Toolkit Support for Interactive Projected Displays

John Hardy
Lancaster University
Lancaster, UK
john@highwire-dtc.com

Jason Alexander
Lancaster University
Lancaster, UK
j.alexander@lancaster.ac.uk

ABSTRACT
This paper presents a software toolkit designed to enable the rapid development of multimedia-rich, multi-touch enabled, and interactive projection-based displays. For instance: door displays, floor displays, wall displays, and interactive tables. Despite recent technological advances and the commercialization of hardware required to achieve this at a relatively low cost, creating and deploying such displays remains a difficult task, even for those with the essential technical skills and experience. We assert that greater accessibility of toolkits like the one presented in this paper will reduce these barriers and allow people (not necessarily from the ubiquitous computing domain) to apply the technology to their own fields. To assess this toolkit's suitability for this role, we present a system and usability evaluation. We observed that participants were able to quickly create their own novel display deployments. Our findings offer insights for potential toolkit users and those considering how to write programs for future ubiquitous projected display environments.

Categories and Subject Descriptors
H.5.2 [**User Interfaces**]: Input Devices and Strategies

General Terms
Design, Experimentation, Human Factors.

Keywords
Ubiquitous displays, multi-touch, interaction, experimentation, projection mapping, interactive surface, deployments, toolkit.

1. INTRODUCTION
The notion of projecting relevant, interactive digital content into the world around us is a fascinating idea with a rich history [16] [19] [20] [25]. Yet, despite being a powerful vision with much potential [14], those who wish to explore it further still face many technical and practical challenges: a potentially discouraging start for many new ideas. Indeed, using projected displays in *application driven* research [1] is a complex process that is difficult to get started, particularly in exploratory cases where the expected results cannot always justify the technology and time investment.

Over the past decade, many tools and platforms have emerged that researchers and hobbyists have used to prototype new concepts [11] [15]. While some aspects of ubiquitous computing are starting to enjoy a democratized design process [7], ubiquitous projected displays are not. As it stands, interactive projector-based displays are still too hard to create, deploy, reproduce and extend. Most of today's installations tend to be developed and tuned for a specific deployment configuration. Even building conceptually simple systems requires skilled people, custom software, time, and ultimately expense. We argue that creating and deploying a functional, visually pleasing multi-touch display should take hours, not days.

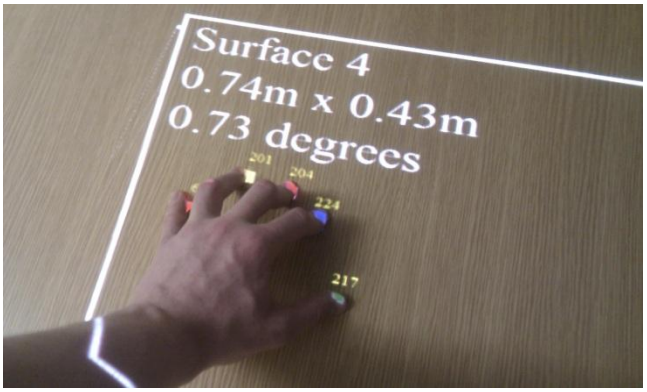


Figure 1: A multi-touch interactive display created using the presented toolkit

As a first step toward enabling more developers to implement their ideas, a general-purpose toolkit with simple developer abstractions is required. To cater to a broad range of potential applications, this must support displays with varying lifespans, geometries, interactive capabilities, and rich content formats. Making this accessible to researchers and hobbyists, the toolkit must operate with small, easily achievable hardware requirements and be robust to different physical configurations.

In this spirit of allowing developers to engage with projector-based displays, we constructed a toolkit based on these goals. This has allowed to us to begin to explore the opportunities and issues that emerge when developing for multi-display installations within a physical space.

This paper offers five main contributions: (1) A software toolkit that simplifies and expedite the process of creating interactive, multimedia rich, projected displays (see Figure 1); (2) A point cloud based multi-touch detection algorithm; (3) An accuracy profile of the toolkit under different hardware placement conditions; (4) A short-term development study where eight participants developed novel projected display applications; (5) A long term deployment of the toolkit in a real-world project.

2. RELATED WORK
At the start of the last decade, much research was motivated by the goal of supporting next-generation workspaces that featured

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MUM '12, December 04 - 06 2012, Ulm, Germany
Copyright 2012 ACM 978-1-4503-1815-0/12/12…$15.00.

interactive ubiquitous displays. This resulted in several seminal visions and many novel prototypes.

2.1 Ubiquitous Projection

In 2001, Pinhanez et al. published '*The Everywhere Displays Projector: A Device to Create Ubiquitous Graphical Interfaces*' [17]. This was a system that provided interactive projections on arbitrary planar surfaces. The prototype consisted of an LCD projector, a rotating mirror, and a camera to detect interaction. When the mirror was rotated, the projection would be cast onto nearby surfaces. Then, software transformed the projected image to correct for distortion on surfaces which did not lie planar to the projection. Using this technology as a base, they were able to apply the idea to many different situations, including retail environments, interactive games, and knowledge workers in workplaces.

In 2003, Raskar et al. introduced a new technique for adaptive projection onto curved and other non-planar surfaces [19]. Their approach used portable hardware units they called 'geometry aware projectors' to recover information about the surrounding environment; creating a conformal map that transformed the resultant projection to minimize distortion over non-planar surfaces. Using these techniques they were able to combine multiple projectors in a way that allowed them to create seamless large area displays. More recently, Microsoft's LightSpace project [28] combined projectors and depth cameras to make any physical surface in a designated space interactive. This included the user's body.

2.1.1 Depth Cameras

The Microsoft Kinect has allowed cheap access to depth-sensing cameras, sparking a flurry of ad-hoc applications from the research and enthusiast communities. Of particular interest is the use of depth sensing for detecting touch. Wilson [27] demonstrates how depth cameras can be used as touch sensors. Although the technique is not as precise as more direct sensing methods (such as capacitive touch screens where the user actually touches the sensor) there are other advantages. For instance, surfaces need not be instrumented, they need not be flat, and it is possible to extract other information about the origin of the touch and movement above the surface.

The dSensingNI framework [10] (Depth Sensing Natural Interaction) is designed to combine multi-touch and tangible interaction with arbitrary physical objects. They state that although their system is primarily designed for use with table tops, it can also be applied to vertical installations like interactive walls. Our toolkits share a common goal of simplifying the process for other developers. TESIS (Turn Every Surface into an Interactive Surface) [2] is a portable device that features a depth camera and pico-projector placed inside a lamp stand. It is capable of detecting multi-touch interaction on non-flat surfaces.

