

Toolkit Support for Interactive Projected Displays



John Miles Hardy B.Sc, M.Res

HighWire Doctoral Training Centre

Lancaster University

A thesis submitted for the degree of

Doctor of Philosophy

26th September 2014

Dedicated to my parents and friends.

Abstract

Interactive projected displays are an emerging class of computer interface with the potential to transform interactions with surfaces in physical environments. They distinguish themselves from other visual output technologies, for instance LCD screens, by overlaying content onto the physical world. They can appear, disappear, and reconfigure themselves to suit a range of application scenarios, physical settings, and user needs. These properties have attracted significant academic research interest, yet the surrounding technical challenges and lack of application developer tools limit adoption to those with advanced technical skills. These barriers prevent people with different expertise from engaging, iteratively evaluating deployments, and thus building a strong community understanding of the technology in context. We argue that creating and deploying interactive projected displays should take hours, not weeks.

This thesis addresses these difficulties through the construction of a toolkit that effectively facilitates user innovation with interactive projected displays. The toolkit's design is informed by a review of related work and a series of in-depth research probes that study different application scenarios. These findings result in toolkit requirements that are then integrated into a cohesive design and implementation. This implementation is evaluated to determine its strengths, limitations, and effectiveness at facilitating the development of applied interactive projected displays. The toolkit is released to support users in the real-world and its adoption studied. The findings describe a range of real application scenarios, case studies, and increase academic understanding of applied interactive projected display toolkits. By significantly lowering the complexity, time, and skills required to develop and deploy interactive projected displays, a diverse community of *over 2,000 individual users* have applied the toolkit to their own projects. Widespread adoption beyond the computer-science academic community will continue to stimulate an exciting new wave of interactive projected display applications that transfer computing functionality into physical spaces.

Declaration

This thesis has not been submitted in support of an application for another degree at this or any other university. It is the result of my own work and includes nothing that is the outcome of work done in collaboration except where specifically indicated. Excerpts of this thesis have been published in the following conference manuscripts and academic publications.

	J. Hardy, C. Ellis, J. Alexander and N. Davies. 2013. <i>Ubi Displays: A Toolkit for the Rapid Creation of Interactive Projected Displays</i> . Demonstration. In Proceedings of the 2nd The International Symposium on Pervasive Displays (PERDIS '13). ACM, Google HQ, Mountain View, CA, USA
	J. Hardy and J. Alexander. 2012. <i>Toolkit Support for Interactive Projected Displays</i> . In Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia (MUM '12). ACM, Ulm, Germany. <i>Best paper award</i> .
	J. Hardy. <i>Reflections: a year spent with an interactive desk</i> . 2012. In ACM Interactions Magazine . Volume 19 Issue 6, November + December 2012. ACM, New York, NY, USA, 56-61.
	J. Hardy. <i>Experiences: a year in the life of an interactive desk</i> . 2012. In Proceedings of the Designing Interactive Systems Conference (DIS '12). ACM, Newcastle, UK, 679-688.
	J. Hardy, E. Rukzio, and N. Davies. <i>Real world responses to interactive gesture based public displays</i> . 2011. In Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia (MUM '11). ACM, Beijing, China, 33-39.
	J. Hardy, C. Bull, G. Kotonya, and J. Whittle. 2011. <i>Digitally annexing desk space for software development (NIER track)</i> . In Proceedings of the 33rd International Conference on Software Engineering (ICSE '11). ACM, Honolulu, Hawaii, USA, 812-815.

Acknowledgements

This thesis would not have been possible without the support and encouragement of so many people. I would specifically like to thank my supervisors: Jason Alexander, Nigel Davies, and Patrick Stacey for their encouragement and being constant sources of insight. I would also like to thank my family for their inspiration, and my friends at Lancaster, HighWire, Excalibur, and beyond for countless great discussions, new perspectives, and many happy days.

Contents

Abstract	I
Declaration.....	II
Acknowledgements	III
Contents	IV
Chapter 1. Introduction	1
1.1 Overview.....	1
1.2 Motivation and User Innovation	3
1.3 Research Question	5
1.4 Research Methodology.....	6
1.5 Contributions	8
1.6 Structure	9
Chapter 2. Background.....	11
2.1 Introduction.....	11
2.2 Visions of Interactive Projection	14
2.3 Projection Technologies.....	24
2.4 Interaction Technologies.....	34
2.5 Content Development	47
2.6 Existing Toolkits	57
2.7 Chapter Summary.....	70
Chapter 3. Research Probes.....	73
3.1 Overview.....	73
3.2 Methodology	74
3.3 Probe I: Software Engineering Table.....	79
3.4 Probe II: Interactive Office Desk	98

3.5 Chapter Summary	114
Chapter 4. Toolkit Requirements	117
4.1 Overview	117
4.2 Stakeholders.....	118
4.3 General Constraints.....	123
4.4 Requirements Specification.....	124
4.5 Chapter Summary	135
Chapter 5. Toolkit Implementation	137
5.1 Overview.....	137
5.2 Architecture.....	139
5.3 Implementation Challenges	155
5.4 Chapter Summary	167
Chapter 6. Toolkit Evaluation	169
6.1 Overview	169
6.2 Performance Analysis	170
6.3 Applied Deployments.....	179
6.4 User Evaluations	185
6.5 Chapter Summary	196
Chapter 7. Toolkit Adoption	199
7.1 Overview.....	199
7.2 Usage Statistics.....	200
7.3 Case Studies.....	217
7.4 Discussion	229
7.5 Chapter Summary	236
Chapter 8. Conclusions	239

8.1 Thesis Summary	239
8.2 Contributions	240
8.3 Discussion.....	242
8.4 Reflection.....	245
8.5 Future Work	248
8.6 Conclusion.....	249
References	251

Chapter 1. Introduction

1.1 Overview

Elements of *ubiquitous computing* research explore technologies that transform physical spaces into appropriately designed computer interfaces [1] [2] [3]. *Interactive projected displays* are an emerging class of computer interface that support this vision by overlaying interactive digital content onto surrounding physical objects and surfaces [4]. Physical spaces that treat our actions as input and use projection to provide output unlock a range of new design opportunities and expand the range of possible user interactions [5] [4] [6] [7] [8].

Projecting interactive digital content onto the spaces we inhabit is a useful and compelling idea that has simulated a diverse research history [4] [9] [5] [10] [2] [3] [11]. Unlike traditional displays that are bound to specific hardware (i.e. fixed-sized LCD screens), interactive projected displays can appear, disappear, and reconfigure themselves to suit the characteristics of a physical environment, user, or application. They can change visual appearance [12], location, physical size and shape [13], or adapt the style of interaction based on design factors or different user needs [8].

Despite a great deal of potential, such displays remain difficult to create and deploy. Those who wish to explore the concept further through the design of applications still face many technical and practical challenges. Specialist equipment, advanced programming skills, and time-consuming development processes all discourage adoption of the technology. This is problematic for exploratory or application driven projects (e.g. hobbyist smart homes, art installations, museum exhibits, and student projects) as the expected results cannot always justify the necessary technology and time investment. *To address this challenge, this thesis explores how toolkits can effectively facilitate user innovation with interactive projected displays.*

1.1 Overview

To effectively facilitate user innovation [14] [15] (the process of enabling users to apply and adapt technology to their own applications) this thesis studies two applied interactive displays through in-depth research probes. The findings inform the design of a toolkit that supports rapid prototyping of applied interactive projected displays. The toolkit is then validated and subjected to in-lab studies preceding a public release. To evaluate in-the-wild effectiveness, statistics of the toolkit's adoption and usage are reported along with a selection of case studies. The toolkit and resultant findings contribute to a greater academic understanding of applied interactive projected displays. Adoption of the toolkit practically engages and empowers the user community with tools and new technological choices (Figure 1).

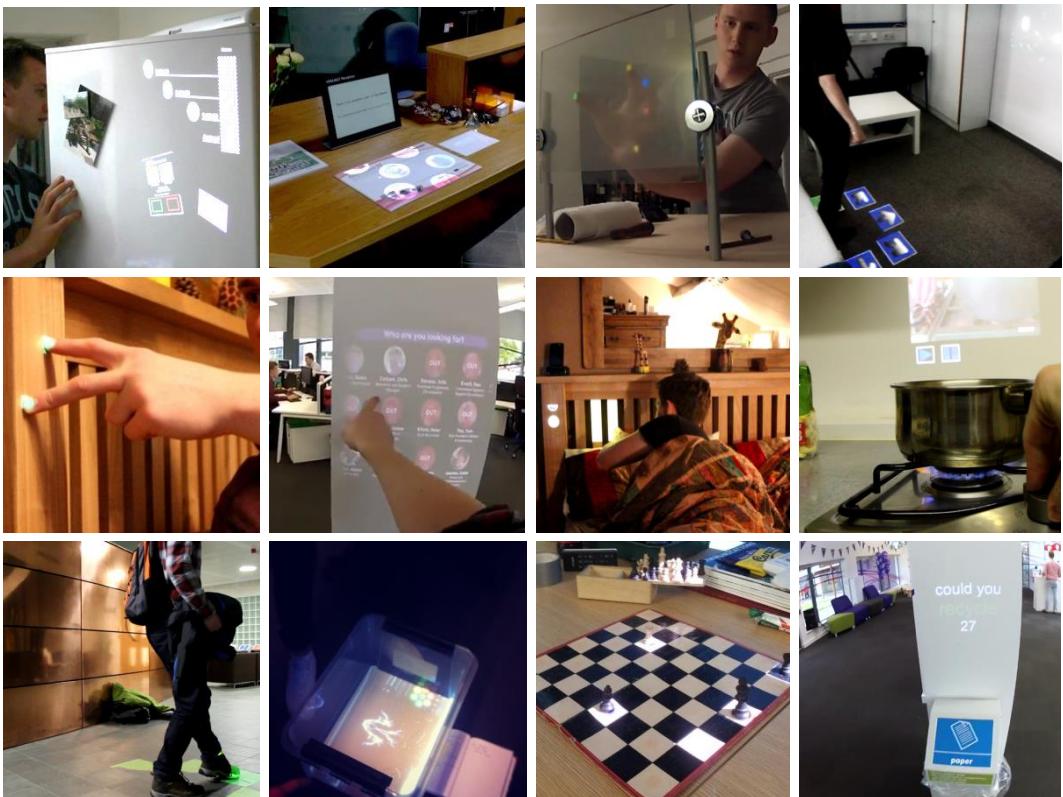


Figure 1: Montage of interactive projected displays created by the toolkit presented in this thesis. Right-to-left: multi-touch fridge, reception desk, two-sided transparent touch screen, dance-floor, multi-touch bed post, building personnel presence pillar, bed controls, cooking video aid, interactive floor, interactive milk project, chessboard, and disposal activated recycling display. Photographs selected to demonstrate a range of interaction modalities and application scenarios. All photographs used with permission.

1.2 Motivation and User Innovation

Interactive projected displays enable a range of new application scenarios to support activities in the physical world [7]. However, the exploration of these is restricted by practical challenges that make it difficult to build pervasive computing applications and experiences [16] [17]. Physical spaces are dynamic, unstructured, and highly volatile in comparison to the relatively constrained desktop setting [18]. Interactive projected display applications that handle these extra conditions require higher levels of technical expertise and often involve non-standard and sophisticated sensor technologies. Developers must overcome significant implementation challenges whilst coping with restricted hardware placements and varying technical constraints. Less technical users have no content-focused developer support (i.e. debugging or common libraries) and must learn domain specific terminology and abstractions (i.e. projection mapping). Furthermore, reaching a standard acceptable for user evaluation takes time, which in turn makes it expensive to iterate on application designs.

All of these factors limit adoption in application driven research and projects undertaken in non-computer science domains that lack the necessary technical skills and experience [16]. A major factor behind the success of the GUI and desktop computing models was that developers were able to draw on a range of tools and software libraries to help realise their ideas [16]. Similarly, the mobile computing field has recently become accessible to a wider range of developers through better application and distribution support. To afford interactive projected displays with the same benefits, researchers need to first simplify the development and rapid prototyping of applications [17] [16] [19].

