ProjectorKit: Easing Rapid Prototyping of Interactive Applications for Mobile Projectors

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

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

[2]Max Planck Institute for 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 building projector-based interactions techniques within an application—is cumbersome and time-consuming. To mitigate this problem, we contribute *ProjectorKit,* a flexible open-source toolkit that eases rapid prototyping mobile projector interaction techniques.

Author Keywords
Mobile projectors, toolkit, rapid prototyping.

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 input/output devices, where their position in the 3D physical environment affects how the interface responds. The problem is that today's programming environments for user interfaces are typically optimized for 2D GUIs. They do not map well to the 3D projection environment. Using mobile projectors as part of an interface requires a different set of development tools, including functionality that reacts to the projector's position/orientation so that the projector correctly projects its scene onto a physical surface. With the lack of such tools, interactive applications for mobile projectors are typically developed from scratch to replicate old work—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, our goal was to develop a toolkit that simplifies how interaction designers and researchers can rapidly prototype applications incorporating mobile projectors. First, we articulate a small set of requirements that specify programming and interaction aspects common to mobile projector usage. Second, we discuss the design of *ProjectorKit*, which realizes these requirements. Third, we provide a concrete example to illustrate a sample *ProjectKit* program. We then discuss how feedback from a small group of programmers led to *ProjectorKit*'s improvements, along with limitations.

REQUIREMENTS FOR A MOBILE PROJECTOR TOOLKIT
Inspired by Rukzio et al. [9], we identified a basic set of five interaction requirements for mobile projectors that collectively describe a set of basic roles played by mobile projectors. These requirements describe functionality that should be supported by a toolkit to ease development of similar interaction scenarios. They encapsulate a projection's visual appearance, and define a projection's interactive behavior.

Project. The most basic purpose of a projector is to project content onto a surface, where projection is bound by how the user is holding it. It creates a display on any surface, which is visible to multiple people (e.g., a video projected onto a wall). The projector's position and motion alters the appearance of the projection: getting closer to the projection surface decreases the projection's size and vice versa.

R1: The toolkit should correct for jitter and keystone effects (Fig. 1a). Mobile projections suffer from two fundamental effects that impact image quality: (1) jitter occurs due to the natural hand tremor; and (2), keystone effects appear when the axis of projection is not perpendicular to the projection surface.

Augment. Projectors can augment real-world objects by projecting additional digital information onto them [8]. 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 projector could project up-to-date information on a physical flight ticket, such as the expected delay of the flight [6].

R2: The toolkit should provide automated projection mapping of the virtual content (Fig. 1b). Augmentation requires anchored projections to be rendered "correctly" regardless of the projector's orientation relative to the projection surface. These textures should remain anchored, correctly positioned, and correctly displayed relative to the physical world regardless of the projector's movement (which will change the spatial relationship between the projector and that object, as well as the projection surface).

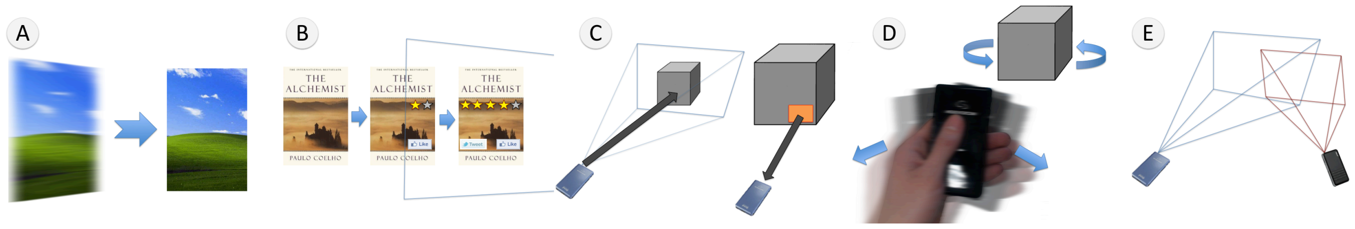

Figure 1. Toolkit requirements. (A) Improving projection quality, here keystone and hand jitter; (B) Automated projection mapping; (C) selection events; (D) Gesture events with projectors or objects; and (E) overlapping events.

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, highlighting a virtual button can trigger an event. Selection can be performed in various ways: by targeting with the projector [1,10]; by directly touching a portion of the projection surface [11]; or by interacting in the light beam [3].