2.1.2 Handheld Projection

A promising trend in the ubiquitous display space is that of the handheld or portable projector. SideBySide [26], SixthSense [12], and OmniTouch [6] are all systems which pursue a vision of personal ubiquitous displays where the individual carries their own display hardware and sensors. In particular, OmniTouch demonstrates how it is possible to track fingers while they interact with surfaces that deform in real-time. Molyneaux et al's augmented projectors [13] demonstrates interactive content with an awareness of the surrounding geometry using both infrastructure-based and infrastructure-less sensing. Here, handheld devices equipped with depth cameras and projectors are able to compute their position and orientation relative to a 3D model of the world, and as a result, reveal interactive content within it. This can be applied to provide shared ubiquitous displays or perhaps combined with the interest in augmented reality glasses to create personal ubiquitous displays.

2.2 Ubiquitous Content

Turning their attention to issues of deployment and content support, in order to simplify the creation of graphical user interfaces, Kjeldsen et al. [9] used Pinhanez' system [17] as the basis for a software layer which decoupled the functional definition of a projected interface from its location in the physical environment. To do this they created an XML based mark-up language to specify widgets and regions for detection in a manner that could be mapped onto system functions.

Today, the demand for rich interactive media formats makes bespoke interface languages unattractive due to the unavoidable costs involved in supporting widely adopted multimedia standards. In the digital signage domain, web browsers offer a particularly inviting solution to the problem as they are cross platform, provide a multi-touch specification [23], are easy to work with, have built-in-support for most content types, programmable logic, internet connectivity, and a massive base of pre-existing community support. Clearly this makes them an attractive option. However, web browsers do not have access to the native platform and thus cannot easily be used to access the underlying hardware.

In terms of interaction, TUIO is an open framework that defines a common protocol and API for tangible multi-touch surfaces [8]. Despite being a community standard, a drawback of TUIO is that developers must implement the support at a low level.

2.3 Toolkit Innovation: Closing the Loop

To effectively develop a toolkit that supports application driven research as well as the enthusiastic hobbyist, it is important that it be designed with an appreciation of their goals.

In Abowd's commentary [1] he characterises application driven research as "*an introduction of technology into a problem domain that makes a research contribution to that domain itself.*" While applauding technologies that have gone on to be applied in this way, he reminds us that the cost of this adoption is that we, as a community, are often not party to the fruits of such research. As a domain wanting to learn from the ways and scenarios in which our new technology can be applied, this poses an interesting problem: How can a toolkit effectively capture external experience and fold it back into our own research and understanding?

In the last decade, the biggest change we have seen may not be in technology, but in its pace of development, thanks to the pervasiveness of the internet, viral media, and the access to technology it provides. We propose that viral technology transfer could become a valuable tool for feeding back information about role of ubiquitous technologies and diversity of applications experimented with during the diffusion process [21]. This would be of value to the ubiquitous computing community as it would help 'close the loop' on the 'lost' results of application driven research and expose researchers to a wide range of new use case scenarios, applications, subtleties, and contexts that may inspire and influence future directions.

3. DESIGN AND IMPLEMENTATION

This section describes the design and implementation of our toolkit. We first discuss the requirements and scope of its design,

followed by an outline of abstractions, features of its implementation, and comments on content development and interaction detection.

3.1 Requirements

Reviewing the technology that is commercially available to application driven research and enthusiastic hobbyist groups, we believe the most versatile combination of readily available hardware is a computer, a projector, and a depth camera (Microsoft Kinect). The role of a software toolkit in this space should be to configure this hardware in order to create (potentially multiple) co-located displays. These displays should support various lifespans, geometries, interactive capabilities, and content.

We use the aforementioned hardware to enable interaction and content display on uninstrumented surfaces. These projected displays are intended to be integrated into a space and managed by the people responsible for that space. The toolkit is not intended to support handheld or mobile projectors as these have varying technical constraints (battery life, etc.) and are often subject to designs involving a personal ownership model. Neither are we supporting steerable projects as these require custom hardware which is not readily available to our target demographic.

The toolkit should be robust against non-optimal hardware placement as it is not always possible to mount a projector or depth camera planar to a display surface. In terms of content support, we elected to use web standards (HTML5, CSS3, JavaScript) as they offer many of the required features and are widely adopted within target demographic. Furthermore, this avoids the requirement of learning a specialized interface language and allows developers to capitalize on a pre-existing wealth of community support and transferable skills.

3.2 Abstractions

The toolkit should support the creation of wall displays, floor displays, table displays, door displays, and any other such user-desired projected display. Such diversity requires developer abstractions that are generic, yet simple to grasp. Firstly, we distinguish between a 'surface' and a 'display'. This allows content to be decoupled from its location in the physical environment, and as such, to be added, removed, moved and modified interactively.

Surfaces – These are flat user defined areas in physical space where it is practically sensible for a display to appear. Each surface is capable of hosting a single display. Surfaces are given unique names upon creation, which are used for addressing. Each surface contains automatically computed metadata (i.e. orientation and physical size) which is accessible to the displays. Furthermore, a surface does not always have to show content and can lie dormant.

Display – Displays are responsible for rendering content in the physical environment. From a developer perspective, a display is effectively visual content and logic stored in a HTML file (either locally or remotely). We implemented displays using a specialised Webkit[1] control that supports the latest web standards. Displays can execute their own sandboxed logic in the form of JavaScript. They have the ability to perform functions such as querying the surrounding environment to find other display surfaces, and the ability to request different interaction methods. For example, a display can request a multi-touch interaction modality. Displays are not assigned an interaction modality automatically because the toolkit should be agnostic of interaction method. The loose coupling between displays and the surfaces hosting them allows display content to 'jump' between surfaces.

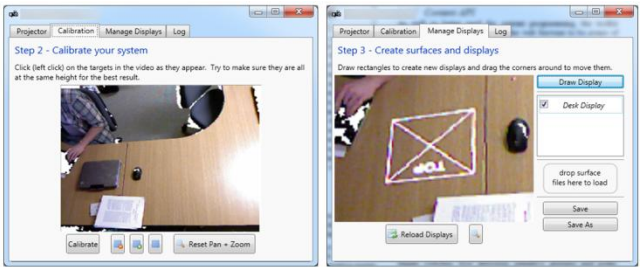
Placing content in the real world requires a means of addressing the physical space. In the scheme used by the toolkit, each surface is given a unique name which is both human readable and descriptive. For relatively small systems such as ours, we make the assumption that this is more developer friendly than a large virtual canvas. An obvious drawback of our addressing scheme is that it requires all display surfaces be pre-defined. However, given that those responsible for spaces often require a high degree of control over their management and aesthetic, manually specifying display areas ensures that the deployment remains controlled by the entity responsible for the space, rather than by passers-by who wish to display content.

