*Cite as:*
Weigel, M., Boring, S., Steimle, J., Marquardt, N., Greenberg, S. and Tang, A. ProjectorKit: Easing the Development of Interactive Applications for Mobile Projectors. Report 2013-1041-08, Dept. Computer Science, University of Calgary, Calgary, AB Canada T2N 1N4

# ProjectorKit: Easing the Development of Interactive Applications for Mobile Projectors

**[12]Martin Weigel, [3]Sebastian Boring, [2]Jürgen Steimle, [1]Nicolai Marquardt, [1]Saul Greenberg & [1]Anthony Tang**

[1]Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, Alberta Canada T2N 1N4

[2]Max Planck Institute of Informatics
Cluster of Excellence MMCI
Campus E 1.7
66123 Saarbrücken, Germany

[3]Department of Computer Science
University of Copenhagen
Njalsgade 128, Bldg. 24, 5th floor
2300 Copenhagen S, Denmark

[mweigel, jsteimle]@mpi-inf.mpg.de, sebastian.boring@diku.dk, [nicolai.marquardt,saul,tonyt]@ucalgary.ca

## ABSTRACT
Researchers have developed interaction concepts based on mobile projectors. Yet pursuing work in this area – particularly in applying projector-based techniques within an application – is a cumbersome and time-consuming. To mitigate this problem, we generalize existing interaction techniques using mobile projectors. First, we identified five interaction primitives that serve as building blocks for a large set of applications. Second, these primitives were used to derive a set of principles that inform the design of a toolkit that ease and support software development for mobile projectors. Finally, we implemented these principles in a toolkit, called ProjectorKit, which we contribute to the community as a flexible open-source platform.

## Author Keywords
Projection Mapping, Mobile Projector, Toolkit, Framework.

## ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI).

## INTRODUCTION
In recent years, small mobile projectors have been increasingly used as an input / output device, where their position in the 3D physical environment affects how the interface responds. The problem is that today's user interface programming tools are typically optimized for 2D GUIs, which does not map well to the 3D projection environment. Using mobile projectors as part of an interface require a different set of development tools, including functionality that reacts to the projector position and that correctly projects its scene on a physical surface. Because there is a lack of tools, interactive applications for mobile projectors are typically developed from scratch – a time-consuming and complex task. While some toolkits for developing 3D real-world interfaces are emerging [5], they do not yet offer support for mobile projection.

To remedy this problem, we pursued the goal of developing a toolkit that simplifies how interaction designers and researchers can rapid prototype applications incorporating mobile projectors. We began by analyzing prior work in the area, where we identified five conceptual interaction primitives with mobile projectors. In turn, we derive a set of generic design principles for such toolkits, which address fundamental projection challenges that underlie all applications. We also present a general set of semantically meaningful events that encapsulate interactive functionality. Finally, we implemented these principles as the open-source ProjectorKit. Details are described below.

## INTERACTION PRIMITIVES OF MOBILE PROJECTIONS
We began by analyzing prior research on mobile projector interaction, where our goal was to identify a set of common interactions. Our set is largely inspired from [7], yet focuses on activities to yield primitives that can be operationalized as design principles of the toolkit.

*Project.* The most basic purpose of a projector is to project content onto a surface, where projection is constrained and bound by how the user is holding it. Ideally, projection creates a display on any surface and at variable sizes, where it is typically visible to multiple people. For example, a user may project pictures or movies onto a wall to show them to friends. Projectors can move, and this can be used as a form of input. For example, the projector's motion can influence the projected content as a motion beam [9] to steer a virtual character.

*Augment.* Projectors can also serve to augment real-world objects by projecting additional digital information onto them. Virtual contents are bound to the real-world object (projection mapping), where content is revealed when the user projects onto the object(s). For example, a mobile projection could show up-to-date information on a flight ticket, such as the expected delay of the flight [6].

*Select.* The projector can act as a selection device, i.e., where the user can select a virtual item to perform an activity bound to that item. For example, selecting a virtual button can trigger an event. Selection can be performed in various ways: by targeting with the projector [1, 8]; by directly touching a portion of the projection surface [10]; or by interacting in the light beam [3].

*Command.* People want to interact with the shown information by executing commands. Examples include zooming, rotating, and moving content. Those commands can be executed through item selection, as described above, or with gestures, which are performed with the projector or the object that is projected upon [4].

*Share.* Sharing allows multiple users to combine and exchange information. For example, two people can overlap two projections of calendar events to merge them, or use overlap to swap pictures in a shared workspace [2].