R3: The toolkit should provide selection-events (Fig. 1c). These events can be defined in two ways. First, specifying virtual content allows for events being triggered whenever a projector fully or partially reveals that content. For example, animations may start once that content is visible in the projection area. Second, specifying a target area within the projection area (i.e., the entire projection, just a specific part of it, or a single pixel) should trigger an event telling the developer what virtual content/real-world object is targeted (i.e., selected) by the projection. For example, the region of a projected mouse pointer can be used to select virtual content.

Command. Beyond merely highlighting and shining a mobile projector around an area, a user should be able to interact with its information by executing commands, such as zooming, rotating, and so forth. One way to allow for such commands is via a user performing gestures with the projector or the object projected upon [4].

R4: The toolkit should automatically interpret basic gestures, and provide simple notification of these events (Fig. 1d). Basic motion-based gestures should be recognized, such as shaking, wiping and rotating. These gestures could be performed over the projector, or on other augmented physical objects.

Sharing. Projections may share the same projection area, overlapping with one another. For example, such sharing allows people to overlap their individual projections of calendar events to merge them, or use overlapping regions to transfer pictures in a shared workspace [2].

R5: The toolkit should provide notification of overlapping projection/display areas (Fig. 1e). Multiple projections can overlap in various ways. Overlapped regions can be exploited, e.g., as a common display space to merge and share information. One projection area can overlap another one, or it may overlap an active screen (e.g., a wall-mounted display). Overlaps may be full or partial.

PROJECTORKIT IMPLEMENTATION

Based on these requirements, we designed and developed *ProjectorKit,* a .NET toolkit for rapid prototyping of mobile projector applications. Our toolkit leverages the proxemics information provided by the Proximity Toolkit [5]: spatial relationships between entities in the environment (people, projectors, displays, projectors, tracked objects), as well as fixed displays and surfaces. The Proximity Toolkit currently supports infrared marker tracking with Vicon and OptiTrack systems. In principle, the "output" of such a system might be similar to prior work using a single fixed projector with a moving arm [7]; however, the mental model provided to programmers by *ProjectorKit* is fundamentally different, allowing programmers to think of interactions with mobile projectors as first-class ideas rather than focusing on interaction of mobile projectors with what can be displayed in the environment.

Client-server architecture. We based *ProjectorKit* on a client-server architecture. The server application imports the toolkit and hosts the application logic and system state (i.e., it contains the *model* and *controller* of the MVC design pattern). In a one-time procedure on the server-side, a developer registers intrinsic parameters of all projectors and displays, and the physical objects and surfaces present in the scene. Each projector is driven by a client, rendering the projected visuals based on the properties of that particular projector (i.e., the application *view* of the MVC pattern). A prebuilt client supplied by the toolkit supports many standard uses without modification, as the client is only responsible for rendering the view. All components communicate via Wi-Fi. While others have developed standalone clients on a mobile device, the client-server architecture is more amenable to the rapid design-develop-deploy-test cycle. Rather than having to move code to different clients (driving a mobile projector), a developer can simply modify the code on the server, and get the same interaction effects on the clients without modification. As well, the server manages and tracks the spatial relationships once rather than having each client to do that individually.

Virtual scene. Based on information from the Proximity Toolkit, the toolkit automatically manages a model of the virtual scene that maps the physical environment, tracking moveable entities such as mobile projectors and objects as they move through the environment. What the projectors display is represented in this virtual scene as multiple textures that are mapped on various surfaces. For example, the wall may have a texture of a map that is shown in the real world only when the projector is shone against the wall. Similarly, a tracked physical object may have a paired virtual texture (containing additional information) that follows the object in the virtual world.

Developers add, retrieve and remove both named and anonymous objects to and from the scene using simple function calls (e.g, `add, get, remove)`. The correct imagery for each projector is displayed by rendering the textures and contained objects of the virtual scene with a perspective camera with the projectors' extrinsic and intrinsic parameters. The projection of this rendering shows all textures properly mapped in the physical world (R2). The programmer therefore does not have to consider image correction, or what is visible to the projector; instead, image is rendered automatically to the projector.

Keystone effects and jitter are automatically removed from projections using a margin around its interface. Depending on the size of this margin, the projection is automatically corrected without resizing and repositioning of the controls.