3.3 Implementation

The toolkit targets the .NET platform and uses the WPF framework for the native user interface as well as the Direct3D maths library and the Microsoft Kinect SDK. It takes advantage of multi-core processor architectures by rendering each display in a separate process. To do this, it uses a specialised version of the Webkit layout engine and has out of the box support for HTML5, CSS, JavaScript, SVG, and Adobe Flash.

3.3.1 User Interface and Display Management

The toolkit is configured by a wizard-based interface on the host computer. The configuration steps are outlined below with corresponding screenshots in Figure 2.



Step 2: Calibration | Step 3: Surface and Display Management

Figure 2: The calibration and display management steps of the toolkit interface

Step 1 – Projector Selection To begin, the developer must select the projector from a list of display outputs (i.e. projectors and monitors). Once one is chosen, the program immediately jumps to the next step.

Step 2 – Interface Calibration Next, the developer calibrates the system in order to map mouse events in the coordinate system of the depth camera's video feed into that of the projected displays. This helps make the interface easier to use. During calibration, the projector will display four sequential planar calibration points which the developer must click on in the video feed. These points are then used to construct a simple homography matrix. We opted to calibrate against a single plane for simplicity, although this has the obvious drawback of not accounting for the parallax distortion of the projector lens. We made the design choice that an eight-point non-planar calibration would be too complex and unnecessary for simple display configurations.

Step 3 – Surface and Display Management Here, the user can create surfaces and deploy display content onto them. Other

[1] WebKit is a layout engine designed to allow web browsers to render web pages.

functions available in this step include: save, load, and reload all active displays.

Not all potential display surfaces lie planar and orthogonal to the projector (see examples in Figure 3). We support such surfaces by transforming the rendered display content with a non-affine transformation matrix. To make the process of constructing such a matrix transparent to the developer, it is computed using coordinates obtained when they 'draw' a display over the desired location in a live video (see Figure 2). Using the rotating calipers method [24] this drawing is then snapped to a best-fit rectangle to get the bounding corners to be used for the display.

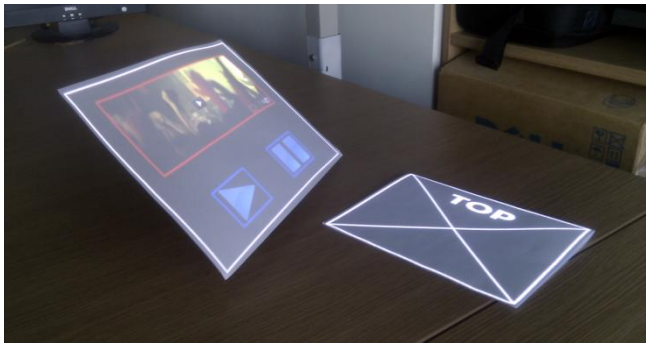


Figure 3: Two displays projected onto two flat physical surfaces which are not planar to the projector

It is possible to adjust each surface's projection and sensing corners is possible by dragging the corners of the surface with the mouse on the live video feed. This is useful in cases where the default interface calibration does not provide enough accuracy, or has been subject to drift—allowing the person deploying the system to interactively adjust the surface geometry. Surfaces with non-rectangular geometries are not supported.

As with the calibration interface, the video feed can be panned and zoomed. A list of surfaces is provided so that they can be manually edited and deleted.

3.3.2 Content API

As well as being used for programmable content, the toolkit exposes the hosting surface's physical characteristics (such as size, orientation, and relative position) to JavaScript—allowing display content to react to deployment conditions. This can be achieved simply by invoking methods on an 'Authority' object which are specified in the toolkit API. This object acts as a mediator between display logic, surfaces, and the capabilities of the toolkit. It is used in two main ways:

Firstly, content hosted by the toolkit is able to open, close, move, and swap surface with other items of content. It also allows displays to invoke JavaScript functions on other displays. For example, this could be used to make content on vertical surfaces automatically align itself to a world-up axis. Or, in an applied scenario, a button on a teacher's desk could trigger the display of interactive instructions in front of each child.

Secondly, content is able to select its own interaction modality. On a typical personal computer, content in a web browser is interacted with via keyboard and mouse. In the physical world, this is not always possible. To receive interaction, display logic can request interaction capabilities that suit its content and surface. The loose coupling between a display surface and interaction modality means that it is also possible for a display to receive interaction from areas which are not spatially connected to it. Using this method, interaction modalities can be used simultaneously. The toolkit has stock support for several forms of interaction, including: single touch, multi-touch, basic trigger switches, foot detection, primitive gestures and point-polygon intersections. It is possible to specify optional tuning parameters such as a height offset and processing limits.

3.3.3 Multi-Touch

The multi-touch technique is the most complex interaction method offered natively by the toolkit. Our multi-touch algorithm is implemented in JavaScript and works by injecting touch events (which meet the W3C specification [23]) into the web browser's event model. Displays that want to use this feature can reference the multi-touch script in the head of their HTML file.

The multi-touch algorithm operates as follows: The display content (running in a web browser) first requests point cloud data from the toolkit regarding the region directly above the intended surface. Then, on receipt of each new frame of data from the depth camera, the toolkit culls points outside the requested region and dispatches those points which were not culled, back into the browser as arguments passed into a JavaScript function. The function which receives this data then uses a kd-tree [3] enhanced DB Scan algorithm [5] to cluster the points based on neighbour density. The resultant clusters are then checked against those detected in the previous frame and paired based on the minimum Euclidean distance from one another. This data is then used to inject multi-touch events into the DOM.

This process differs from most algorithms which are used to optically detect multi-touch because it operates on point cloud data rather than an optically contiguous image frame. Advantages of this are: greater robustness to different sensor positions and orientations, it is considerably easier to integrate multiple sensors by merging point clouds, and it can operate on comparatively small amounts of data—meaning that the sensor can be placed further away. As with any optical touch detection system, moving the sensor further away from the surface reduces resolution and thus accuracy. Disadvantages of the point cloud approach include that it is harder to extract information derived from the perspective of the sensor (i.e. the curve of a finger) and that it can be more computationally expensive than other optical methods.

4. SYSTEM EVALUATION

To enable rapid deployment in various physical environments, the toolkit needs to support various physical hardware positions. This system evaluation serves to profile the toolkit's multi-touch accuracy when the depth camera is placed at various distances and angles from the interaction surface. It also characterises the software performance by measuring the frame rate under different load conditions. We do not assess projection mapping resolution (the accuracy with which displays are mapped to physical surfaces) as this is dependent on projector resolution and placement.