Toolkits are an extremely effective mechanism for simplifying application development and enabling users to achieve their goals themselves. Tools that remove complexity make it possible for a wider range of people to use them. This, in-turn, helps facilitate multi-disciplinary work within the field of ubiquitous computing [20] [16] [17] and allows the community to learn from the practical challenges of deploying new technologies [18] [16].

1.2 Motivation and User Innovation

Over the past decade numerous tools and platforms have emerged which support various aspects of ubiquitous computing development [21] [22] [23]. While these areas of the field are able to benefit from more open design processes, interactive projected displays have yet to receive such support. The creation of even simple applications still requires specialist equipment [24] [25], relatively controlled circumstances [26], and advanced programming skills [27] [8] [28]. This thesis strives to address these issues by making the process of creating and deploying functional and aesthetically pleasing interactive displays take hours not days.

Achieving this through the creation of a toolkit necessitates an appreciation of the target toolkit user groups and how their desired outputs fit into a broader picture of technology adoption. Subsequently, this work focuses on supporting the innovator and early adopter groups identified in Rogers' diffusion of innovations¹ theory [29] (Figure 2). These groups are willing to experiment with new ideas and provide considerable and candid feedback on new technologies. They are a source of user innovation involving hobbyists, professionals, and academics.

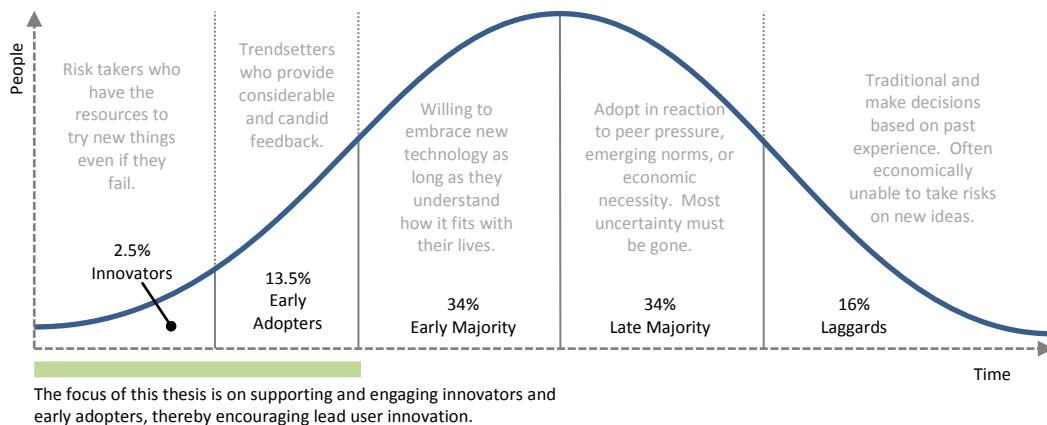


Figure 2: The diffusion of innovations according to Rogers [29]. The focus of this thesis is on supporting the innovators and early adopters.

¹ Diffusion of innovations theory seeks to explain the rate at which new ideas and technology spread through cultures [29].

1.3 Research Question

User innovation stems from the observation that many technologies are developed and refined at the site of use, rather than exclusively by providers [15] [30]. According to Tuomi [31], in the age of the internet, key applications are often unintended and invented by user communities that reinterpret and reinvent the meaning of emerging technological opportunities. Toolkits are an important part of this process as they empower users directly. According to von Hippel [32], toolkits in user innovation “allow manufacturers [or in this case, researchers] to actually abandon their attempts to understand user needs in detail in favor [sic] of transferring need-related aspects of product and service development to users along with an appropriate toolkit”. In a purely academic setting, toolkits play an important role in application driven research. Abowd [16] characterises application driven research as: “[the] 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 the technology community is rarely exposed to the findings of these works.

1.3 Research Question

The central question asked by this thesis is: *how can a toolkit effectively facilitate user innovation with interactive projected displays?* To address the research question the thesis is divided into three research objectives:

1. **Exploration** of interactive projected displays in application driven research in order to identify and converge on an appropriate scope and feature set.
2. **Development** of a toolkit which simplifies and expedites the process of creating interactive projected displays.
3. **Evaluation** of the toolkit in terms of technical viability, suitability for adoption, valuable features, and analysis of in-the-wild adoption.

These are expanded through the contributions outlined in the following sections.

1.4 Research Methodology

Due to its constructive nature, this thesis adopts a design process methodology to shape its approach and coverage of the research objectives (Figure 3). Although there is no best-practice design process [33] [34] it is common to address exploratory goals first (*Objective 1, Exploration*). This maximises the amount of information that can be fed into the toolkit creation (*Objective 2, Development*) and subsequent evaluation (*Objective 3, Evaluation*).

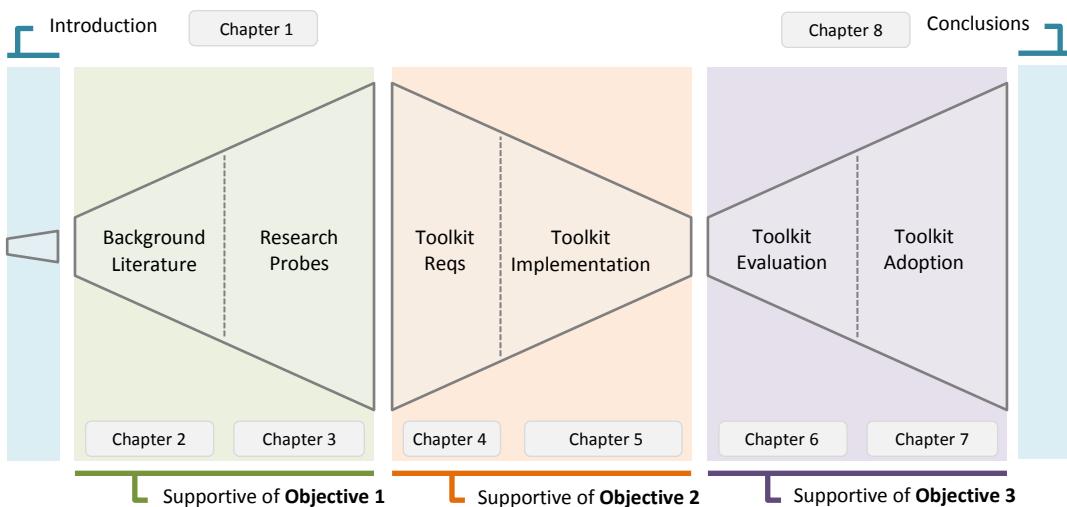


Figure 3: Overview of the design process methodology used in this thesis, showing the divergent and convergent phases (y-axis). Each stage is mapped to thesis objectives and chapters.

Objective 1 is exploratory. Its role is to inform the toolkit design through a series of in-depth research probes which study a range of interactive projected displays characteristics in different application scenarios. These probes are conducted iteratively and consist of a series of generative and evaluative stages that converge on a preferred requirements scope for the toolkit. The methods used in the probes are guided by the needs of the application scenario. In summary, analysis of the first probe uses comparative statistical methods, video coding analysis, expert analysis, and structured interviews. The second probe adopts a longitudinal reflective study,

1.4 Research Methodology

similar to an auto-ethnography². The methods chosen are appropriate for an exploratory goal because they provide a broad view through multiple theoretical lenses. More detail behind the rationale for each of these methods is given in context (Sections 3.3.2 and 3.4.2).

Objective 2 involves identifying requirements for a toolkit that facilitates user innovation (Chapter 4) and describing their implementation into a single cohesive toolkit design (Chapter 5). The requirements are based on the findings of *Objective 1*—drawing on a mixture of the literature in Chapter 2 and the findings of the research probes in Chapter 3.

Objective 3 evaluates the toolkit in terms of its *effectiveness* at *facilitating* user innovation. The evaluation is conducted in two stages. The first stage (Chapter 6) validates that the toolkit satisfies the requirements and is suitable for adoption. This involves profiling the implementation and conducting a controlled experiment with a pilot group of toolkit users; making use of applied statistical analysis, structured questioning, and freeform interviews. The second stage (Chapter 7) analyses the public adoption of the toolkit over the period of one year. This longitudinal approach involves statistical analysis of the toolkit usage data along with qualitative analysis and case studies of external users.

The main limitation and risk of this approach is that its evaluation is contingent on toolkit adoption which is difficult to guarantee *a priori*. However, the use of in-depth application driven research probes, longitudinal analysis, and an evaluation which considers adoption suitability as well as performance profiles help to reduce the risk of an unsuitable toolkit design. The conclusion reflects on the hypothesis that toolkits are an effective method of facilitating user innovation; considering the overall success of the method and its execution as a factor.

² Auto-ethnography is a form of self-reflection that accounts a researcher's personal experiences. Its use in this thesis could be seen as practice-based research through design.

1.5 Contributions

This thesis makes technical, conceptual, and applied contributions to the domain of interactive projected displays. Major contributions are categorised around the three research objectives:

Exploration: *A review of existing interactive projected displays literature and the identification of a set of common characteristics of projected display applications. Insights into the practical challenges of creating interactive projected display applications through the development of two probes.*

- C1. A literature search that covers seminal visions, projection and interaction technologies, content development, and existing toolkits.
- C2. Two research probes that explore applied interactive projected displays. These yield insights into the practical challenges of developing applied interactive projected displays and concurrently make research contributions into each probes' application domains. Specifically:
 - a. The concept, design, implementation, and evaluation of an interactive projected display applied to the software engineering domain.
 - b. The implementation and longitudinal investigation of an interactive projected office desk.

Development: *The requirements, design, and implementation of a toolkit which facilitates user innovation with interactive projected displays.*

- C3. A set of toolkit requirements structured around von Hippel's criteria for toolkits that support user innovation [32].
- C4. A software architecture and toolkit implementation that supports these requirements and integrates them into a cohesive design sensitive to the needs of the target user community.
- C5. The introduction of a number of novel display toolkit concepts and associated implementations including: physical responsive design, platform-

1.6 Structure

agnostic interaction modalities, and a point-cloud based multi-touch detection algorithm that enables a wider range of hardware placements.

- C6. Online support and discussion forums for a community of over 2,000 users that have downloaded and used the toolkit.

Evaluation: *Validation of the toolkit and longitudinal analysis of external adoption.*

- C7. A technical assessment of the toolkit implementation and a profile of touch accuracy and performance.
- C8. A user-study based evaluation of the toolkit's suitability for adoption and ability to support diverse application scenarios.
- C9. An analysis of the diversity and volume of user innovation through quantitative longitudinal analysis and qualitative case-studies.

1.6 Structure

Chapter 1: Introduces interactive projected displays, motivates a toolkit based research approach, and outlines the research objectives, method, and contributions.

Chapter 2: Describes a scope and academic background of interactive projected displays. This focuses on prominent visions, implementation technologies, user interactions, content development approaches, and existing toolkits.

Chapter 3: Explores a range of interactive projected display characteristics in application scenarios through two research probes. These inform the requirements scope for the toolkit by identifying important features and development lessons. The probes are: (1) a display designed to improve the collocated software development process, and (2) a display used to examine long term usage of an interactive desk.

Chapter 4: Specifies the toolkit requirements; drawing on the background in Chapter 2 and the probe findings from Chapter 3. These requirements are structured around von Hippel's toolkits for user innovation criteria [32] and rationale for their inclusion.

1.6 Structure

Chapter 5: Consolidates the requirements into a cohesive toolkit design and implementation. It presents a high level architecture and a discussion of implementation challenges. This includes an in-depth description of the point-cloud based multi-touch detection algorithm.

Chapter 6: Evaluates the toolkit implementation in order to determine the extent to which it satisfies the requirements and is suitable for adoption. This is done through a technical profile, sample deployments, and a short and long term user study.

Chapter 7: Presents a longitudinal analysis of toolkit adoption (one year) through usage statistics, application scenarios, and case studies. It discusses the strengths and weaknesses of the toolkit and analysis approach.