Event-driven programming. The toolkit performs ray-tracing calculations based on the spatial relationships between the different entities in the scene. Each time the Proximity Toolkit sends a tracking update, *ProjectorKit* automatically updates the scene model, and determines whether the programmer should be notified of semantically "interesting" events (R3–5). Selection events are fired when a projector's projection frustum overlaps with a target area (R3). Similarly, a gesture event that has been subscribed to by a developer is fired when one is detected (R4). Finally, if projections overlap between projectors, or if a projection overlaps with a fixed display, the overlap event is fired (R5). Additional interaction-events, e.g. body gestures and selection using touch or interaction with the light beam can be implemented using these events.

Each event has three different parts, i.e. `started`, `changed`, and `ended`. Each can be bound to event handlers, much like for touch events in a multi-touch programming environment. *ProjectorKit* calculates intersections between projector pixels and objects in the virtual scene for each tracker update, which determine which events should be fired. Because *ProjectorKit* relies on a tracking infrastructure, errors are mainly introduced by the calibration and accuracy of the tracking infrastructure. In practice, given a set of tracking cameras, small tracking volumes produce very little error, but errors increase as tracking volume increases in size. This mostly influences the overlapping events. In our setup we experienced an overlapping error around 0.5-7cm. As mentioned, better calibration could reduce this error.

Event properties. Table 1 shows the basic set of proxemic and projection properties that are automatically calculated by *ProjectorKit*'s event system. The proxemic properties − the *distance* and *orientation* between the projector and the projection surface − are essential to implement all event classes. For example, the distance between projector and projection surface can be used to adjust the level of detail of projected contents [1], while orientation can adapt the perspective of a rendering [10]. From these proxemic properties, additional projection properties are derived using the intrinsic parameters of the projector, i.e. resolution, opening angle, and center pixel. For example, the average pixel density of the projection on the surface can be used to adjust the level of detail of a texture.

Events	Properties
Selection	Projector, target, distance, rotation, intersection angles, targeting pixels, pixel density, visible part of the target.
Gestures	Object, start, duration, used area.
Overlap	Surface, overlapping coordinates, area size. *For both displays:* display, overlapping pixels, average pixel density, number of overlapping pixels, percentage of the display, displayed content *For projections*: distance, rotation, intersection angles, max/min pixel density.

Table 1. Proxemic and projection event properties that are automatically calculated by the event system.

Performance. In our setup, the applications of the toolkit had latency of 10-100ms. This depends primarily on the network and the rendering performance of the projectors.

EXAMPLE

The example in Listings L1 and L2 is written in the .NET language C#. It shows projector-based interactions on an augmented fantasy book. It contains a detail-on-demand map, friend's recommendations and a list of related books. It only requires a server application that uses the previously described design principles and the prebuilt client. The application developer maps an image map of the fantasy world as a texture onto the book and sets it at a fixed relative position onto the book (L1 line 3-4). The toolkit continuously adapts the texture's position and orientation such that it stays at a stable position on the book even when location or orientation of the book changes.

The book contains a detail-on-demand map that shows additional detail when the projector is close. Hence, a selection-event is registered (L1 line 6-7) to indicate the visibility of the map by calling the *SelectionEvent*-method (L2 line 1-3). The event provides proxemic information between the projection device and the surface; in this case their respective distance. We use this to adjust the map's level of detail, i.e., show or hide small villages, when the projector moves closer to/further away from the book. Shaking the book for half a second changes the texture to show different information, like reviews and related books in a person's library (L1 line 8-9 and L2 line 4-5).

When discussing the book with other people, overlapping projections can show a combination of friends who liked the book or show related books that these people have read.

```
 1 var book = env.World.Get("Book");
 2 // Load image with size 2000x1600mm
 3 var image = new ImageElement(2000, 1600, @"image.jpg");
 4 image.PositionOn(book, 0, 0);
 5 env.World.Add(image);
 6 var selection = new Selection(image, region, projector);
 7 selection.Changed += SelectionEvent;
 8 var shaking = new ShakeGesture(book, 0.5);
 9 shaking.Recognized += ShakingEvent;
10 var overlap = new OverlappingDisplays(projector, p2);
11 overlap.OverlappingChanged += OverlappingEvent;
```

Listing 1. Setup of the augmented interactive book.

```
 1 void SelectionEvent(object sender, SelectionEventArgs e)
 2 {   if (e.Distance < 200) // Distance in mm
 3           /* Show additional detail */ }
 4 void ShakingEvent (object sender, ShakingEventArgs e)
 5 { /* Code to change the texture information */ }
 6 void OverlappingEvent(object s, OverlappingEventArgs e)
 7 { if (e.Display1.PixelDensity<=e.Display2.PixelDensity)
 8       e.Display1.BlackoutOverlapWith(Display2);
 9   else
10       e.Display2.BlackoutOverlapWith(Display1);
11   /* Combine views of two projectors */ }
```

Listing 2. Methods to interact with the book example.