4.1.1 System Configuration and Profiling Method

All system evaluations were conducted using an Intel Core i5 2500K (3.30Ghz) computer with 4GB of RAM running the Windows 7 (64 Bit) operating system. We used the commercially available Microsoft Kinect and top-mounted short throw projector which had a native resolution of 1280x800 pixels.

To profile the system, we obtained a total of 30 multi-touch accuracy samples at different angles and distances. To reliably vary angle and distance independently, we fixed a depth camera to a pivoting boom attached to a table as shown in Figure 4. The process of obtaining an accuracy sample involved touching a

randomly positioned target 100 times. In each sample the target size was a 1cm^2 circle. This was displayed on a surface 30x21cm in size.

We measure 'accuracy error' as the distance between the on-screen target and the recorded touch position. Following each touch, the on-screen target would disappear and then re-appear 500ms later in a randomised position. Randomising the position rather than using a repeating grid helps to minimise sampling error resulting from sensor artefacts. A drawback of measuring accuracy error this way is that it encounters variance due to user error. However, as the objective of this study is to obtain an empirical profile of the toolkit's accuracy, we consider this information valuable.

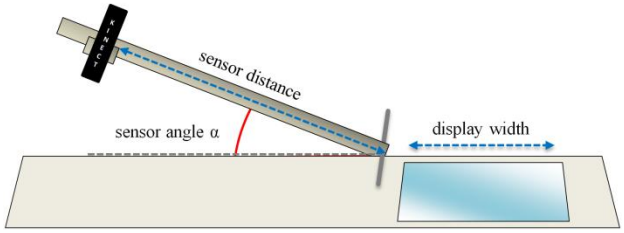


Figure 4: Hardware configuration used for accuracy profiling

4.1.2 Comparison to Capacitive Touch Screen

Figure 5 compares the accuracy error of the toolkit to that of a capacitive touch screen. As one would expect, the capacitive touch surface exhibited less accuracy error (μ=1.0mm, σ=0.7mm) than the depth camera (μ=4.5mm, σ=2.7mm). However, in the context of a typical finger (pictured) we assert that a variance of ≈5mm acceptable for coarse interaction. Cross referencing this with findings from a qualitative study (see Section 5.1); all participants indicated that the accuracy and speed of the multi-touch system was good enough to support their application.

Although this is acceptable for coarse interactions, it is far from perfect. We suggest that developers add a healthy margin of error to the size of touch targets and avoid creating buttons smaller than a finger. The toolkit content API (Section 3.3.2) can access the surface dimensions to help developers automatically convert between pixels and meters.

4.1.3 Distance and Depth Accuracy Profile

Figure 6 visualises the separated effects that sensor distance and angle have on touch accuracy. The trends in the graph show that accuracy is a function of sensor distance and is largely independent from sensor angle. Within the context of finger size and the toolkit's goals, we can conclude that our multi-touch algorithm is able to offer interaction over a range of angles and distances.

Up to approximately 1.3m, the graph shows that our algorithm was able to operate over most angles and distances with performance comparable to the result in Figure 5 (accuracy error of ≈5±2mm). Performance begins to degrade faster as distance increases past 1.4m: the error can be sometimes as much as 2cm (almost double the width of a typical finger). As the variance reported in Figure 7 suggests, this makes it very difficult to use multi-touch interaction for precise operations under such circumstances. We suggest using larger interactive controls if it is not possible to move the sensor.

It is worth noting that while touch accuracy does not degrade with angle, calibrating the system and drawing displays at particularly acute angles (≲ 21 degrees) can be problematic on a video feed. A surface at these angles occupies only a thin slice of video frame. At angles approaching 90 degrees, more of the video frame is occupied by the surface, so calibration and drawing is easier.

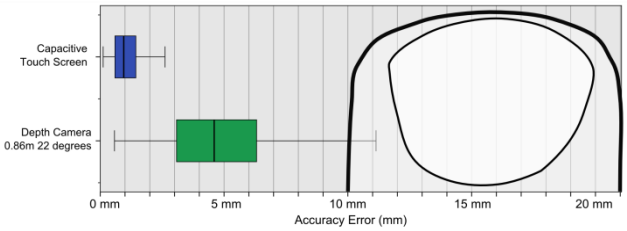


Figure 5: A comparison of capacitive and optical (depth camera) touch accuracy

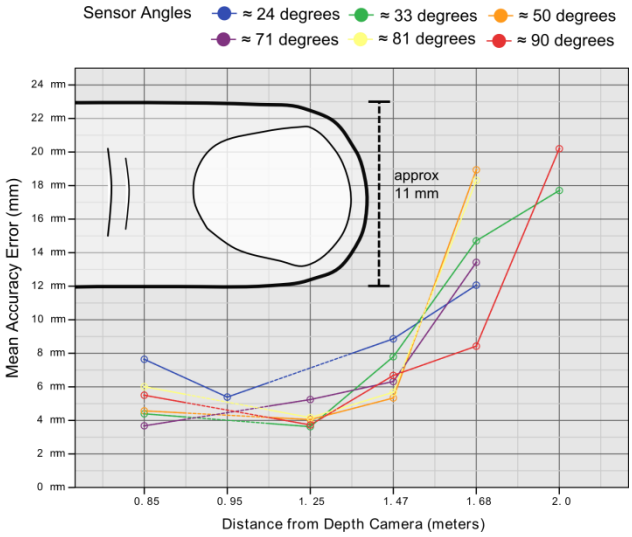


Figure 6: Graph to show the mean touch accuracy error (y-axis) separated by angle (colours) and distance (x-axis). Dashed lines show interpolated measurements

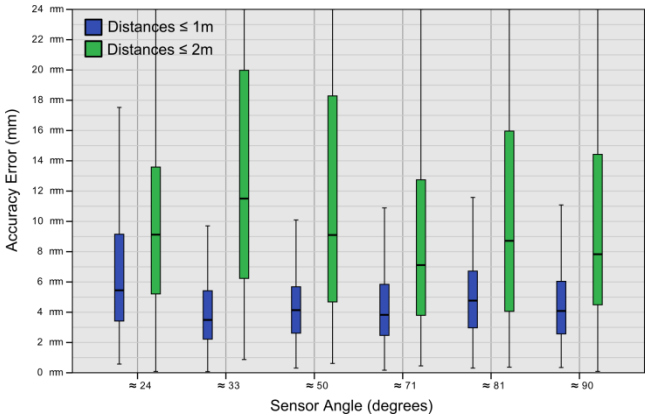


Figure 7: Shows the variance in 'accuracy error' by angle, grouped by exclusive distances ≤1m and ≤2m.

