensor-rich environments, WOz approaches are particularly beneficial because of the substantial time investment in building sensors and recognizers to produce the desired data. Another benefit of a WOz mode is that developers can continue to work when sensors are unavailable (e.g., on a laptop). In Papier-Mâché's WOz mode, camera input is simulated with a directory of user-supplied images. These images can be either raw images before recognition or bi-level images after recognition. In the bi-level image, all pixels are either white (object) pixels or black (background) pixels. Post-recognition images are useful for prototyping on the assumption the vision will work as planned.

6 Vision implementation details

We now present the vision processing pipeline in Papier-Mâché, including details on the implementation of our algorithms.

6.1 Processing Pipeline

The vision processing in Papier-Mâché has three phases: 1) camera calibration, 2) image segmentation, and 3) creating and dispatching events. These methods form the body of a `processFrame`() method called by a `VisionThread`. For the duration of the application, the thread calls `processFrame`() and then sleeps, allowing other application code to execute.

Camera calibration. We have implemented camera calibration using perspective correction. Perspective warp is an effective, efficient method; most contemporary graphics hardware, and the JAI library, provide perspective warp as a primitive. (More computationally expensive and precise methods exist, see [11], Chapters 1–3 for an excellent overview of the theory and methods). Calibration takes place on application start-up and is accomplished using a projector[1]. The projector projects a white border around the area of interest (image $\mathbf{I_1}$), the camera captures $\mathbf{I_1}$, the projector displays an empty (all-black) image $\mathbf{I_2}$, and the camera captures $\mathbf{I_2}$. We then subtract $\mathbf{I_2}$ from $\mathbf{I_1}$, and the resultant pixels are the border pixels. We conduct line-fitting on the resultant border image, and the intersections of these four lines are the

[1] We provide a default implementation of a `CalibrationListener` that works with a projected display. Camera calibration is also possible when visual projection is unavailable (e.g., Collaborage). In this case, developers implement their own implementation that provides one image with a border around the desired region of interest and another image without this border.

four corners. We initialize a perspective warp with these four corners, and this warp is used in all subsequent frames.

Image segmentation. Segmentation partitions an image into objects and background. (See [11], Chapters 14–16 for an overview of image segmentation.) We employ edge detection, a common technique, to generate a bi-level image where white pixels represent object boundaries and all other pixels are black. We then group the labeled foreground pixels into objects (segments) using the connected components algorithm [15], standard fare for this task.

Generating events. In the connected components algorithm, we build `VisionEvents`. Connected components labels each pixel with a segment ID. When we come across a new segment ID, we create a new `VisionEvent`. As we find subsequent pixels with the same ID, we update information about the event: its center of mass, second-order moments (used for moment-based orientation), and bounding box.

The `ImageSourceManager` fires the events. In each time-step, objects present for the first time yield a `visionAdd(VisionEvent)` call, objects present before yield a `visionUpdate(VisionEvent)` call, and objects that are no longer present yield a `visionRemove(VisionEvent)` call to all registered listeners. This mechanism for deciding on event types can be overriden. Temporal methods of object-finding explicitly generate remove events, for example.

6.2 Finding object orientation

We compute an object's orientation using one of two methods. First, we compute the major and minor axis eigenvector using image moments [15]. The major axis is the long axis of the object; the minor axis is the short axis. If the eigenvectors are robust (i.e., the eigenvalues are significantly above 0), then we use the major eigenvector as an object's orientation. The eigenvectors are unstable when an object is symmetrical about both the X and the Y axes (e.g., squares and circles). When we can't use eigenvectors, we perform line-finding to find linear edge segments. We use the longest linear edge as the orientation. The particular algorithm we use for line-finding is the Hough transform because it is fast, and robust enough for our purposes. (See [11], Chapters 15–16 for a thorough discussion of line and curve fitting.)

We remind the reader that our interest is in API design; our vision algorithms are drawn from the literature, they are not the subject of our research. Additionally, each of these processing steps can be overridden by application developers if they are so inclined.