## TOOLKIT DESIGN PRINCIPLES

We then used these five interaction primitives to derive a set of design principles that would ultimately inform the design of our toolkit. The first two encapsulate the base functionality for a projection's visual appearance. The last three define a projection's interactive behavior as three fundamental events.

### Handling a Projection's Visual Appearance

*Project: Correcting for Jitter and Keystone Effects.* Because of their portable nature, mobile projections suffer from two fundamental effects that impact the image quality. (1) Jitter occurs due to the natural hand tremor. (2) Keystone effects appear when the axis of projection is not perpendicular to the projection surface. To counter this (and thus improve a projection's image quality), a toolkit should automatically remove jitter by stabilizing the image and remove keystone effects by warping the image (Figure 1a).

*Augment: Automated Mapping of Projection.* Mobile projectors can be used to augment real-world objects. Such augmentation requires that projections are correctly shown on their surfaces, i.e. by mapping them as virtual textures onto them. At the same time, these textures should remain correctly positioned, regardless of the projector's movement (which will change the spatial relationship between the projector and that object). Implementing such functionality requires both objects and the projector to be represented in a 3D world. That is, correct imagery can be calculated from their respective poses and by 3D raytracing (Figure 1b).

The low level programming to perform both of the above is complex. Thus a toolkit should include this low-level functionality, where it is presented as an abstraction layer to ease implementation. For example, through automated projection mapping, developers should be able to easily bind textures onto real-world objects. The toolkit itself would then automatically render and project that texture correctly from the projector's perspective.

### Select: Hotspot and Targeting Events

Hotspot and targeting events are a semantically meaningful abstraction layer for the "selection" interaction primitive, where can be implemented using proxemic properties.

*Hotspots* define an interactive area on a visual texture (Figure 1c, top). Hotspot areas are activated when they are fully or partially inside the projection. If a hotspot area is activated by the projection, a corresponding event is triggered. This allows application developers to easily implement interactive behaviors on projected textures, e.g. a button, a menu item or animations that start when the object is in the projection.

In contrast, a *targeting event* tells the developer which texture or real-world object (or which part of it) is currently targeted and therefore selected by the projection (Figure 1c, bottom). However, developers must be able to specify what exactly the targeting area is, i.e., the entire projection, or just a specific part of it, or even a single pixel.

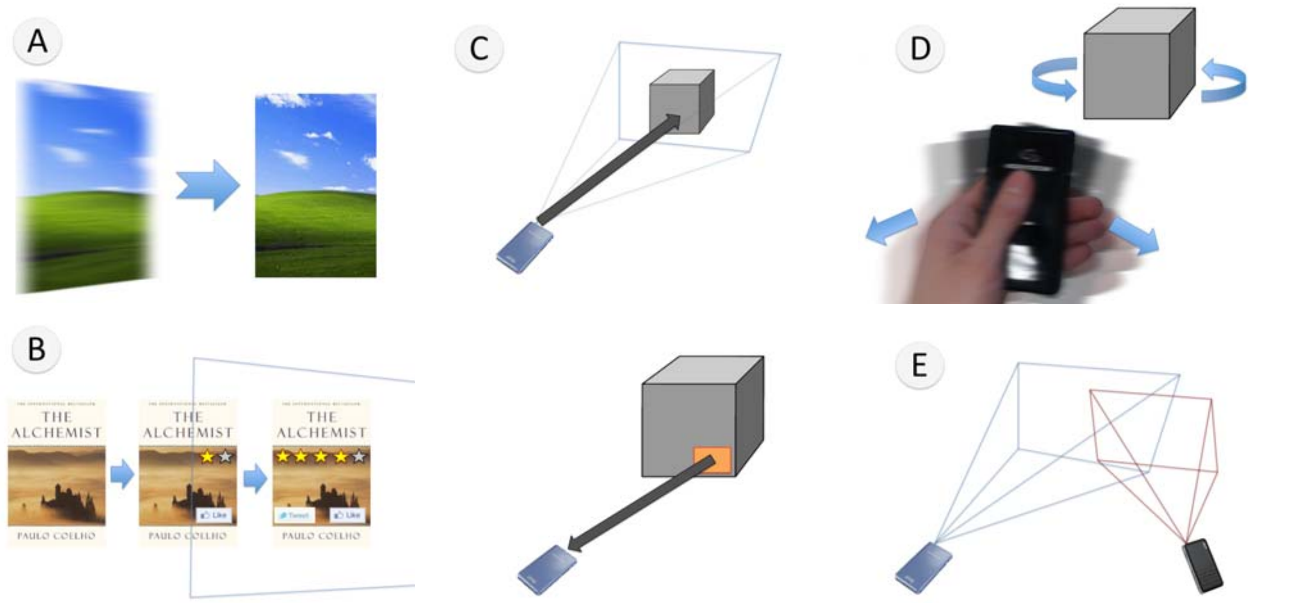

**Figure 1. Five toolkit design principles derived from the interaction primitives.** (A) Improving projection quality, here keystone and hand jitter; (B) Automated Projection Mapping, where textures are bound to a book and are revealed as soon as they are (partially) in the projection; (C) Targeting (top) and hotspot (bottom) events; (D) Gesture events with the projector and object; and (E) Overlapping Events.