Chapter 8: Concludes the thesis through a reflection on the extent to which the thesis goals and contributions have been achieved, and a discussion of future work.

Chapter 2. Background

2.1 Introduction

Using projection to transform physical surfaces into interactive surfaces has been a persistent goal for researchers investigating post-desktop models of interaction [10] [25] [5] [24] [2] [35] [28] [9]. Although many technical challenges are now understood [28] [4] [36] [37] [38], those who wish to create systems and study the concepts further lack tools that decouple implementation technologies from the content creation process, enable a range of user interactions, and embrace the multi-disciplinary nature of content design and deployment. This chapter is structured to reflect that separation (Figure 4).

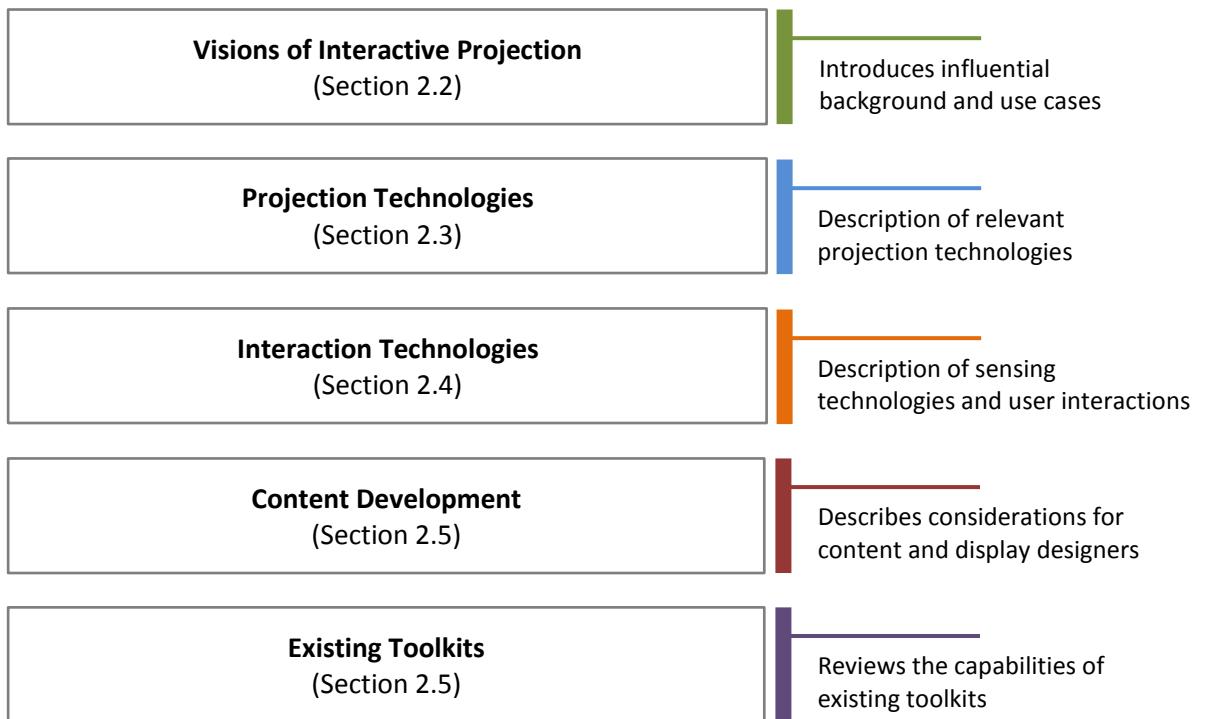


Figure 4: High level overview of the background chapter structure.

2.1 Introduction

2.1.1 Scope

The purpose of this chapter is to review existing related work and describe the opportunities and challenges an interactive projected displays toolkit could address. This places the thesis in a context that frames its contribution. This chapter draws on four main academic research communities: projection-based augmented reality [39], uninstrumented interactive surfaces [9] [8] [40], pervasive computing [1] [2] [18], and situated-displays [41]. The intersection of these interests lies on the far left of the Milgram-Weiser continuum [42] (Figure 5) as they are based within the real environment.

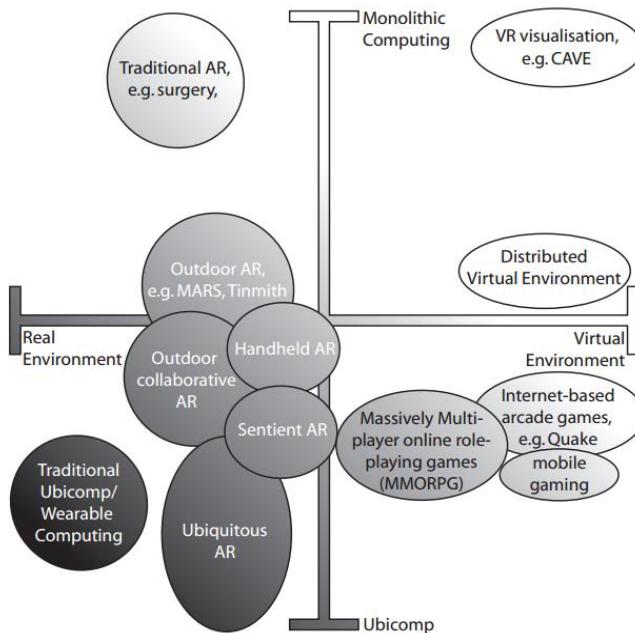


Figure 5: Interactive projection exists on the left-hand side of the Milgram-Weiser continuum [42].

Research has studied interactive projected displays across a range of sizes from small wearable systems to very large building scale media-frontages, as shown in Figure 6. This thesis focuses on supporting *object*, *furniture*, and *room* scale interactive projections that rely on instrumentation of the environment rather than the user. These displays can range in size from millimetres to meters. A selection of significant influential examples and motivating visions are presented in Section 2.2.

2.1 Introduction

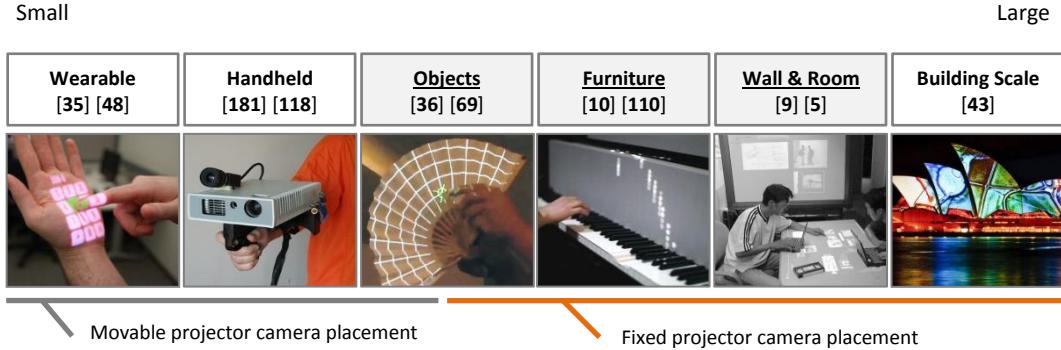


Figure 6: A range of projection scales from smallest (left) to largest (right). Excluding building scale projection, the focus of this thesis is on fixed projector and camera placement (orange). Photographs are taken from the respective literature.

Building-scale interactive projected displays such as Media Façades [43] are not discussed as they face specialist engineering, artistic, and regulatory compliance issues [44] [45] [46]. Similarly, portable projectors [47] [48] [35] (personal ubiquitous displays where individuals carry their own display hardware and sensors) are not included as they face technical issues (e.g. battery life) and design constraints (e.g. a personal ownership model) that are beyond the scope of this thesis.

2.1.2 Structure

The chapter structure is split across five sections outlined in Figure 4 and explained in more detail below:

- **2.2 *Visions of Interactive Projection*:** is intended to create a context by describing how projection has been used to support post-desktop visions of pervasive computing. It covers seven important visions selected based on their influence and lasting contribution.
- **2.3 *Projection Technologies*:** describes methods and techniques for implementing projected displays.

2.2 Visions of Interactive Projection

- **2.4 Interaction Technologies:** describes methods and techniques for creating interactive projected interfaces. It includes a discussion of interaction with different modalities.
- **2.5 Content Development:** discusses design challenges, opportunities, and considerations for content on interactive projected displays.
- **2.6 Existing Toolkits:** reviews relevant toolkits that can be used to create aspects of interactive projected displays.

2.2 Visions of Interactive Projection

The following subsections describe influential academic works that use interactive projected displays. Each is presented chronologically with a summary that highlights distinguishing characteristics of the system and supported application scenarios. Almost all of the systems described are related to the concept of *ubiquitous computing*. This was proposed in a widely cited 1991 paper in which Mark Weiser describes a vision where computation is seamlessly integrated with the fabric of everyday life [1]. He distinguishes the challenges of ubiquitous computing from those in virtual reality by saying [49]: “*Virtual reality is primarily a horse power problem; ubiquitous computing is a very difficult integration of human factors, computer science, engineering, and social sciences.*”

Weiser proposed that three device scales that would be important in a successful ubiquitous computing implementation. From smallest to largest, these are: *tabs*—centimetre scale wearable devices such as badges and watches, *pads*—decimetre scale devices such as smart phones and tablets, and *boards*—meter scale devices such as interactive whiteboards and walls. Weiser reasoned that the power of this concept is not contingent on a strong implementation of only one of these scales, but rather emerges from the interaction of all three.

2.2 Visions of Interactive Projection

2.2.1 The Digital Desk

The '*Digital Desk*' [10] published in 1991 aimed to transfer the GUI 'desktop metaphor' back onto a physical desk. Wellner argued that the ways we physically interact with electronic documents are limited in comparison to our interactions with paper, pencils, rubbers, and other physical tools [10]. That is to say, we lack a computational equivalent of the muscle memory that enables us to defer common tasks to our periphery. There are also tasks (such as copy, paste, summation, etc) that are greatly aided by computerisation. Wellner used the Digital Desk as a way to explore this digital-physical intersection.

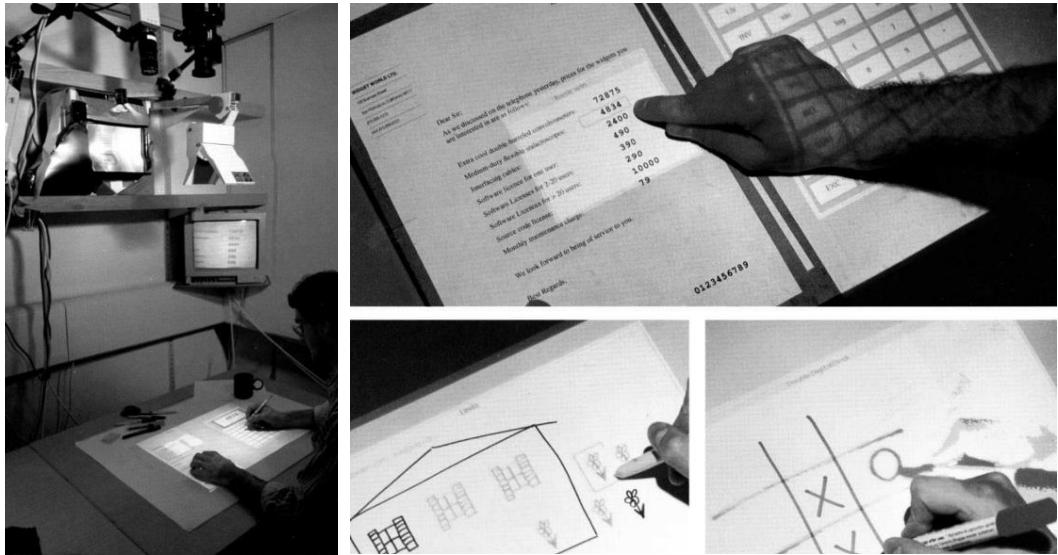


Figure 7: Left: Prototype implementation of the digital desk. **Right:** Calculator, drawing and remote collaboration applications. Photographs taken from: [10].