7 Evaluation

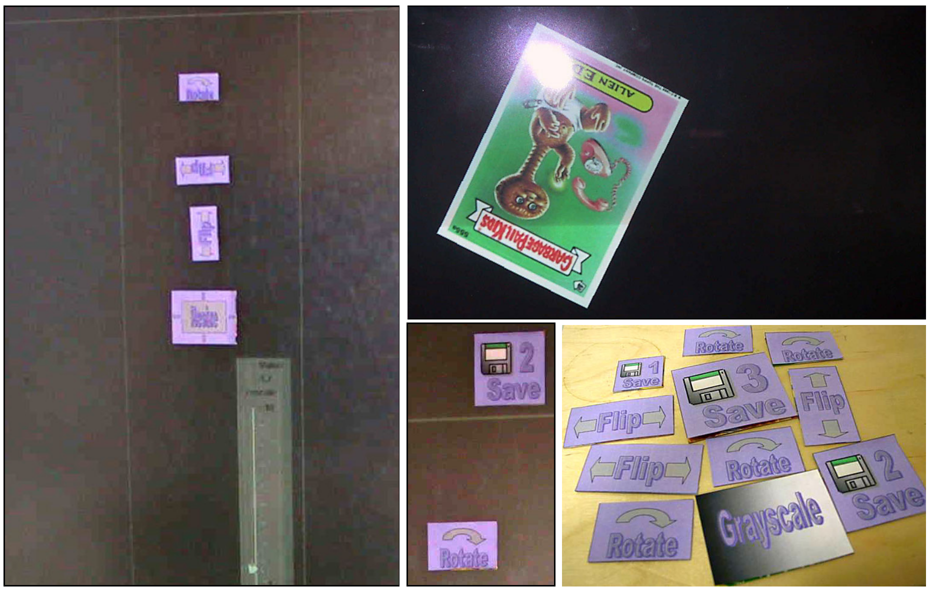


Figure 5. The physical macros class project. *Left:* the application recognizes a rescale block (the last item in the four element chain), and projects a slider, prompting the user for more input. *Top-right:* The composition is displayed on the output monitor. *Bottom-center:* user performs function composition by saving a chain in a save block. *Bottom-right:* The blocks.

We present three types of evaluation for the system. Most importantly, we discuss two class projects that were built using Papier-Mâché. Second, we discuss some performance results. Last, we show a bare-bones "hello vision world" application.

7.1 Applications Built With Papier-Mâché

Two groups in a Spring 2003 offering of the graduate HCI class at our university built projects using Papier-Mâché.

Physical Macros. Physical Macros [7] is a *topological* TUI. The students were interested in researching a physical interface to macro programming environments such as "actions" in Adobe Photoshop. For their domain, they chose image editing. The system provides function blocks (see Figure 5, bottom-right) that can be composed (see Figure 5, left). As the user composes the function set, the graphical display (see Figure 5, top-right) is updated accordingly. A set of functions can be saved in a save block for later reuse.

When the students wrote their system, Papier-Mâché had no visual analogue facilities. Looking through their code, we found that geo-referenced event handling and graphical presentation was a substantial portion of the code. Reflecting on this, we realized that many of our inspiring applications, including [18], also require this feature. For this reason, we introduced the visual analogue classes.

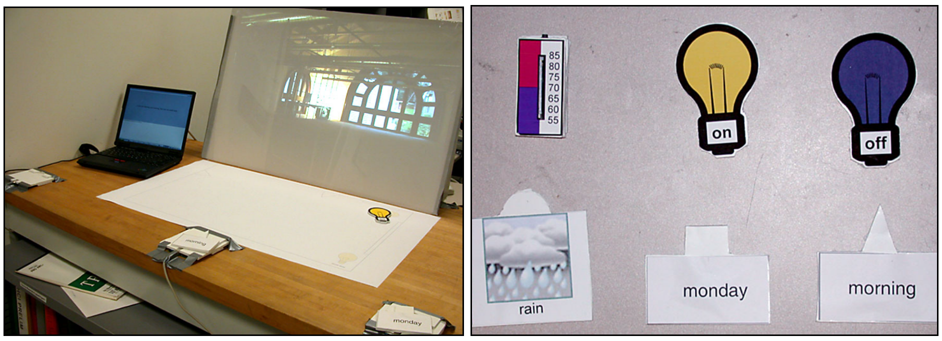


Figure 6. SiteView. *Left:* A light-bulb physical icon on the floor-plan, with projected feedback above. *Right:* The six physical icons.

SiteView. SiteView is a *spatial* UI with similarities to Underkoffler et al's urban planning system [32]. SiteView presents physical interaction techniques for end-user control of home automation systems. On a floor plan of a room, users create rules by manipulating physical icons representing conditions and actions. The system provides feedback about how rules will affect the environment by projecting photographs onto a vertical display. It employs a ceiling mounted camera and three RFID sensors. The camera tracks the objects whose locations the application needs. The RFID reader is used when only the presence or absence of the physical icon is necessary.

SiteView uses `EventFilters` to find the location and orientation of the thermostat and the light bulbs on the floor-plan. The thermostat is distinguished by size, the bulbs are distinguished by size and color. In general, the system worked well, but human hands were occasionally picked up. This inspired our addition of a `MotionlessTranslator`. With this in place, human hands do not seem to interfere with recognition. SiteView is roughly 3000 lines of code (includes comments); of this only about 30 lines access Papier-Mâché code; we consider this a tremendous success. (Outpost, built on top of OpenCV, required several thousand lines of code to achieve a comparable amount of vision functionality.)