4.1.4 Sensor Accuracy for Touch Detection

Although accuracy can be improved simply by moving the sensor closer to the target surface—a practical and acceptable approach in the context of this toolkit—as we examine why accuracy degrades, it is apparent that these findings have more general implications for the use of depth cameras as touch sensors.

The Microsoft Kinect works by projecting a grid of infra-red dots over a physical space. A camera then recognises patterns in these dots; calculating the depth of each dot based on how the pattern has shifted. These values are then adjusted by the perspective matrix of the sensor lens, yielding a 3D point cloud where each point represents a dot. The further away from the centre of the projection, the sparser the point cloud becomes. Simply, objects closer to the sensor have more detail, as shown in Figure 8.

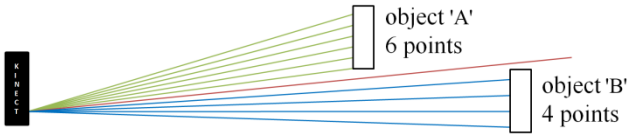


Figure 8: Illustrating why resolution decreases with distance from the sensor. Note 'A' has 6 intersections and 'B' has 4

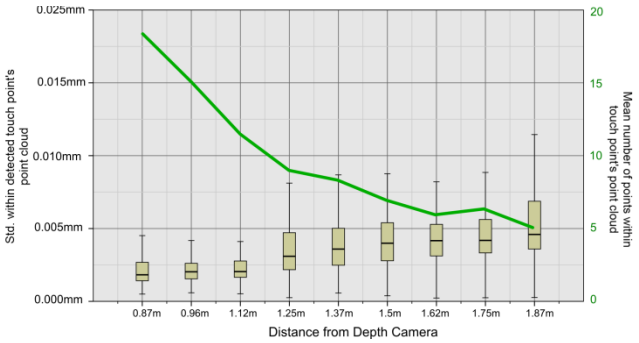


Figure 9: Showing standard deviation within a touch's point cloud increasing over distance (box plots), while the number of points in the same cloud decreases (line)

To examine the effects of decreasing resolution on our multi-touch algorithm, we created a large interactive surface using the toolkit which spanned the distance where accuracy most quickly degraded (as shown in Figure 6). The results in Figure 9 demonstrate how deteriorating point cloud resolution affected the algorithms ability to form a coherent touch point. The left axis (box plots) describes the average standard deviation within each point cloud representing a finger. This variance increases with distance. That is to say, the further away a touch is from the sensor, the more spread out its point cloud becomes. The right axis (line) shows the average number of points used to identify a touch. This decreases as distance from the sensor increases.

The implications for deployments that endeavour to provide interactive displays using sensors mounted at large distances (i.e. over 1.4m) are that unless those sensors can offer sufficient resolution over the interactive areas (approximately 15 points per finger as in Figure 9) they may not be sufficiently accurate. In the context of current hardware limitations, this offers a compelling argument for the use of portable sensors or instrumentation of spaces with pre-defined interactive surfaces. A person configuring the space (or holding a sensor) can then optimise its placement if they know where most interaction will take place. To help developers achieve optimum accuracy it may be useful to offer a sensor placement efficiency measure for each surface.

4.1.5 Performance

To profile the performance of the toolkit we measured the number of frames each surface was able to draw per second once all processing had completed. Once a base line under no load had been established, we varied the stress level of the toolkit in three different ways and examined the response. The three stress types were: the number of touch points, number of rendering surfaces and number of rendering video streams.

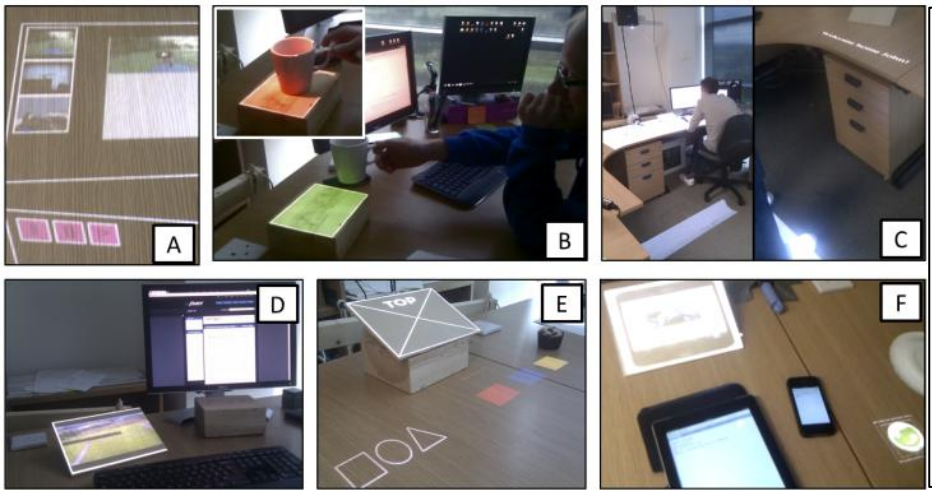
In all cases, the toolkit offered good performance under the measured conditions. It consistently ran at 30fps while rendering video and audio on 18 simultaneous displays (drawing at 800x600px). The most processor intensive condition was detecting touch points. However, it should be noted that this could easily be improved by implementing the expensive multi-touch algorithm in the toolkit (.NET) rather than JavaScript hosted by the content.

5. USABILITY EVALUATION

To assess the usability of the toolkit, two studies were undertaken. The first study involved observing eight participants familiar with web programming while they used the toolkit. They were asked to follow a series of introductory steps, followed by a free-reign session where they were asked to develop an application of their own. The second study took place over a longer period of time and was designed to assess the toolkit in terms of how well it met the projects requirements and performed over a longer period of time.

5.1 Short Term

The short term study is divided into two parts. In the first part, participants were asked to follow a simple tutorial text (10 sentences, one per step followed by the creation of two example



List of participant projects developed using the toolkit:

P1: Video Viewer (A)
P2: Coffee Mug Alarm (B)
P3: Foot Combination Lock (C)
P4: Security Camera (D)
P5: Shape Mixer (E)
P6: Shape Mixer (no pic)
P7: Multi-touch Web Browser (no pic)
P8: Uploadable Picture Frame (F)

Figure 10: Selected display systems created by participants

display applications) that guided them through using the main features of the toolkit. In the second part, participants were given free rein to design and program their own (relatively complex) display deployment. Participants were allowed to implement their own creative design or choose one from a list of three suggestions intended to cover a range of the toolkits capabilities. The first was to implement a mechanism for transferring content between a display and a mobile device, the second was an interactive video viewer with separate control panel, and the third was a 'shape mixer' whereby a user would select a colour on one display and a shape on another so that the combined shape and colour was shown on another display.