A prototype Digital Desk was primarily composed of a low-resolution projector and number of video cameras (Figure 7, left). Since its creation this technology has inspired many more researchers to investigate this area. However, interactive desks have yet to challenge the workstation metaphor as a commodity product. With that in mind, relatively few studies have examined the issues surrounding long term use of interactive surfaces [50] [51] [52]. Research tends to focus on short walk-up-and-use scenarios that do not address sustained interaction [53] [54] [55] [56]. While

2.2 Visions of Interactive Projection

this research strategy is often desirable and well suited to focused analysis of specific issues, it struggles to address how these systems are perceived after periods of extended use. The same is true of other forms of interactive projected displays.

2.2.2 Tangible User Interfaces

Ishii and Ullmer's 1997 paper '*Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms*' [2] describes a vision where users are able to physically 'grasp and manipulate' representations of digital information in the centre of their attention, whilst simultaneously being aware of ambient media in their periphery. These tangible user interfaces (TUI) are suggested as an alternative to the dominant GUI model and a step towards ubiquitous user interfaces.

They explore this concept through three prototype systems: the metaDESK, the transBOARD, and the ambientROOM. They consider that being able to transform each physical surface into an interactive surface helps to: "*bridge the gap between cyberspace and the physical environment*". Their intention was to take advantage of natural physical affordances [57] to achieve a heightened legibility and seamlessness of interaction between people and information. They reflect that their vision is not about making computers ubiquitous per se, but rather '*awakening richly-afforded physical objects, instruments, surfaces, and spaces to computational mediation*'.

In terms of interactive projected displays, Ishii and Ullmer conclude that the metaphors of light, shadow, and optics are particularly compelling for interfaces spanning virtual and physical space.

2.2.3 The Office of the Future

The University of North Carolina's '*Office of the Future*' Group³ explored the technical challenges of applying spatially immersive projected displays to an office environment [3]. From 1998 to 2009, this group pioneered many of the projection

³ UNC Office of the Future Group: <http://www.cs.unc.edu/Research/stc/>

2.2 Visions of Interactive Projection

techniques described in Section 2.3. Figure 8 shows the envisioned concept (left) and implemented prototype of this work (right).

In terms of modelling the surrounding space, Raskar et al. [58] used computer vision techniques to dynamically extract per-pixel depth and reflectance information for the visible surfaces in the environment. They conceive that by replacing all the lights in the room with projectors, it is possible to control the appearance of each surface. Their system combines panoramic image displays, tiled display systems, image-based modelling, and immersive environments.



Figure 8: The Office of the Future concept (left) and implementation (right). Pictures taken from [3].

Raskar et al. [11] also developed techniques that enabled them to graphically animate physical objects with projectors using what they termed '*Shader Lamps*'. Figure 9 shows an example of how Shader Lamps can be used to augment a physical Taj Mahal model.



Figure 9: Un-augmented physical object (left) and the same physical object coloured with projected light (right). Figures taken from [11].

2.2 Visions of Interactive Projection

Although object-based 3D spatial augmented reality is of interest, the toolkit contributed by this thesis is intended to support interactive surface based augmented reality: focusing on application support rather than introducing or improving computer-vision and optical methods like Shader Lamps.

2.2.4 Augmented Surfaces

The '*Augmented Surfaces*' concept published in 1998 by Rekimoto et al. [9] blends the focused *Digital Desk* [10] with broader visions such as the *Tangible User Interface* [2] and *Office of the Future* [3]. In their paper, they propose a spatially continuous workspace where people can freely display, move, or attach digital data among their computers, tables, walls, and objects. Their system (Figure 10) consists of a top projected digital table and a front projected digital wall. Users bring notebook computers to a table that are recognised and tracked by a camera placed above. This setup enables the surrounding table and wall to act as an extended and shared interaction space for each notebook computer.



Figure 10: The Augmented Surfaces collaboration environment that consists of a digital table and digital wall. Photograph taken from Rekimoto et al. [9].

2.2 Visions of Interactive Projection

They propose a number of techniques such as hyper-dragging, and pick and beam to seamlessly drag objects across the boundaries of the displays. For instance, when ‘hyper-dragging’, a user is able to move their mouse cursor off the computer screen and onto the table or wall. A coloured line is projected between the cursor and notebook computer to maintain the relationship between the current position and its origin. The system is also able to track physical objects in the space such as a note book, or video tape. When content is hyper-dragged onto these objects it becomes associated with it. Then, by moving the physical object, the user is also able to move the associated digital objects. This is a good example of how the digital and physical worlds can intersect in a meaningful and useful way.

2.2.5 The Luminous Room

In 1999, Underkoffler et al. [5] described a conceptual infrastructure for pervasive environmental output and sensing that they called '*The Luminous Room*'. The concept enabled graphical display and interaction on each surface within a physical interior outfitted with devices that they called '*I/O Bulbs*'. These were lightbulb-style devices capable of sensing interaction and projecting output. Although building a working I/O Bulb remains a research goal to this day [4] [27], treating the concept as a thought experiment enabled them to examine the demands of such an infrastructure. Their paper discussed the feasibility of the required graphical, computation and networking needs. While many of these demands have since been met, some of the core challenges remain (e.g. scaling technical load and content generation).

In terms of interaction, they pose the question [59]: “*If every room surface really is capable of display, what interactions does it make sense to pursue there?*” They explore I/O Bulb supported user interaction through a series of scenarios that they referred to as “*luminous-tangible*” interactions [59]. These scenarios were motivated by real applications in an experimental Luminous Room space. These are enumerated in Figure 11. Contrasting with Ishii and Ullmer’s ‘*phicons*’ [2] (tangible objects that have a symbolic correspondence between a digital meaning and a physical form), objects in the Luminous Room have a direct correspondence with

2.2 Visions of Interactive Projection

physical artefacts and digital meaning. They use the example of optics to demonstrate the corresponding faithfulness of the interaction modalities to the real physical behaviour of light. An obvious critique of this is that it then limits interactions to those for which we have a physical analogue.

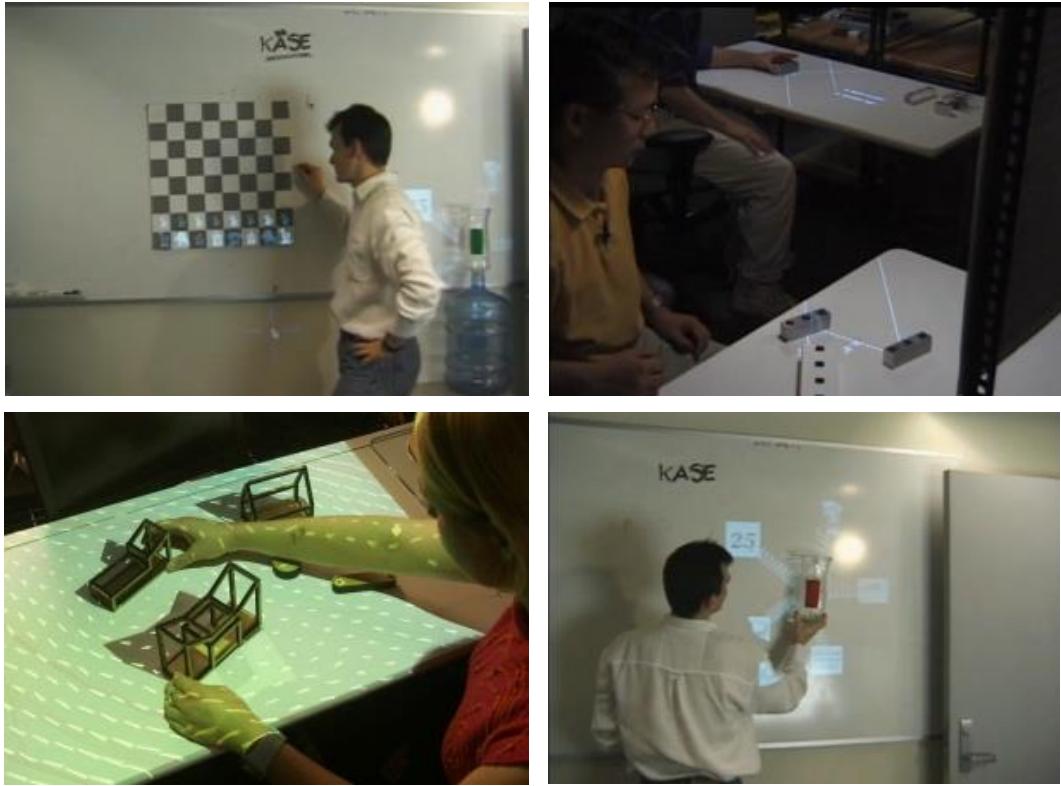


Figure 11: Projected chess board (left, top), distributed lighting optics table (right, top), optical flow simulation table (left, bottom), and projection tracked vase (right, bottom). Photographs from: <http://tangible.media.mit.edu/project/io-bulb-and-luminous-room>

2.2.6 The Everywhere Displays Projector

In 2001, Claudio Pinhanez published '*The Everywhere Displays Projector: A Device to Create Ubiquitous Graphical Interfaces*' [4]. This system created interactive projections on arbitrary planar surfaces within view of a steerable projector-camera assembly. 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 corrected for distortion in the projected

2.2 Visions of Interactive Projection

image for surfaces that did not lie planar to the projection. A schematic and photograph of this system are presented in Figure 12.

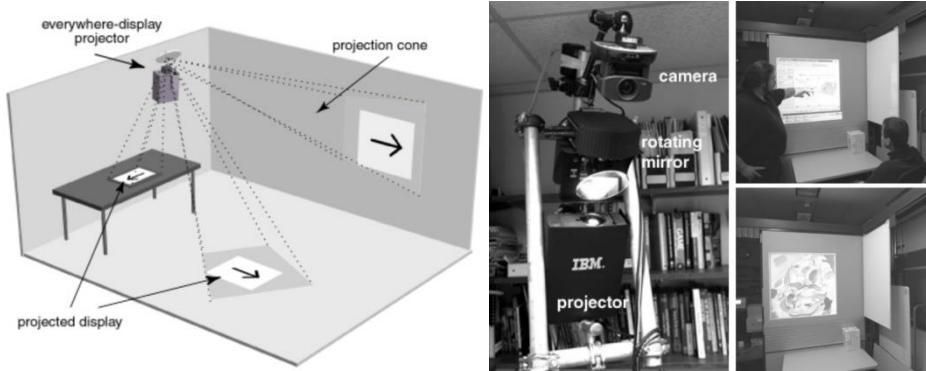


Figure 12: Design drawing of the Everywhere Displays Projector (left) and the actual system in use in an office-like environment (right). Images taken from [4].

Pinhanez et al. were able to demonstrate the technology in many different application scenarios, including retail environments [60] [61], knowledge work [62], and interactive games [61]. They also looked at content production languages [26] and the design affordances of different content styles [12]. They initially expected to have to support the current GUI style paradigms but also saw no reason to confine the interaction to rectangular frames as we are forced to do with monitors.

2.2.7 Summary

The systems above were selected due to their influential early use of interactive projected displays and subsequent impact⁴ on the ubiquitous computing interactive projected display field, both conceptually and technologically. Although the selection is not exhaustive—there is no logical stopping point—each system falls within the size parameters identified in Section 2.1.1 and demonstrated a novel working system that could be applied to one or more scenarios. As such, these works are likely to inform future user experiences with applied interactive projected displays that a toolkit could support, in addition to the toolkit design process itself.

⁴ The mean citation count is 856 and ranges from 182 to 3424 per publication. Citation counts according to scholar.google.com (last updated Sept. 2014).