We avoid overlapping interference through an *OverlappingDisplays*-event that blacks out the overlapping area in the lower-density projection (L1 line 10-11 and L2 line 6-11). This pixel density depends upon the distance (further away means lower density) of the projector to the surface and the projectors intrinsic parameters.

DISCUSSION

Early feedback. We gathered feedback on an earlier version of our system in two prototyping workshops and two longer-term projects. Participants successfully implemented a variety of applications using the toolkit, each with less than 100 lines of code, 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. All applications were implemented without any obstacles.

Lessons learned. From this feedback, we modified the toolkit to provide detailed event properties. This allows programmers to focus on programming logic, instead of calculating these properties (e.g. pixel density in the projection). We also added 2D as well as 3D coordinates while dealing with textures. Novice programmers preferred simpler positioning as provided by 2D coordinates on the planar surfaces, but 3D coordinates provide high flexibility.

Limitations. The toolkit currently relies on expensive high-end tracking. In contrast to relatively tracking the position of a mobile device, these tracking systems support accurate and reliable tracking of absolute positions of mobile devices as well as physical objects. This prototypical setup allows for early explorations of new interaction concepts before such tracking is directly integrated into mobile devices. The underlying tracking system of the toolkit is modular [5], which means that other tracking approaches (e.g., depth sensors) can be reasonably incorporated in the future. *ProjectorKit* also only supports compositions of planar surfaces without too much texturing in the scenery, i.e., it does not handle projection over curved or irregular objects and does not correct for the underlying colors and textures. This too could be corrected by modeling projections on a more sophisticated model of the underlying 3D space. The client/server infrastructure requires a server and restricts on-device computations. This is a reasonable requirement in prototyping environments: it avoids conflicting states, and allows for easier debugging in multi-projector applications. Finally, applications are currently limited to any of the .NET languages along with WPF or Silverlight. This restricts the mobile devices to Windows XP/7/8 handhelds, tablets, and notebooks, e.g. the Microsoft Surface Pro. However, the lightweight client architecture and the message protocol allow for reasonable porting to other platforms.

CONCLUSION

Based on an analysis of prior work, we described five requirements for the design of toolkits that ease developing interactive applications that make use of mobile projectors. We used these requirements to design and implement a software toolkit, ProjectorKit. As a service to the community and to encourage further development of mobile projector interactions, we contribute *ProjectorKit* as an open source toolkit: http://grouplab.cpsc.ucalgary.ca/cookbook/index.php/Toolkits/ProjectorKit.

ACKNOWLEDGEMENTS

We thank NSERC SurfNet and Nokia for their support.

REFERENCES

1. Cao, X. and Balakrishnan, R. Interacting with dynamically defined information spaces using a handheld projector and a pen. In *ACM Proc. UIST'07.*
2. Cao, X., Forlines, C., and Balakrishnan, R. Multi-user interaction using handheld projectors. In *ACM 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 *ACM Proc. UIST'11.*
6. Mistry, P., Maes, P., and Chang, L. WUW - wear Ur world: a wearable gestural interface. In *ACM Proc. CHI EA'09.*
7. Pinhanez, C. The Everywhere Displays Projector: A Device to Create Ubiquitous Graphical Interfaces. In *Proc. Ubicomp'01.*
8. Raskar, R., van Baar, J., Beardsley, P., Willwacher, T., Rao, S., and Forlines, C. iLamps: geometrically aware and self-configuring projectors. In *ACM SIGGRAPH'03.*
9. Rukzio, E., Holleis, P., and Gellersen, H. Personal projectors for pervasive computing. *Pervasive Computing, IEEE, 11*(2).
10. Schmidt, D., Molyneaux, D., and Cao, X. PICOntrol: Using a Handheld Projector for Direct Control of Physical Devices through Visible Light. In *ACM Proc. UIST'12.*
11. Winkler, C., Reinartz, C., Nowacka, D., and Rukzio, E. Interactive phone call: synchronous remote collaboration and projected interactive surfaces. In *ACM Proc. ITS' 11.*