*Proxemic properties* are essential to implement both event classes, as they can specify or modify the selection event. Proxemic properties comprise the distance and orientation between the projector and the projection surface. For example the distance between projector and projection surface can be used to adjust the level of detail of projected contents [1]. Orientation can adapt the perspective of a rendering [9]. Given those proxemics and the intrinsic parameters of the projector, further *projection properties*, such as the pixel density on the surface, should be provided. Table 1 shows a basic set of properties which should be supported by the event system.

### Gesture Events

*Commands as gestures* are one way to execute commands. A toolkit should automatically trigger events for a set of basic gestures, e.g., shaking, wiping and rotating. These gestures are performed either with the projector itself or with objects the user is projecting on (Figure 1d).

### Display Overlapping Events

*Sharing via overlap.* Multiple projections can overlap in various ways. A projector can overlap another projector, or a projector can overlap an active screen (e.g. a wall-sized display. Overlap can be full or partial (Fig. 1e). Overlapped regions can be exploited as a common display space, e.g., to merge and share information. An event system should trigger events when overlapping of two or more projections starts, changes or ends. The events should contain information about the proxemics of each participating projection and about the geometry of the overlap.

| Events | Properties |
|---|---|
| Gestures | Object, start, duration, used area. |
| Hotspot | Projector, distance, rotation, pixel density, visible part of the hotspot. |
| Overlap | Surface, overlapping coordinates, area size.<br>*For both displays:* overlapping pixels, average pixel density, number of overlapping pixels, percentage of the display, displayed content<br>*For projections*: distance, rotation, intersection angles, max/min pixel density. |
| Targeting | Target, distance, rotation, intersection angles, targeting pixels, pixel density. |

**Table 1. Basic properties of the events.**

## PROJECTORKIT IMPLEMENTATION

We developed a toolkit that implements our design principles, which developers can use to create applications in any of the .NET languages. The developer first creates a *server application*, which imports the toolkit and will hold the application logic. In a one-time procedure, the developer then registers physical objects and surfaces that are present in the scene. The toolkit automatically manages the model of the virtual scene, tracks moveable objects, triggers events for the application logic, and updates the clients. A lightweight *client application* that is part of the toolkit then runs on mobile devices to render the visualization that is going to be projected.

Internally, the implementation relies on the Proximity Toolkit [5] for tracking the spatial relationships between physical objects and projectors. It currently supports infrared marker tracking with Vicon and OptiTrack systems; these systems are nowadays widely available in academia and industry and can be effectively used for prototyping purposes. Since the tracking system is modular, other tracking approaches (e.g., depth sensors) can be implemented and used without modification of the toolkit or the applications. The current implementation only supports planar surfaces in the scenery.

The following running example shows how a rich projector-based interaction on an augmented book can be implemented by using all prior described design principles. A texture on the book contains a zoomable map that changes its zoom level with the distance to the projector. Shaking the book changes the texture to show different information, like reviews and related books in the users library. Overlapping two projections on the book combines the related books of both users in one sorted list.

### Adding a Texture to a Real-World Object

The following code snippet maps a texture onto a region of a real-world object, for example a book that is tracked by the tracking system:

```
var book = env.World.Get(“Book”);
// Load image with size 2000x1600mm
var image = new ImageElement(2000,1600, @"image.jpg");
image.PositionOn(book, 0, 0);
image.SetRelativeTo(book);
env.World.Add(image);
```

The texture is mapped in three steps: (1) positioning it on the book; (2) setting it relative to the book so that it stays at the position when the book changes; and (3) the texture is added into the world to be automatically shown when it is inside the projection.