2.2 Visions of Interactive Projection

Following a review of the systems in this subsection, projected display characteristics are separated into two categories: *interface* characteristics (i.e. form-factor and interaction modalities) and *application* characteristics (i.e. applications and content that the displays support). An exploratory approach is warranted as no such taxonomy currently exists for the space. These categories are relevant to interactive projected display toolkits because the first reflects the features that the toolkit could support, while the second reflects the types of end-goal that the toolkit could be used to help achieve. Figure 13 lists twelve distinguishing characteristics that describe the systems in this subsection in terms of the interface and application categories above. These are posed as questions to assist analysis of the systems. To avoid duplicate characteristics, those that are mutually exclusive (i.e. multi-device and single-device) are given a single question.

Distinguishing Interface Characteristics

- **Multi Device:** Is the user interface spread out over different spaces (i.e. multiple projections)?
- **Frameless:** Does the user interface have no implicit or explicit borders (see Section 2.5.1.2)?
- **Dynamic Geometry:** Does the user interface have the ability to change size and shape?
- **Body Interaction:** Do users interact directly with their body (i.e. touch or gesture)?
- **Device Interaction:** Do users interact with a device (i.e. stylus or mouse)?
- **Tangible Elements:** Does the interface react to the presence or position of physical objects?

Distinguishing Application Characteristics

- **Symbolic AR:** Does the application use graphical symbols as projected overlays. (i.e. icons)?
- **Spatial AR:** Does the application augment reality using spatial metaphors (i.e. lights and shadow)?
- **Public Use:** Is the system applicable in public spaces. (i.e. shopping centres)?
- **Private Use:** Is the technology applicable to (semi-)private spaces. (i.e. domestic and office)?
- **Collaborative:** Is the system designed to enable or assist with collaboration or communication?
- **Task Specific:** Is the operation of the system designed to complete a specific task?
-

Figure 13: Twelve distinguishing characteristic questions for applied interactive projected displays.

Table 1 cross-references these characteristics with the visions discussed in this section. Cases where the vision explored different variations of these characteristics

2.2 Visions of Interactive Projection

through more than one prototype are represented with separate rows. For instance, metaDESK and ambientROOM are both Tangible User Interfaces described by Ishii et al. [7] but have different interface and application features.

Table 1: Cross reference of applied projected display characteristics with the visions discussed in this section. Characteristics shared by more than half of the visions are shaded in the last row.

Vision Characteristics	Interface					Application						
	Multi Device	Frameless Design	Dynamic Geometry	Body Interaction	Device Interaction	Tangible Elements	Symbolic AR	Spatial AR	Public Use	Private Use	Collaborative	Task Specific
TUI metaDESK [2] (3424 cites)						✓	✓	✓	✓		✓	
TUI ambientROOM [2] (3424)	✓	✓				✓	✓		✓			
Office of the Future [3] (789)	✓				✓		✓		✓	✓	✓	
Shader Lamps [11] (333)	✓	✓	✓		✓			✓	✓			
Luminous Room [5] (182)	✓	✓	✓			✓	✓		✓	✓		
Everywhere Displays [4] (347)		✓	✓				✓	✓	✓			
Digital Desk [10] (347)				✓	✓			✓	✓		✓	
Augmented Surfaces [9] (668)	✓	✓			✓	✓	✓		✓	✓		
Frequencies	5	5	3	1	4	4	7	3	3	8	3	3

The most common characteristics shown in Table 1 are highlighted in orange. This indicates the systems are used in a private / semi-private context, such as in a home or office. This may be partly due to the convenience of conducting research in a laboratory. However, the Everywhere Displays projector were deployed in a number of different public scenarios to explore its acceptance and utility [4]. Symbolic augmented reality (i.e. overlaying symbolic graphics, such as icons and text) on physical objects is the next most popular characteristic. Methods for achieving this are discussed in Section 2.3 (technical) Section 2.5 (content design). These are followed by multi-device scenarios and frameless-design discussed in Section 2.4 and Section 2.5.

2.3 Projection Technologies

This section describes a selection of technologies—hardware and software—that can be used to implement the kinds of interactive projected display described in the previous section. In the order of presentation, the subsections are:

- **2.3.1 Projection Hardware:** description of projection hardware technology, relevant trends, and emerging technology.
- **2.3.2 Projection Mapping:** describes projection mapping techniques used to project images onto potentially non-flat surfaces that do not lie planar to the projection lens.
- **2.3.3 Multi-Projector and Multi-Surface Rendering:** describes techniques that can be used to combine multiple projectors into a single image, or use one or more projectors to create consistent images on surfaces with different material properties (such as different colours).

The implementation (and indeed combination) of these techniques present a number of technical challenges that must be overcome by developers in order to create interactive projected displays. Existing toolkits that implement some of these features and abstract complexity from toolkit users are discussed later in Section 2.6.

2.3.1 Projection Hardware

A *digital video projector* is a device that receives a video signal and projects the corresponding image onto a surface (typically a projection screen) using a lens system. Modern projectors (circa 2013) typically use a bright lamp (typically 500-3000 lumens⁵) to project an image (typically between SVGA and HD1080 resolution⁶) onto a flat surface directly in-front of them. Most large modern projectors are able users to manually correct for minor image distortions using

⁵ Lumens Ratings <http://www.projectorpoint.co.uk/Projector-Brightness-Advice.htm>

⁶ Diagram of video resolutions: http://en.wikipedia.org/wiki/Display_resolution

2.3 Projection Technologies

keystone, focal length and other manual settings. Table 2 below presents a glossary of terms relevant to projection technology.

The most common contemporary projection technologies are LCD and DLP⁷, although laser diode projectors⁸ are currently emerging as an alternative. The advantage of DLP projectors over LCD projectors is the use of digital micro-mirror devices [63]. These are microscopically small mirrors laid out in a matrix on a semiconductor chip. The number of mirrors on the chip corresponds to the number of pixels in the projected image. Using electrodes, the micro-mirrors can be rapidly reoriented to reflect light either through the projector lens or onto a heat sink. The advantage of laser diode technology over LCD and DLP is that it generates less heat, improves colour saturation, and the image is never out of focus. Staying in focus is a particularly relevant challenge for interactive projected displays that cover multiple surfaces at different distances and angles to the projection lens.

As with many other vision-based technologies like monitors and televisions, the cost of projectors has consistently fallen whilst technical specifications have improved. Further, new market opportunities such as mobile devices are motivating the development of cheaper, portable, and increasingly energy-efficient projectors [64].

Table 2: Glossary of terms relevant to projection. Adapted from: <http://projectorcentral.com/glossary.cfm>

Term	Description
<i>Brightness</i>	Overall light output from an image. Typically measured in lumens.
<i>Perceived Brightness</i>	The intensity of the light output as perceived by a human rather than a measuring device. The human eye has a logarithmic response to light.
<i>Chromaticity</i>	The colour quality of light that is defined by the wavelength (hue) and saturation. I.e. all the qualities of colour except its brightness.
<i>Contrast Ratio</i>	The ratio between white and black. The larger the contrast ratio the greater the ability to show subtle colour details and tolerance to ambient light.
<i>Focal Length</i>	The distance from the surface of a lens to its focal point.
<i>Frame</i>	A frame is one complete video image.

⁷ Common projection technologies: <http://tinyurl.com/commonprojectiontechs>

⁸ Laser Video Projectors: http://en.wikipedia.org/wiki/Laser_video_projector

2.3 Projection Technologies

<i>Gamma</i>	Relationship between input video voltage and output brightness. Determines how mid-tones appear.
<i>Projector Geometry</i>	Characteristic of a display to accurately show an image without distorting it. Most projectors output a square geometry.
<i>Lens Shift</i>	Helps to reduce keystoneing and provide greater flexibility in the placement of the projector relative to the screen.
<i>Jaggy (Aliasing)</i>	The stair-step or saw tooth effect seen on rasterised lines that are not horizontal or vertical in digital displays. Also known as aliasing.
<i>Keystone</i>	Keystoning occurs when the projector is not perpendicular to the screen, thereby creating an image that is not rectangular.
<i>Latency</i>	The time between a device being requested to do something and the start of the device actually doing it.
<i>Native Resolution</i>	Native Resolution is the number of physical pixels in a display device.
<i>Refresh Rate</i>	The speed at which a display updates its picture given in Hz.
<i>Resolution</i>	A measure of the ability of a display to render detail.
<i>Saturation</i>	Saturation is a measure of colour intensity.
<i>Throw Distance</i>	Throw distance is the measurement from the projector's lens to the screen
<i>Throw Ratio</i>	For any given projector, the width of the image (W) relative to the throw distance (D) is known as the throw ratio D/W.

2.3.2 Projection Mapping

Projection mapping—also known as spatial augmented reality [65]—is a geometric calibration technique that enables digital images to be projected onto irregularly shaped physical objects (Figure 14) and surfaces that do not lie planar to the projection (Figure 15). In essence, a physical object (which can be as simple as a planar surface) is spatially mapped on a virtual 3D model that mimics the real environment to be projected from the perspective of the camera.

The process relies on a virtual model of the physical area or object that will receive the projection. This virtual model expresses both the physical shape of the object and the spatial relationship between the object and the projector (such as the xyz orientation, position). As it is difficult to precisely maintain an accurate virtual model of a physical space, adjustments are typically needed to correct alignment errors. In most commercially available projection mapping tools this is normally achieved by manually tweaking the physical or virtual scene (see Section 2.6.3).

2.3 Projection Technologies



Figure 14: Appearance augmentation of a toy car using projected light. No projection (left) and projection (right). Photographs from Lee et al. [36].

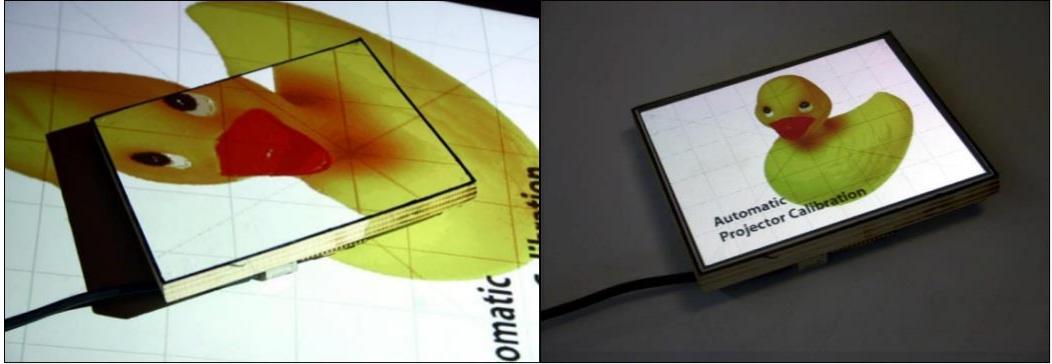


Figure 15: Showing how a standard projected image (left) can be mapped to a specific flat plane (right). Photographs from Lee et al. [36].

As long as the projection surface does not have a shape that occludes the projected light, it is always possible to correct for oblique angles [25]. Lee et al. [36] demonstrate how this can generalise to reflected light and even surfaces that are angled slightly away from the projector. Theoretically, this enables projectors to be mounted anywhere in the environment with respect to the projection surface. However, issues of focus, resolution, and diminishing brightness remain.

A range of different geometric calibration methods have been proposed. Molyneaux [66] presents a classification of projector-camera geometrical calibration methods based on Borkowski [24]. This has been updated (Figure 16) to include commercially available depth cameras.