Each study session lasted on average two hours. Of the eight participants who undertook the study, five were PhD students with programming skills and the remaining three were programmers working in industry. All but one indicated they had experience with web development and none indicated any experience with projection mapping or developing for the Microsoft Kinect. As participants progressed through the study, they were asked to fill out a questionnaire featuring Likert scales and open ended answers.

5.1.1 Part 1 - Tutorial

All participants were able to follow the tutorial to a successful conclusion. The majority agreed that the steps were simple to follow and that the interface was easy to use. Many of the issues participants reported were small, easy to fix usability issues. For example: removing technical language in tooltips and displaying monitor brand names when selecting a projector. However, only five out of eight realised that it was possible to pan and zoom the video stream for more accurate drawing and calibration. One participant suggested that the toolkit offer to help by providing semi-transparent mini animated overlays which demonstrate the process the developer must undertake.

Of the steps which were flagged as being harder than others (for example, manipulating the dimensions of an existing surface), all either agreed or strongly agreed that they would be able to do this again unaided. This pattern was observed throughout the study—a steep initial learning curve which soon diminished once acted out. This highlights a need to provide examples (possibly via a short tutorial video) which visually demonstrate the purpose of each step and its effects.

Most participants noted that they did not expect to be able to 'draw' a surface freehand and would have rather work directly with a rectangle which could be manipulated after an initial placement. All but one participant strongly agreed that they would be able to repeat this process. The participant who disagreed (P7) said that he felt more control was needed over the placement process and that 'eyeballing' the projected surface was not accurate enough. He suggested to providing accelerators for common functions like moving, rotating and scaling each surface, along with direct coordinate control. When asked what they would change, five participants suggested additional visual hints such as highlighting which corner of the surface was being modified, in addition to projecting display bounds as they were drawn.

During the process of re-creating the two sample display applications (the first to make a button which plays a sound on touch, the second to make a display that is able to jump between surfaces), participants felt the majority of the confusion they experienced stemmed from the web development (CSS behaviours etc) rather than the functionality of the toolkit.

All participants agreed that the process of deploying display content onto a surface by drag-dropping onto the relevant part of the video feed was easy to understand. During the development and debugging process, this function was used heavily in an 'edit and deploy' cycle.

One participant modified his display content so that it would only show on a particular surface. If it were deployed to another, the display would automatically locate the intended surface and move to it. When asked why he did this, he said that he "*want[ed] to be able to drag it anywhere and have it appear in the right place automatically*". This 'content homing device' worked until two instances of the same display code were deployed at the same time. This created a loop where one display would displace the other, causing the other to displace the first. While not particularly harmful in a small configuration such as this one, it raises an interesting question: If many individuals are responsible for their own personal display content, is a mediating system required to detect such conditions or provide permission to displace other content?

5.1.2 Part 2 - Involved Development

In the open ended development task, half of the participants opted to design their own display. Their project names and a selection of photographs featuring the systems they created are provided in Figure 10. These designs demonstrate a range of creative and interesting applications—all of which were successfully realised. In addition to traditional multi-touch interaction, two participants implemented non-traditional interaction techniques. P2 developed a coffee mug detector and P3 implemented a foot based combination lock which would enable a desk display when the user stood in the correct location. To implement the latter, P3 also adjusted the projector lens and re-mounted the depth camera to get a better view of the floor and table.

5.1.2.1 Overall Perceptions

Perhaps the most overwhelming reaction from the participants was that it was both a "*fun and different*" experience. This information was obtained after the study had concluded—often following more than two hours of involvement. As a result, this feedback is particularly encouraging in the context of adoption by hobbyist community; it is important that the process of developing stays both rewarding and enjoyable. To qualify this, we are not suggesting that the toolkit's design was the source of this enjoyment, but rather the creativity and novel concepts it exposed. Indeed, all participants indicated that they were happy with what they had built.

In terms of transferability, all but one (P7) agreed that it would be easy to teach others how to use, and everyone agreed that should they want to create an interactive projected display in the future, they would consider using this toolkit. All participants indicated they could imagine using the toolkit to create systems other than the one they had developed. It was mentioned in one participant's general remarks that they thought the cost of the projection hardware required was still too much of a barrier for wide scale adoption within the hobbyist community.

5.1.2.2 Conceptual Understanding

None of the participants experienced issues understanding the conceptual differences between the 'display' and 'surface' abstractions. The idea of naming surfaces was understood by all the participants. Interestingly, in applications that were deployed over a larger physical scale (or made use of distinctive physical objects such as a large block of wood) surfaces were named to reflect physical characteristics (e.g. 'floor' or 'mug stand').

However, where participants made systems that were less dependent on physical situation (i.e. P4 and P5's shape-mixer and P1's video browser) surfaces were usually given names which reflected their function (e.g. 'video controls').

Although either model of surface addressing is appropriate for relatively small-scale systems, in larger deployments such as those on a room or a building scale, naming surfaces based on physical characteristics requires content to be developed with an appreciation for the naming conventions present. A better solution may lie in disregarding named surfaces altogether and instead, referencing a 3D model of the environment that can be addressed as a large virtual canvas. While this makes the process of autonomously configuring displays easier, it does so at the cost of developer control. In the future, it may be prudent to create a description language which combines the desirable features of both so that content might easily locate suitable display surfaces.

The ability to change interaction modality from multi-touch to other detector types was used by two of the participants (P2 and P3). However, most participants expected multi-touch to be enabled by default. They saw it as a hindrance that they had to add it themselves. In future toolkits, it may simplify the experience if common interaction methods can be toggled on or off from the toolkit interface.

5.1.2.3 Developing for the Physical World

A particularly interesting observation made throughout the development process was that all but two participants heavily tested and debugged their systems in the physical world (deployed on a surface), rather than in an on-screen browser. In that sense, the ability to interactively place content onto target areas made the toolkit's interface a programming environment that emphasised the relationship with the physical environment. Without this relationship, participants would have been forced to imagine how their application would behave. Although far from a complete solution, we believe our toolkit and the abstractions offered may be a first step towards what Abowd referred to as a programming environment for programming environments [1].

When programming for physical spaces using the toolkit, much time was spent debugging. Unlike testing on a screen or in a simulator, participants would have to stand up, move around, or interact with physical objects. This presented an interesting set of challenges. For instance, how do you debug and monitor applications when you are not at your computer? While one solution would be have a remotely accessible debug log that could be carried on a mobile device, P3 suggested that it would be nice to be able to clone a display, so that he could have one next to his computer and another live in the environment.