7.2 Lowering the threshold: "Hello Vision World"

Hello World has often been a simple example of how much code is needed to create a simple application. Here is our Hello Vision World, a simple application that graphically displays the objects found by the vision system. It is only three lines of code!

```
1 ImageSourceManager mgr = new CameraImageSourceManager
  (SHOW_WINDOW, MIN_SIZE, SLEEP);

2 VisionListener l = new VisualAnalogueFactory(new
  PMWindow(manager, CALIBRATE).getContentPane(),
  JPanel.class);
```

```
mgr.addVisionListener(l);
```

7.3 Performance

On contemporary hardware, Papier-Mâché runs at interactive rates. On a dual Pentium III, we achieve performance of 3.5 frames/second. On a dual Pentium 4, runs at 5.5fps at processor load of 30%. (It's possibly the way Java handles memory access causes excessive blocking.) SiteView was run on the student's Pentium III laptop, and here performance began to be a consideration. Here, it runs at 2fps, which is acceptable for this application, but lower performance would not be. Surprisingly, the three Phidget RFID readers accounted for a substantial portion of the laptop's processing time.

Within Papier-Mâché, the processing is, not surprisingly, bound by the image processing computations. These performance numbers should be considered lower bounds on performance, as our image processing code is entirely unoptimized.

8 Conclusions

We have presented a toolkit for building vision-based tangible interfaces. Our event abstractions shield developers from having to get "down and dirty" working with cameras. The toolkit is open-source, written in Java, and available for download at *anonymized URL*. Applications can be built with real camera input, or they can be prototyped with a Wizard of Oz image slide show. The two class projects built using our system show how vision-based applications can be built by developers without vision experience. The primary contribution of this paper is the introduction of high-level events for vision-based TUIs.

We are continuing to analyze the results of our survey with nine TUI researchers. Our next major goal in the Papier-Mâché project is to create an event layer that has some independence from the underlying technology, enabling developers to experiment with this technology. For example, a developer may be unsure whether vision or RFID is a more appropriate technology for their application. Our goal is to make it easy to switch between the two. The exact design of this event layer will be based on the results of our survey. With our vision system, we plan to incorporate support for high-resolution still cameras (still cameras do not have a video driver). This will be useful for applications that incorporate document capture.

9 Acknowledgements

We thank Jen Mankoff, Terry Winograd, and Marc Davis for excellent discussions on this work. Thanks to Jimmy 'Lincoln Highway' Lin for feedback on drafts. We thank Chris Beckmann, Gary Hsieh, and Edward De Guzman for building applications with Papier-Mâché. A huge shout out to the nine researchers who participated in our

survey. As Isaac Newton wrote in 1676, "If I have seen further, it is by standing on the shoulders of giants."

10 References

1 *Java Advanced Imaging*, 2003. Sun Microsystems, Inc. http://java.sun.com/products/java-media/jai/

2 *Java Media Framework*, 2002. Sun Microsystems, Inc. http://java.sun.com/jmf

3 Ballagas, R., M. Ringel, M. Stone, and J. Borchers, iStuff: a physical user interface toolkit for ubiquitous computing environments. *CHI Letters*, 2003. **5**(1): p. 537 - 544.

4 Bradski, G. Open Source Computer Vision Library, 2001. http://www.intel.com/research/mrl/research/opencv/

5 Brave, S., H. Ishii, and A. Dahley. Tangible Interfaces for Remote Collaboration and Communication. In Proceedings of *The 1998 ACM Conference on Computer Supported Cooperative Work*. Seattle, WA: ACM Press, 14-18 November, 1998.

6 Crampton-Smith, G., The Hand That Rocks the Cradle, *ID* pp. 60-65, 1995.

7 De Guzman, E. and G. Hsieh, *Function Composition in Physical Chaining Applications*. Technical, UC Berkeley, Berkeley, CA, April 14 2003. http://inst.eecs.berkeley.edu/~garyh/PhysicalMacro/cs260paper.pdf

8 Dourish, P., *Where the action is: the foundations of embodied interaction*. Cambridge, Mass.: MIT Press. x, 233 pp, 2001.

9 Everitt, K.M., S.R. Klemmer, R. Lee, and J.A. Landay, Two Worlds Apart: Bridging the Gap Between Physical and Virtual Media for Distributed Design Collaboration. CHI 2003, Human Factors in Computing Systems, *CHI Letters*, 2003. **5**(1).

10 Fails, J.A. and D.R. Olsen, A Design Tool for Camera-based Interaction. *CHI Letters*, 2003. **5**(1): p. 449-456.

11 Forsyth, D.A. and J. Ponce, *Computer Vision: A Modern Approach*. Upper Saddle River, NJ: Prentice Hall. 693 pp, 2003.