2.3 Projection Technologies

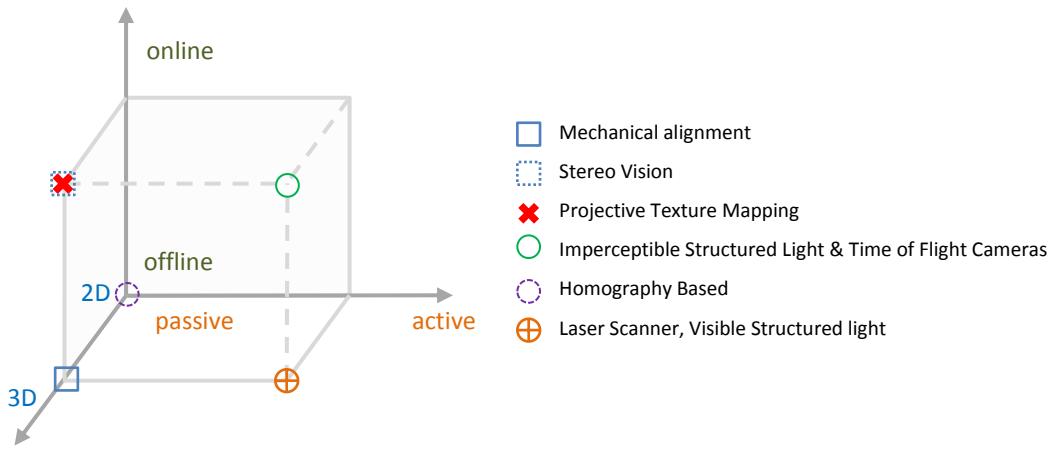


Figure 16: Classification of projector-camera calibration methods. Updated from Molyneaux [66] based on Borkowski [24]. 2D / 3D refers to the spatial dimensionality of the computed information. Online / Offline describes if calibration can take place while the system is operating (online) or requires an initialisation period (offline). Passive / Active describes if the system transmits extra information (i.e. light) into the environment in order to sense the geometry.

The following subsections focus on homographic and projective texture based approaches as these do not require specialised hardware. Both of these approaches generalise to multi-projector and multi-surface projection as described in Section 2.3.3. Specialised hardware is not discussed as it is beyond the scope of the thesis goals and implementation.

2.3.2.1 Homography Based Texture Mapping

A homography matrix is projective transform that operates as an invertible affine correspondence between two projective planes. In Figure 17, a given point in Plane A (i.e. pixel in a 2D image) is transformed by a homography matrix to produce the corresponding coordinate in Plane B (i.e. pixel in a warped image that when projected, corresponds to a physical surface).

2.3 Projection Technologies

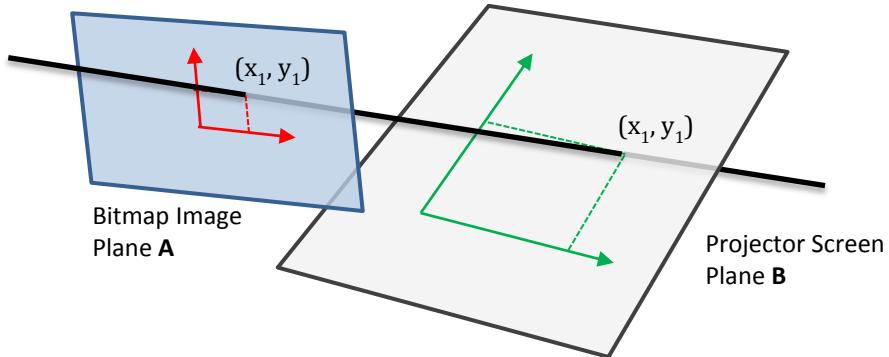


Figure 17: Visualising how a point (x_1, y_1) in Plane A behaves when transformed by a homography to produce the corresponding point in Plane B. Inverting the homography matrix can be used to perform the inverse transformation.

Wren [67] shows how to derive a corresponding homographic transform given the coordinates of the four corners of a quadrilateral in Plane A and the four corresponding coordinates of the second quadrilateral in Plane B.

Sukthankar et al. [68] show how homography based geometric calibration can be achieved automatically using a projector and a camera. Here, they project a series of dots on the screen. By detecting the location of each dot in the camera image, they are able to compute a homography for the projector and camera image planes. Obviously this technique is limited to single static physical planes. However, Lee et al. [69] show how this can be achieved in real-time using systems that are able to track four corners of a display surface or four points of an arbitrary object.

2.3.2.2 Projective Texture Mapping

Projective texture mapping uses a 3D virtual representation of the physical space and a virtual camera (with the same optical properties as the physical projector) is used to render the scene. The virtual camera is positioned to reflect the position and orientation of the physical projector. By exploiting the equivalence properties of the projector and camera, the image seen by the virtual camera will exactly replicate the view covered by the projector in physical space. By rendering virtual content over the relevant parts of the virtual model, this content can be

2.3 Projection Technologies

projected over the physical scene thereby augmenting it with correctly distorted optics. This is illustrated in Figure 18.



Figure 18: Mapping between physical space (left) and virtual space (right) can be used to project rendered virtual content onto a physical surface given an accurate virtual model of the physical space. Figures based on Pinhanez [25].

Strengths of this approach include that it is simpler to model non-planar surfaces such as curves and complex shapes [28]. This approach is relatively simple and extends to dynamic physical scenes [69] and multiple projectors [28] given controlled circumstances and an accurate model of the physical space. The challenge with this approach is that constructing an accurate model and calculating the camera and projector lens properties can be difficult, especially if there are imperfections or non-linear distortions involved. There are a number of methods for automatically creating the required virtual model. Both visible [70] and invisible [71] structured light can be used as a method for unknown surface topology recovery. Recent structured light and time-of-flight based depth cameras have been used to achieve this at improved interactive rates [27].

2.3.2.3 Dynamic Surfaces

Dynamic surfaces include surfaces that move or change size, geometry or rotation at runtime. Enabling planar and non-planar geometric calibration processes to operate at interactive rates incurs a number of challenges [69]. For instance, capturing, modelling, and updating the projected image to match the physical world is a difficult task. If this is not accomplished (A) very quickly and (B) at a stable rate, it can give users the impression of a laggy interface and create a sense of disorientation. Recent advances in high definition object tracking that use steerable

2.3 Projection Technologies

mirrors are able to largely resolve this problem [72], although they do remain an expensive solution limited by the refresh rate of the projector.

2.3.3 Multi-Projector and Multi-Surface Rendering

2.3.3.1 Occlusion

An immediately apparent problem with front-projection is occlusion. Common sources of occlusion are poor projector placement, interacting users, objects, and features of the physical environment. An easy solution to occlusion is to use multiple off-axis projectors. However, each additional projector requires additional geometric calibration. This can lead to redundancy and imbalanced lighting. Summet et al. [73] propose an active system where a camera detects occlusion and enables multiple redundant projectors to fill in the occluded region.

2.3.3.2 Techniques for Combined Projections

Multiple projectors can be automatically combined into a single large addressable canvas using geometric compensation techniques introduced by Raskar et al. [74]. Using a relatively casual placement strategy the projected images may overlap, not necessarily be rectangular, or even aligned (as in Figure 19). Once geometric calibration has been applied, the problem that remains is that the regions where two projections overlap are brighter than regions where they do not.

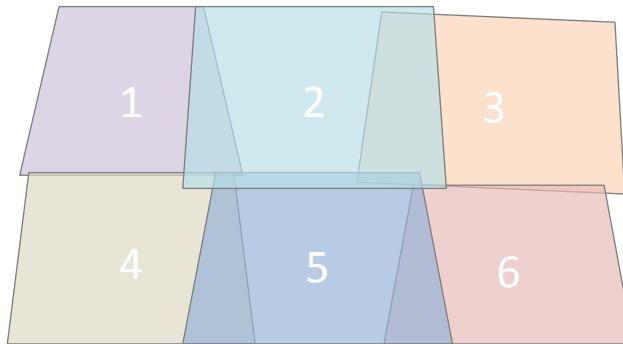
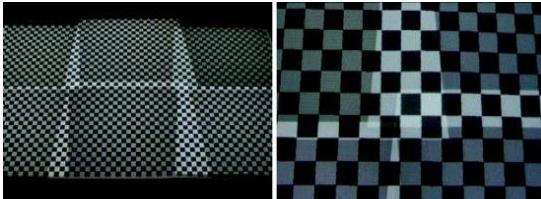


Figure 19: Six overlapping projections. The images are not perfectly rectangular or aligned, creating overlapping projections.

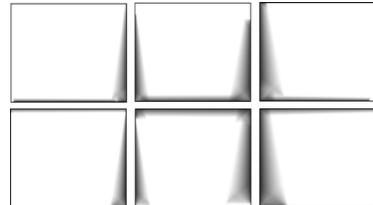
2.3 Projection Technologies

A naïve solution could simply black out the overlapping pixels in all but one of the projectors. This is known as binary masking [75]. A better solution is to use edge blending [38] [76] to create an alpha mask that is applied to the output of each projector. This is able to achieve a smooth blend between all the projections (Figure 20).

(A) Geometric calibration aligns the image.



(B) An edge blending mask for the corresponding projector configuration in A.



(C) Alpha blending adjusts the projection to

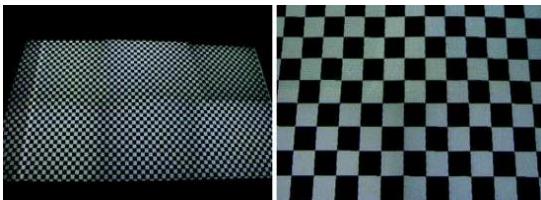


Figure 20: Showing how edge blending can be applied to a six projector configuration with pre-applied geometric calibration. Adapted from Tuddenham et al. [76].

Using a sum based blend function makes the assumption that all projections have the same brightness output capabilities. Although it is possible to manually adjust the brightness setting on some projectors, it caps the brightness to the lowest common denominator and is not always sufficient. An in-depth understanding of the methods and practical issues is presented by Stone [77].

2.3.3.3 Photometric Compensation

Photometric compensation is a form of advanced colour correction that can be used to project consistent colours when using multiple projectors with different colour characteristics, or projecting onto surfaces with different material properties (i.e. colour).

2.3 Projection Technologies

In multi-projector scenarios, as luminance is generally higher near the centre of a projected image, Majumder et al. [78] show how obtaining a measure for the maximum luminance achievable at each pixel can be used to remove spatial variation.

In multi-surface scenarios, given knowledge of the target surface material properties it is possible to exploit the additive colour mixing used by projectors to reduce such effects as shown in Figure 21 [79]. The calibration process uses a camera to capture a surface material's appearance in response to a series of projected colour calibration images.



Figure 21: Colour correction for physical surface material colours. (A) Target projection surface. (B) Projection without photometric compensation. (C) Projection with photometric compensation applied. Figures from Bimber et al. [79].

Commodity projectors currently lack the colour range required to completely remove the effects of saturated (i.e. dark) surfaces. Real-time colour correction in dynamic environments is achievable using GPU techniques described by Grundhöfer and Bimber [80]. Lee et al. show how it is possible to perform photometric compensation on uneven surfaces given a known object transformation relative to the projector and surface topology [70].

2.4 Interaction Technologies

2.3.3.4 Steerable Projection

Steerable projectors (typically a projector-camera unit mounted on a steerable assembly) remain a popular method of achieving in-door multi-surface scenarios in research scenarios [4] [66] [24]. Steerable projection can be created by fitting a pan and tilt mirror to the lens of a projector [4]. Similarly, interaction can be achieved by fitting a sensor to the same mirror, such as a camera. However, unlike traditional vision-based interfaces, steerable systems are required to work seamlessly on distinct surfaces under very different observation angles and lighting conditions [6]. Similarly, they must also be able to cope with changes in observable user interaction caused by transferring the interface between different target surfaces. Although a viable technology, they are not a commodity technology and not commonplace enough for general user adoption. As a result their operation is not extensively discussed in this section.

2.4 Interaction Technologies

This section describes approaches for sensing physical input on interactive projected displays. For each modality, technical methods and techniques are presented, along with user considerations that impact the interaction.

Interaction modalities are discussed in terms of *direct physical interaction* (where the user is in physical contact with the projection surface), *indirect interaction* (where the user may be present but not directly in contact), and *remote interaction* (where the user is geographically separate from the display but is able to interact). This section focuses on direct and indirect interaction as these require physical co-location with the system and are thus considerations of the toolkit. Similarly, non-spatial indirect modalities (i.e. voice recognition) are omitted to retain scope, as are methods that require user augmentation, surface instrumentation, or expensive specialist hardware.