Another challenge in programming physical spaces is the notion of trigger events and distributed display applications. For instance, pressing a button on one display may cause a change on another. The toolkit provides basic support for this kind of behaviour (i.e. function calls between content hosted on different surfaces), but in order to use them one must develop a display that reacts to certain conditions and informs others of its change in state. This encourages decentralised architectures that are formed from multiple pieces of interacting content. Deployments like this may become difficult to manage as they scale.

From a usability perspective, an intuitive solution may already exist in the form of 3D level editors used in computer games. This would allow designers to 'wire-up' common triggers located in physical spaces (such as 'person in radius of display surface') to content functions. This would be useful when creating exotic, multi-modal display configurations like that of P3. In this situation, a 3D view would have made positing and configuring the surfaces easier. However, this would make the user interface more complex and computationally expensive. For the purposes of this toolkit—where most displays will be created at different angles along a larger dominant plane—the video feed was an intuitive and simple solution.

5.1.2.4 Suitability for Rapid Prototyping

In terms of the toolkit's ability to facilitate rapid prototyping, integrating open source libraries and samples was relatively smooth. For example, P4 used an online webcam streaming service to construct a peripheral desk security camera. Participants also liked that the JavaScript was able to both easily manipulate content and interoperate between displays. For instance: making another display fade out before completely disappearing. In the same spirit, one participant remarked on the possibility of integrating external devices (such as a large physical push button) to the toolkit via JavaScript web sockets. While the toolkit supports this, the process of doing so requires more technical skills.

5.2 Long Term

To complement the shorter term evaluation, the long term evaluation focuses on requirements satisfaction, developer usability and general feedback on the experience of using the toolkit over the period of a few months.

The toolkit was given to an application driven research project investigating how novel and engaging displays can help to improve feedback quality within public spaces. They used the toolkit to construct a large (~2m^2) interactive floor display (Figure 11) which is capable of detecting and reacting to people as they walk over certain areas. People walking over certain areas (i.e visual representations of buildings on a 2D map), trigger the display of related content on another co-located display.

Figure 11: A prototype of the deployment used in the long term project

The investigator responsible for the project said that the toolkit allowed them to create what they had intended. He highlighted that the easy setup and deployment process let them progress with their interests rather than focusing on the technical aspects of the supporting system. Furthermore, the ability to tweak the surface locations and swap out content interactively was useful during the development process as it enabled them to quickly experiment with alternative deployment configurations.

In terms of integration, the use of web standards allowed them to adapt web code in their existing project eco-system: "*[It was] a*

simple matter of adding some multi-touch capable JavaScript code to the visualisations which previously just used mouse interaction." "*We found it no more difficult than developing for desktop or mobile browser interaction.*" The investigator highlighted a need for careful consideration of the colours (and other design choices) as the material and texture of the floor greatly affected visibility.

Overall, speed and responsiveness appeared to be a primary concern. It transpired that the deployment PC they initially selected (an Intel Celeron) was not powerful enough to support the toolkit. A feature they desired which was not present was the ability to manage the deployment remotely, perhaps via a web interface. This also suggests a certain amount of automation may be necessary in situations where a maintainer is not or cannot always be present. A notable advantage of projected displays in this scenario is that they do not appear 'broken' when switched off.

6. DISCUSSION

The toolkit achieves its goal of being a simple to use method of rapidly creating interactive projected displays. This is evidenced by the developers who used the system reporting that they were able to create a diverse range of applications almost exactly as they had envisioned.

Our toolkit takes a different approach to existing frameworks [2] [4] [10] [15] as it makes efficient use of projection resources by supporting multiple visual surfaces and sensing areas with a single projector and depth camera. The deployment configuration process is also greatly improved by integrating projection mapping into the surface creation process. However, a limitation of this is that unnecessary resources are expended when a piece of display content does not require a visual an interface (i.e. a light switch).

Given that the toolkit is open source[2], there is potential for community development. From our observations of both short and long term use, we recommend that it be extended to support multiple projectors and depth cameras to enable the creation of larger displays. In light of participant feedback, providing video tutorials, sample code, and the provision of high quality documentation will be key to its success. Fostering a community is also valuable as encourages people to share display code, new interaction algorithms, and an emerging prosody of interaction patterns.

The toolkit is suitable for the target demographic as it makes use of familiar web standards and is able to provide accurate enough multi-touch interaction for most rapidly-prototyped concepts. Most participants agreed that the use of web standards was not limiting in terms of what could be created. To help address lower accuracy at larger sensor distances, we recommend that developers increase the size of interactive controls to suit the accuracy profile provided in the system evaluation section. It may be that 'frameless' [18] projection (i.e. projections without borders) onto uninstrumented surfaces is conducive to the design of content that requires less precise touch interaction. The lower accuracy may also encourage developers to experiment with such designs, taking advantage of physical shapes and embodied context.

[2] http://code.google.com/p/ubidisplays/

6.1 Supporting Developers

6.1.1 User Interfaces

We asked all who used the toolkit what the three most important aspects they would change or improve were. Discounting minor user interface tweaks, the most common responses were: more documentation, a detachable debug log, and an HTML element inspector/debugger.

Following the suggestions for more display layout functionality (and P3's experience deploying surfaces over two planes, the floor and table) we would also recommend an advanced display management mode featuring a rotatable 3D representation of the deployment environment. This would give developers more precise control over how they positioned surfaces, potentially making it easier to align data from multiple depth cameras and projectors.

6.1.2 Mental Models

To help make the process of development and deployment a welcome prospect, it was important the abstractions ('surface' and 'display') were suitable, simple, and easy to work with. Exposing these as programmable constructs was particularly valuable as developers could use them to tightly integrate content to physical context and conditions. Furthermore, present within these abstractions is a loose coupling between programmable displays and the physical deployment surfaces that hosted them. This opens up many possibilities for content automation methods and other creative applications—as demonstrated by the participants.

6.1.3 Located Code

The toolkit's abstractions also had the effect of perceptually giving display content a physical location. During short term user tests, it made conceptual sense for participants to literally place the display logic on the relevant part of the environment. For example, P3 placed display content on the floor which had no visible interface and functioned only as a trigger. We argue that a mental model which associates logic with a physical space is advantageous. The design of this toolkit helped to promote this way of thinking.

The idea of 'located code' could be extended to allow displays and 'trigger logic' to follow a user, perhaps by being hosted on their mobile device. As the number of available display surfaces increases, the need for developer tools that support the programming of physical spaces becomes clear. The computer game 'level editor' concept is particularly compelling. Furthermore, if display content is to interoperate—combining several depth cameras for better accuracy or projectors for a greater display size—then a distributed approach to the system design is needed.