### Reacting on Projector Distance for Semantic Zooming

A picture that is part of the books texture shows the location of the books’ story on a map. A hotspot event indicates the visibility of the map and provides proxemic information (distance) between the projection device and the surface. This is used here to control zooming to show map detail as the projector is moved closer to the map.

```
var hotspot = new Hotspot(image,mapregion, projector);
hotspot.Changed += HotspotEvent;
void HotspotEvent (object sender, HotspotEventArgs e)
{
   if(e.Distance < 200) // Distance in mm
       // Zoom with the distance
}
```

### Registering a Shake Gesture

Shaking the book is used to change the information that is displayed on its cover. The following code binds a 2 second shake gesture to the book.

```
var shaking = new ShakeGesture(book, 2.0);
shaking.Recognized += ShakingEvent;
void ShakingEvent (object sender, ShakingEventArgs e)
{
   // Code to change the cover information
}
```

### Creating a Common Workspace

When discussing the book with other users, multiple overlapping projections can combine friends who liked the book or show related books that both users have read.

The toolkit supports overlapping workspaces, while at the same time also avoids potential interference of one projection on other projections or active displays. It does so through two techniques. The first technique blacks out the complete overlapping area (or specific parts) of one projection, i.e., so that the other projection displays without interference in that area. The following example illustrates this, where it considers the overlap of a mobile projector and a fixed display (such as a wall-sized display). It calculates which would have the lower resolution of the two (which depends upon the distance of the projector from the display), and blacks that one out dynamically. The effect is that the projector can serve as a high-resolution magic lens when brought close to the fixed display. However, the code below only shows how the lackout region is determined.

```
var overlap=new OverlappingDisplays(projector,display);
overlap.OverlappingChanged += OverlappingEvent;
void OverlappingEvent(object s, OverlappingEventArgs e)
{
  if (e.Display1.PixelDensity<=e.Display2.PixelDensity)
     e.Display1.BlackoutOverlapWith(Display2);
  else
     e.Display2.BlackoutOverlapWith(Display1);
}
```

A second technique to handle interference is to black out only specific parts of the overlap area rather than the whole regions. While we do not show an example of its use, it is implemented in the toolkit and only somewhat more complex than the above example.

## USER FEEDBACK

We tested and collected feedback in two prototyping workshops with 7 computer scientists (6m, 1f; median age 25) and two longer-term projects with 3 developers (2m, 1f; median age 24). Participants developed a wide variety of applications for mobile projectors, including a multi-user game, a book browser, projection of personal content on a public display and a map interface with focus plus context behavior. Each of these applications was implemented with a small number of statements (less than 100). All projects could be realized without any major obstacles.

Overall, we received very positive feedback from the participants. All participants appreciated the ease of augmenting the environment and the possibility of using events to easily respond to projector-based interactions. One participant suggested that providing dedicated UI controls and labels would further ease application development. Another participant reported that better support for visual debugging of the 3D environment would be helpful.

## CONCLUSION

Based on an analysis of prior work, we presented principles for the design of toolkits that ease developing interactive applications that make use of mobile projectors. We used these principles to design and implement a software toolkit, ProjectorKit.

**Acknowlegements.** Funding was provided in part by the NSERC SURFNET Network Grant, and by that AITF/NSERC/SMART Technologies Industrial Chair.

## REFERENCES

1. Cao, X. and Balakrishnan, R. Interacting with dynamically defined information spaces using a handheld projector and a pen. In *Proc. UIST’07.*
2. Cao, X., Forlines, C., and Balakrishnan, R. Multi-user interaction using handheld projectors. In *Proc. UIST’07.*
3. Cowan, L. and Li, K. ShadowPuppets: supporting collocated interaction with mobile projector phones using hand shadows. In *Proc. CHI '11.*
4. Huber, J., Steimle, J., Liao, C., Liu, Q., and Mühlhäuser, M. LightBeam: Interacting with Augmented Real-World Objects in Pico Projections. In Proc MUM’12.
5. Marquardt, N., Diaz-Marino, R., Boring, S., and Greenberg, S. The Proximity Toolkit: Prototyping proxemic interactions in ubiquitous computing ecologies. In *Proc. UIST’11.*
6. Mistry, P., Maes, P., and Chang, L. WUW - wear Ur world: a wearable gestural interface. In *Proc. CHI EA'09.*
7. Rukzio, E., Holleis, P., and Gellersen, H. Personal projectors for pervasive computing. *Pervasive Computing, IEEE, 11*(2), 30-37.
8. Schmidt, D., Molyneaux, D., and Cao, X. PICOntrol: Using a Handheld Projector for Direct Control of Physical Devices through Visible Light. In *Proc. UIST’12.*
9. Willis, K., Poupyrev, I., and Shiratori, T. Motionbeam: a metaphor for character interaction with handheld projectors. In *Proc. CHI '11.*
10. Winkler, C., Reinartz, C., Nowacka, D., and Rukzio, E. Interactive phone call: synchronous remote collaboration and projected interactive surfaces. In *Proc. ITS' 11*