2.4.1 Touch Interaction

The Digital Desk [10] was one of the earliest systems to use optical touch sensing. Wellner used a camera to detect finger position and improved touch detection rates with a microphone to listen for contact sounds. Achieving stable and accurate touch sensing that is comparable to instrumented surfaces is particularly challenging as optical methods are dependent on line-of-sight, high resolution image processing, and lighting conditions.



Figure 22: Camera view of touch interaction with a bucket. (a) RGB camera view, (b) computed image difference, and (c) overlay of square search region, circular button activation region, and rectangular fingertip template match. Figure from Kjeldsen et al. [81].

Kjeldsen [81] and Letessier [40] both implemented methods that use single colour cameras to detect touch by exploiting the shape of fingertips. However, there are a number of challenges associated with visible spectrum optical sensing. For instance, moving a finger through a projected image changes its colour. Techniques based on background subtraction often give unreliable results, as changes in the projected image can overwhelm the inherent colour of the foreground surface [37]. Detection of complex features (such as skin pigmentation changes) requires relatively high resolution image of the fingertip [82]. This typically means placing the camera close to the interaction surface (see Figure 22).

Recent developments have enabled the use of commercially available depth cameras as touch sensors [37]. Although accuracy is still directly correlated with resolution, there are a number of other advantages such as more reliable subtraction between foreground and background, and models of per-pixel depth enable touch on non-flat surfaces [37] and ‘above surface’ interaction [83] [84]. A simple method for touch detection using depth cameras is to apply a threshold [37].

2.4 Interaction Technologies

For flat surfaces, such Figure 23, it is sufficient to model the 3D position and orientation of the surface using a representative plane [8]. However, this model does not account for deviations due to noise in the depth image, variations in surface flatness, or uncorrected lens distortion effects [37]. According to Wilson [37], the noise profile of the Microsoft Kinect is not consistent across the image; making threshold selection difficult.

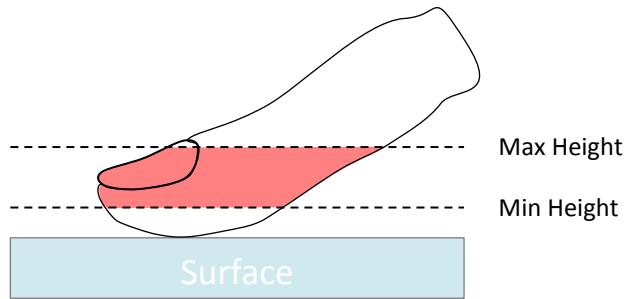


Figure 23: The highlighted area (red) shows the part of the finger that forms a touch event using threshold based sensing between a min and max height from the surface.

To address this problem, Wilson created a per-pixel histogram of raw depth values over several hundred frames of a motionless scene. This revealed that while some parts of the depth image were remarkably stable, others can fluctuate between two adjacent values. He used the upper-values of this histogram as a minimum touch sensing height and obtained the maximum touch sensing height by adding the resting height of a finger lay down on a table. Applying a binary classifier to all pixels in the depth image (according to if they fall within these two thresholds) meant that the resulting data could easily be fed into a tracker to create touch events usable by an application. Wilson's system was tested with a pre-release Microsoft Kinect mounted at 0.75m and 1.5m above a flat table. Observations indicate that the worst case error was approximately 15mm (1.5m height) and 7mm (0.75m height).

Dippon et al. [85] conducted an accuracy test that compared touch detection (using Wilson's method [37] and the libTISCH library [86]) to that of a capacitive touch screen. Their depth camera was mounted 0.75m above the interaction screen. Their results are shown in Figure 24. Although they found accuracy to be worse than that of a capacitive system, they believe it to be good enough for large displays.

2.4 Interaction Technologies

Practical limitations of this technique include that it is only able to operate effectively from highly elevated angles, and additional processing is required to distinguish between touches and other objects placed on a surface. Furthermore, users may not interact with the system while it is being calibrated.

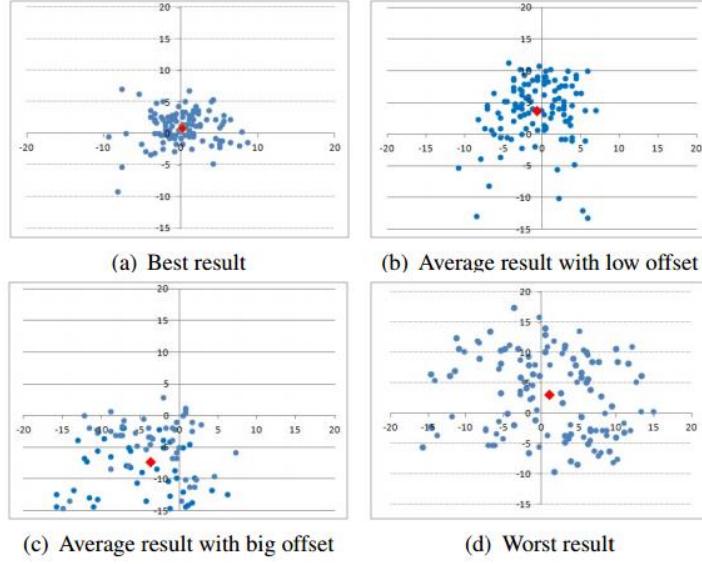


Figure 24: Distribution of touch points measured in the comparative study by Dippon et al. [69] using the Microsoft Kinect mounted 0.75m above the interaction screen. Units are given in mm. Figure from Dippon et al. [69].

Scaling up from single surfaces to physical spaces, Light Widgets [87] enabled room-scale ubiquitous interaction without requiring surface or user augmentation. They used computer vision techniques to search specific areas of an image for disturbances. Live images were streamed from a series of cameras placed around an environment. They approached multi-camera integration by resolving the ‘votes’ for light widget interaction values emit by each camera stream. Skin detection is performed by a lookup of quantized hue and saturation values.

It was possible to create three widgets that offer different kinds of interaction: *Button Light Widgets* (an interactive region), *Linear Light Widgets* (able to select different values based on the part of the region that is intersected), and *Circular Light Widgets* (able to calculate a value based on a radial intersection). A developer could add widgets to a scene by simply drawing them on a snapshot from each

2.4 Interaction Technologies

camera. Although they do not specifically address the accuracy of the interactions in their paper, the limited resolution and positioning of their cameras is not enough to compute a stable position or accurate contact event for finger-scale touch detection.

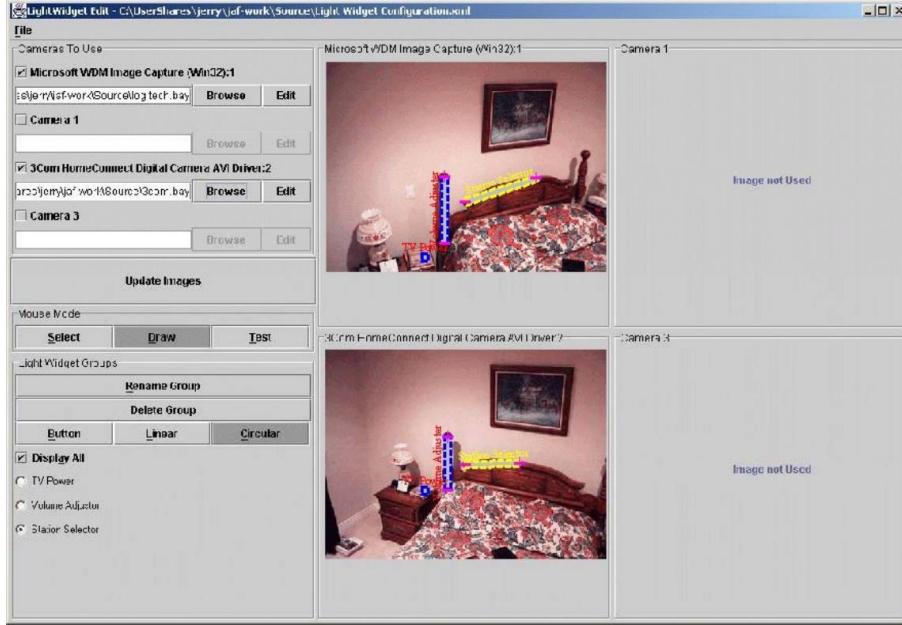


Figure 25: A screenshot of the Light Widgets graphical user interface where a user can select a radio station (yellow horizontal widget), change volume (blue vertical widget), and turn on or off their TV (blue square widget). Figure from Fails et al. [87].

Generally, touch detection with optical methods on uninstrumented surfaces tends to be less accurate, less responsive, and more prone to misinterpreted touches than instrumented alternatives [37] [85]. Below are approaches that can help reduce these issues.

Accuracy: Researchers have proposed several approaches to improving touch accuracy: adding a fixed cursor offset [88], providing on-screen widgets to aid selection [89], and dual-finger interactions for pixel-accurate targeting [90]. However, these approaches require users to learn a new modality or remove the directness of touching on-screen objects. In cases where users of a system cannot afford a learning time and poor accuracy is unavoidable, simply enlarging the target

2.4 Interaction Technologies

touch area is the most effective approach [85]. Using a depth camera⁹ based-touch sensor, Dippon et al. [85] recommend that for a button to be hit by 95% of the touches would need to have a diameter of 28mm.

Responsiveness: Ng et al. [91] studied the effects of direct touch latency on user experience. Modern capacitive touch systems approximately incur a 50-200ms delay between the surface being touched and the display updating in response¹⁰. This creates a poor experience in interfaces that use a ‘dragging’ metaphor. When Ng et al. [91] used a custom touch sensor to reduce this latency to 1ms, it effectively become unperceivable and led to a much improved user experience. Higher latency systems (i.e. depth cameras) should consider using graphical metaphors and interactions that do not rely on the responsiveness of touch input. One strategy is to use a fixed-speed point-to-point animation rather than a user-driven drag. Audio can also be used to improve system feedback [92].

Incorrect Recognition: A system may mistakenly register touches (false-positive) or fail to register touches at all (false-negative). Common source of this are systematic errors, such as sensor noise, and external errors, such as calibration-drift. Improving algorithms, factoring error rates into application design, and better awareness of the sensor constraints are all ways to address this problem.

Misinterpreted Touches: Kjeldsen et al. [26] call this the “*Midas touch problem*”; where incidental gestures are be misinterpreted as commands. They propose reducing this by giving the system knowledge of when to attend to user actions and when to ignore them, perhaps using a presence indicator.

2.4.2 Above Surface Interaction

Restricting interaction to a 2D plane forgoes a wealth of information available above and between screens [83]. For example, finger height, which hand is touching, the angle of the users arm, the user identity, the user height, the surface

⁹ A Microsoft Kinect mounted at 0.75m above the display surface.

¹⁰ For context, a finger moving at 1m/s with a 100ms latency touch sensor would lead to the touch point following behind at a distance of 10cm.

2.4 Interaction Technologies

orientation, and the location of other nearby surfaces. It can also include information about objects near, above or stacked upon the surface [83] [8] [84].

Today's general purpose computer interfaces are predominately single-user. Although multi-touch systems are able to physically support multiple simultaneous users, the interface is not able to fully model the interacting users if it lacks the knowledge of which touches belong to which hand, or which touches belong to which user. For instance, without a model that distinguishes between users, a painting application for two would require both use the same colour at any given time. The dSensingNI framework [84] (described as a toolkit in Section 2.6.1.4) supports user identification, touch detection, hand detection, and object interaction such as grasping, tracking, and stacking.

Hilliges et al. [83] present a technique that allows users to seamlessly switch between interacting with an interactive table and the surface above it. For example, 'picking up' a virtual 3D ball and placing it in a virtual 3D container. Their intention was to use the space above the table to improve the ways people interact with 3D objects. They used virtual shadows as a means of providing feedback to the user. Although these kinds of interaction offer new possibilities, they note that it can also break the direct-interaction metaphor (i.e. touching the surface).