6.2 Scaling Ubiquitous Displays

Broadly speaking, there are two apparent approaches in the design of a large distributed display ubiquitous system. The first is based on the assembly of many ad-hoc individual deployments, each suited to a specific purpose. The second is a centralised, general purpose approach which involves constructing a global coordinate system for the entire target environment.

In the ad-hoc approach, each deployment can be managed independently and interconnection can be achieved without a need for centralisation. It is also not underpinned by a global coordinate system (although one could be inferred by a user manually positioning each installation relative to a blueprint of a building or a 3D model, if available). Challenges for this approach include designing a surface addressing scheme with scalable abstractions

(i.e. surface naming conventions), and creating an efficient distributed system design. We envisage a design based on the end-to-end principle [22], where the complex processing of sensor data or visual output is placed directly in the environment, located close to the sensors performing the processing. Other challenges include managing permissions and security between deployments.

An advantage of the centralised approach is that it is easy to automate: content can be located on any surface without being pre-defined. A challenge for this approach is to provide enough accuracy for common interaction techniques (such as multi-touch) while maintaining their generality. Approaches using handheld projectors, as demonstrated by Molyneaux et al. [13], offer a nice solution but at the cost of requiring each user to carry a special device. Another challenge is to understand if the overhead involved in constructing such a model is worth the effort expended.

Do people have ideas for applications which would make use of content on 'any' surface? Are these ideas valuable? It is our hope that this toolkit can help uncover answers these questions by enabling more application driven research using technology from this domain.

7. CONCLUSION

We believe the software toolkit presented in this paper is a valuable stepping stone that will enable non-experts to engage with the idea of ubiquitous displays. By offering a simple and fast deployment process, we have demonstrated how this can be used to quickly create interactive and multimedia-rich projected displays. In doing so, we shed light on the accuracy issues faced when using depth cameras as a means of multi-touch detection and offer insights into the development experience when writing program logic for the physical environment.

Following both short and long term studies we are struck by the range of novel systems that people created using the toolkit and that all who used it could imagine it being useful in other situations.

8. ACKNOWLEDGEMENTS

This project is supported by the RCUK-funded Centre for Doctoral Training, HighWire (highwire.lancs.ac.uk). Grant reference EP/G037582/1.

9. REFERENCES

1. Abowd, G. What next, Ubicomp? Celebrating an intellectual disappearing act. In *UbiComp'2012* (2012).
2. Bellucci, A., Malizia, A., and Aedo, I. TESIS: Turn Every Surface into an Interactive Surface. In *ITS '11 (Video)* (2011).
3. Bentley, J.L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18, 9 (1975), 509-517.
4. CYCLING74. *MAX/MSP*. 2012. cycling74.com/whatismax/.
5. Ester, M., Kriegel, H., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD-96* (1996), 226–231.
6. Harrison, C., Benko, H., and Wilson, A. OmniTouch: wearable multitouch interaction everywhere. In *UIST'2011* (2011), 441-450.
7. Hodges, S., Villar, N., Scott, J., and Schmidt, A. A New Era for Ubicomp Development. In *IEEE Pervasive Computing* (2012), 5-9.
8. Kaltenbrunner, M., Bovermann, T., Bencina, R., Costanza, E. TUIO - A Protocol for Table-Top Tangible User Interfaces. In *GW'05* (2005).
9. Kjeldsen, R., Levas, A., and Pinhanez, C. Dynamically Reconfigurable Vision-Based User Interfaces. In *ICVS'03* (2003).
10. Klompmaker, F., Karsten, N., and Fast, A. dSensingNI: a framework for advanced tangible interaction using a depth camera. In *TEI '12* (2012).
11. Lee, J. Hacking the Nintendo Wii Remote. In *IEEE Pervasive Computing* (2008), 39-45.
12. Mistry, P., Maes, P., and Chang, L. WUW - wear Ur world: a wearable gestural interface. In *CHI'09* (2009), 4111-4116.
13. Molyneaux, D., Izadi, S., Kim, D. et. al. Interactive Environment-Aware Handheld Projectors. In *Pervasive'2012* (2012).
14. Molyneaux, D. and Kortuem, G. Ubiquitous Displays in Dynamic Environments: Issues and Opportunities. In *Workshop on Ubiquitous Display Environments (Ubicomp'04)* (2004).
15. Noble, J. *Programming Interactivity: A Designer's Guide to Processing, Arduino, and openFrameworks*. O'Reilly Media, 2009.
16. Pinhanez, C. The Everywhere Displays Projector: A Device to Create Ubiquitous Graphical Interfaces. In *Ubicomp '01* (2001), ACM, 315-331.
17. Pinhanez, C. The Everywhere Displays Projector: A Device to Create Ubiquitous Graphical Interfaces. In *Ubicomp '01* (2001), ACM, 315-331.
18. Pinhanez, C. and Podlaseck, M. To Frame or Not to Frame: The Role and Design of Frameless Displays in Ubiquitous Applications. In *Ubicomp'05* (2005).
19. Raskar, R., van Baar, J., Beardsley, P., Willwacher, T., Rao, S., and Forlines, C. iLamps: Geometrically Aware and Self-Configuring Projectors. In *ACM Trans. Graph.* (2003), 809-818.
20. Rekimoto, J and Saitoh, M. Augmented Surfaces: A Spatially Continuous Workspace for Hybrid Computing Environments. In *CHI'99* (1999), 378-385.
21. Rogers, E. *Diffusion of innovations (5th ed.)*. Free Press, New York, 2003.
22. Saltzer, J., Reed, D., and Clark.D. End-to-End Arguments in System Design. In *Second International Conference on Distributed Computing Systems* (1981), 509-512.
23. Schepers, D, Moon, S, and Brubeck, M. *Touch Events Specification*. 2011.
24. Toussaint, G. Solving geometric problems with the rotating calipers. In *MELECON'83* (1983).
25. Underkoffler, J., Ullmer, B., and Ishii, H. Emancipated pixels: real-world graphics in the luminous room. In *SIGGRAPH'99* (1999), 385-392.
26. Willis, K., Poupyrev, I, Hudson, S, and Mahler, M. SideBySide: ad-hoc multi-user interaction with handheld projectors. In *UIST'2011* (2011), 431-440.
27. Wilson, A. Using a Depth Camera as a Touch Sensor. In *ITS'2010* (2010), 69-72.
28. Wilson, A. and Benko, H. Combining Multiple Depth Cameras and Projectors for Interactions On, Above, and Between Surfaces. In *UIST'2010* (2010), 273-282.