12 Greenberg, S. and C. Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In Proceedings. pp. 209-18, 2001.

13 Heiner, J.M., S.E. Hudson, and K. Tanaka, Linking and messaging from real paper in the paper PDA. *CHI Letters (Proceedings of the ACM Symposium on User Interface Software and Technology)*, 1999. **1**(2): p. 179-86.

14 Hinckley, K., R. Pausch, J.C. Goble, and N.F. Kassell. Passive Real-World Interface Props for Neurosurgical Visualization. In Proceedings of *Conference companion on Human factors in computing systems*. Boston, MA: ACM Press. pp. 232, April, 1994.

15 Horn, B., *Robot vision*. MIT Press ed. The MIT electrical engineering and computer science series. Cambridge, MA: MIT Press. x, 509 pp, 1986.

16 Ishii, H. and B. Ullmer. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In Proceedings of *ACM Conference on Human Factors in Computing Systems: CHI '97*. pp. 234–241, 1997.

17 Klemmer, S. Twenty-four tangible interfaces inspiring Papier-Mâché, 2003. http://guir.berkeley.edu/papier-mache/quals/TwentyFourApps-table.pdf

18 Klemmer, S.R., M.W. Newman, R. Farrell, M. Bilezikjian, and J.A. Landay, The Designers' Outpost: A Tangible Interface for Collaborative Web Site Design. The 14th Annual ACM Symposium on User Interface Software and Technology: UIST2001, *CHI Letters*, 2001. **3**(2): p. 1–10.

19 Klemmer, S.R., M. Thomsen, E. Phelps-Goodman, R. Lee, and J.A. Landay, Where Do Web Sites Come From? Capturing and Interacting with Design History. CHI 2002, Human Factors in Computing Systems, *CHI Letters*, 2002. **4**(1).

20 Mackay, W.E., A.L. Fayard, L. Frobert, and L. MeDini. Reinventing the familiar: exploring an augmented reality design space for air traffic control. In Proceedings. pp. 558-65, 1998.

21 Mankoff, J., S.E. Hudson, and G.D. Abowd, Providing Integrated Toolkit-Level Support for Ambiguity in Recognition-Based Interfaces. *CHI Letters*, 2000. **2**(1): p. 368-375.

22 McGee, D.R., P.R. Cohen, R.M. Wesson, and S. Horman, Comparing paper and tangible, multimodal tools. *CHI Letters*, 2002. **4**(1): p. 407 - 414.

23 Moran, T.P., E. Saund, W. van Melle, A. Gujar, K. Fishkin, and B. Harrison. Design and Technology for Collaborage: Collaborative Collages of Information on Physical Walls. In Proceedings of *Symposium on User Interface Software and Technology 1999*: ACM Press. pp. 197–206, November, 1999.

24 Myers, B., S.E. Hudson, and R. Pausch, Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, 2000. **7**(1): p. 3-28.

25 Myers, B. and M.B. Rosson. Survey on User Interface Programming. In Proceedings of *SIGCHI'92: Human Factors in Computing Systems*. Monterrey, CA: ACM Press. pp. 195-202, May 3-7, 1992.

26 Myers, B.A., A new model for handling input. *ACM Transactions on Information Systems*, 1990. **8**(3): p. 289-320.

27 Newman, M.W. and J.A. Landay. Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. In Proceedings of *Designing Interactive Systems: DIS 2000*. New York, NY. pp. 263–274, August 17–19, 2000.

28 Sellen, A.J. and R. Harper, *The myth of the paperless office*. Cambridge, Mass.: MIT Press. xi, 231 pp, 2002.

29 Snibbe, S.S., K.E. MacLean, R. Shaw, J.B. Roderick, W. Verplank, and M. Scheeff, Haptic Metaphors for Digital Media. *CHI Letters*, 2001. **3**(2).

30 Szekely, P. Retrospective and challenges for model-based interface development. In Proceedings of *Eurographics Workshop on Design, Specification and Verification of Interactive Systems*. pp. 1-27, 1996.

31 Ullmer, B. and H. Ishii, Emerging frameworks for tangible user interfaces. IBM Syst. J. (USA), *IBM Systems Journal*, 2000. **39**(3-4): p. 915-31.

32 Underkoffler, J., B. Ullmer, and H. Ishii. Emancipated Pixels: Real-World Graphics in the Luminous Room. In Proceedings of *SIGGRAPH 1999*. Los Angeles, CA: ACM Press. pp. 385–392, August, 1999.

33 Wellner, P., Interacting with Paper on the DigitalDesk, *Communications of the ACM*, vol. 36(7): pp. 87–96, 1993.