Wilson et al. [8] created '*LightSpace*'—a room augmented with fixed projectors and depth sensors—to explore interaction above and between surfaces. They identify three interaction spaces:

- *Surface Everywhere*: Where all physical surfaces could become interactive displays. For example, tables become interactive tables and walls become interactive walls.
- *The Room as a Computer*: Not only are all physical surfaces interactive but so are the spaces between surfaces. For example, a piece of digital information could be rendered as a projected ball that can be passed between users or placed on other objects (such as a large display) for viewing.
- *The Body as a Display*: Refers to the idea of projecting graphics onto the body to enable interactions in mid-air. For example holding and carrying items of

2.4 Interaction Technologies

data, or creating an information conduit between displays by touching them simultaneously.

2.4.3 Gesture Interaction

As most physical actions involve gesturing at some level, this section is interested in interaction using intentional coarse body gestures, such as arm waving and pointing, as opposed to finger gestures on a touch table. Optical gesture recognition in the context of projected displays is challenging for the same reasons as with detecting touch discussed previously. However, modern depth cameras such as the Microsoft Kinect can reduce the impact of these issues in indoor environments as they use infra-red structured light to extract depth. This has led to the development of tools and libraries that are able to sense the pose of a hand¹¹ and track the skeleton and joint motion of multiple users simultaneously [93].



Figure 26: The 'Put-that-there' system as in 1979. Picture taken from Bolt et al. [94] video: <http://youtu.be/RyBEUyEtxQo>

Early examples of gesture sensing include '*Put-that-there*' by Bolt et al. [94] in 1980 (magnetic sensors) and '*Video Place*' by Krueger et al. [95] in 1985 (optical

¹¹ SigmaNIL: <http://www.sigmanil.com/>

2.4 Interaction Technologies

sensing of shadows). Bolt et al. [94] combined a large-screen graphics display with deictic gestures and speech recognition. A user was able to move simple shapes around a large screen by simultaneously pointing and talking. They note that interaction is more natural—and expression more economic—as a result of the free use of body and pronouns.

Hardy et al. [96] studied real world user responses to gesture based interaction with a wall mounted LCD public display. They found that gesturing to an absolute point in space (informed by camera feedback on a display) is quicker and easier to learn for discrete interactions (such as item selection on a menu) than gestures that use relative kinaesthetic motion. Although the repeatability and dependability of absolute gestures offer a more suitable means of correcting for large errors in menu item selection, relative gestures (such as spinning a scrolling menu) can offer a more intuitive mechanism for small and continuous movements such as error correction. They also found that people do not always successfully adopt the correct interaction technique when presented with clear onscreen instructions (or even a demonstration from another person). In these cases the system would not behave as expected and users tended to gesture more vigorously in response to encountering problems.

2.4.4 Object Interaction

Objects can be used as props to control the interface or as output surfaces. Special markers [5] and actively augmented objects [2] are popular methods for identifying and locating objects in physical spaces. Passive object recognition can be achieved using a range of optical feature detection algorithms that rely on effective and efficient generation of key-points in an image. Popular approaches include the SIFT [97] and SURF [98] algorithms. In order to support object recognition, a toolkit would require a relatively high resolution (i.e. enough to identify shapes and feature points, depending on the choice of algorithm) view of the target object to implement usable feature recognition.

Huber et al. [99] use a web camera and depth camera to track objects in order to displayed projected content (Figure 27). They detect flat surfaces of 3D objects

2.4 Interaction Technologies

(i.e. boxes) and model them as planes in 3D space. A projected image is mapped to the extracted plane using a homography matrix. Using optical flow in the colour image they are able to determine if objects have been rotated.



Figure 27: A photograph is projected onto a box and can be changed by turning a cup (left). Hardware configuration (right). Figure from Huber et al. [99]

IBM's OASIS project investigated combining object recognition with interactive projected displays [100]. For instance, creating an interactive kitchen counter-top that is able to recognise specific ingredients [101]. They detect objects above the surface and are able to use features such as shape, colour, and size to match items in a database. Once an object is recognised, their system can retrieve nutritional information. If two or more objects are recognised, it can suggest recipes. Another example shows a count-down timer placed next to some iced-cream; alerting users that they need to put it back in the freezer before it is too late.



Figure 28: Objects are used to scope interactions (left), and moving a toy train along a surface is used to create a virtual track (right). Figures from Ziola et al. [100].

In terms of general interactions, they found that it is useful to activate interface behaviours based on the proximity of objects. For instance, when two objects are brought together, it is possible to show functions that are relevant to that combination of objects. In that sense, object recognition can be used to scope the

2.4 Interaction Technologies

functionality of a projected interface (Figure 28). It is also possible to have the system interact with objects, such as sliding them along a surface automatically. Patten et al. [102] demonstrate control of physical object positions on a specially augmented surface using magnet arrays. They present a set of interaction techniques that leverage users' mechanical intuition about the behaviour of objects in the physical world (such as adding weight to a movable puck to prevent it from being moved or constraining distance between two objects using a rubber band).

2.4.5 Mobile Devices

The continued wide-scale adoption of smartphones and tablets increases the palette of interactions available to designers of interactive projected displays. Although carrying a mobile device conflicts with the walk-up-and-use scenario, their popularity gives them a practical relevance.

Boring et al. [103] presented the Touch Projector in 2010. As the name suggests, this system allows users to manipulate content on distant displays (typically displays that are unreachable such as large displays outside a window or content on crowded table-tops) by allowing users to interact with a mobile device's camera view of that display. When the user touched the camera feed, their touches were 'projected' from the mobile device back onto the display. Their system uses mobile computer vision and a centralised environment manager server. Schmidt et al. [104] use a 'tap' detected by a smartphone microphone and a touch event registered on an interactive surface to determine if a specific device has made contact with an interactive surface. Interaction using this method can be used to share private identity information as well as content such as photos and video. Their system was implemented using a single FTIR table, a challenge would be to scale the event-pairing algorithm to multiple interactive surfaces. Both Boring et al. [103] and Schmidt et al. [104] design for walk-up-and-use scenarios. However, these are contingent on the wide-scale adoption or standardisation of the relevant mobile application. Another strategy is to use existing standards to support interaction between projections and mobile devices. Davies et al. [105] created a system that

2.4 Interaction Technologies

used the Bluetooth name of a smartphone to inform nearby displays of presence information.

2.4.6 Presence Sensing and Location Tracking

Systems that are able to acknowledge presence of an entity in a given region of space can be used as a simple interaction modality. This can be used for detecting people [96] [106], objects [27] [13], or the lack of either. Presence can be used implicitly (i.e. turn on when a person walks by my display) or explicitly (i.e. tell Story A when user places Toy A on the surface).

Optical presence sensing is able to determine if a marked region of an image is intersected by an object. Simple 2D systems compute this by comparing consecutive key frames for disturbances [96]. More complex systems apply the same intersection principles in 3D using information gathered from a depth camera. Location tracking methods are able to localise one or more objects or persons within a given space. Recent depth-camera based approaches are able to achieve this with accuracy levels suitable for a wide range of applications [93]. As typical with optical sensors, localisation quality diminishes with distance and is subject to line-of-sight and occlusion considerations. Furthermore, sensor placement defines the range of space that can be tracked or tested.

The use of presence information (i.e. a user walking past a display) can be used to create reactive display behaviours that aim to attract attention in public settings [96]. Vogel et al. [106] used proximity information to manage how much content is displayed as users get closer. Alt et al. [107] found that mirrored user silhouettes and images are more effective than avatar-like representations at conveying that a system is interactive. Xiao et al. [27] use object based presence to estimate the number of objects in certain areas (bounded by projected borders). They give the example of collecting the ingredients, such that once all the projected regions are filled, their system can assume that all ingredients have been gathered.

The location of objects can also be used to help position displays on a surface. Cotting et al. [13] use structured light to detect the presence and location of fixed

2.4 Interaction Technologies

and movable object geometry. They use this information to interactively compute new display geometry regions and sizes.

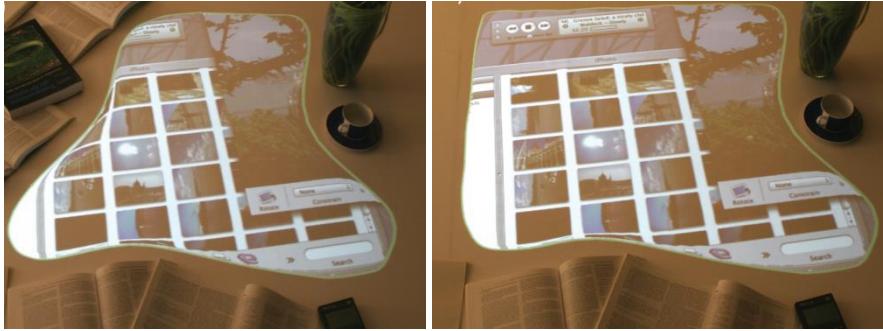


Figure 29: Showing an environment aware display bubble reacting to the presence of books (left) and absence of books (right). Figures from Cotting et al. [13].

Localisation differs from presence in that it contains more precise information about whom or what is where. In terms of localisation of users, Want et al. [108] created '*The Active Badge Location System*' that was able to locate people in an indoor office environment. They describe how one of the most popular uses of their system was for a receptionist to locate a person in the building and then forward a phone-call to their current location. Combining this type of location aware design with interactive projected displays could enable new scenarios where display 'appear' where they are needed and are dismissed if not wanted. This type of location information could also be used to conserve power by turning off displays when people are not likely to be around them.

Greenberg et al. [109] explore how one can design for a proximity and orientation-aware pervasive environment. They argue that spatial relationships are rarely used in interaction design, but can afford many benefits. In particular, they show how a system aware of proximity information can understand and use implicit and explicit interaction techniques. They also show how proxemic interactions can be triggered by continuous movement, or by movement in and out of discrete regions.

2.5 Content Development

The bespoke nature of many interactive projected displays leads to a tight coupling between content, implementation technology, and deployment location [4] [110] [35] [27] [69] [84]. This makes it difficult to both develop new content and transfer existing content to other deployments. This section discusses projection specific design challenges faced by content designers and presents development languages that aim to decouple content from deployment.

2.5.1 Design Challenges

Previous works have explored the use of projected content to support a wide range of goals, including: knowledge work [62], games [111], cooking [100], home automation [112] and more. Modern user expectations of rich multi-media and high-quality production values have increased the complexity of content production for displays in general [113] [114]. This creates problems when implementation divorces designers from the tools they are familiar with. Furthermore, unlike content designed to operate within screen-shaped rectangles, interactive projected displays have a much larger vocabulary of sizes, shapes, and contexts for developers to account for.

2.5.1.1 *User Attitudes toward Interactive Projection*

There are many factors that impact user attitudes towards projected display systems, including assumptions about its purpose and capabilities. To investigate user experience ‘in the wild’, Horneker et al. [53] conducted an ethnographic field study of visitor reactions to a projection based interactive table in a museum. Overall, visitors showed no signs of being intimidated by the table and little hesitation to touch and interact. Some saw it as a toy for children, rather than an information display for visitors. Occasionally users would look up to find the source of the projection and explore how close they needed to bring their hand to the surface in order to trigger the touch. They note how interfaces that do not resemble

2.5 Content Development

typical computer displays evoke a rich repertoire of multi-fingered and bi-manual gestures, although button-like objects evoke mostly pointing and button-pressing. Sensing glitches required museum visitors to invest effort in learning how to work the interface, that could distract from the actual content.

Moving away from rectangular form factors can also have advantages. Digitised touch sensing can also enhance interaction with existing physical objects using interactive projected displays. For example, in '*The Perpetual Cannon*' [115] each note played by a pianist shoots up as a digital '*cannon ball*' and echoes the original note with the same intensity as it falls back down into the key. They argue the content is designed in response to the aesthetic and function of physical furniture.

To examine pervasive projection in domestic environments, Heidrich et al. [112] created an interactive kitchen installation that enabled users to affect lighting and other physical items around a domestic environment such as automatic windows and radios. They collected user responses and attitudes towards the system including usefulness and how easy the interactions were to understand across a range of tasks (such as turning on a light) and recorded data about how often users typically perform these tasks separately from the system.

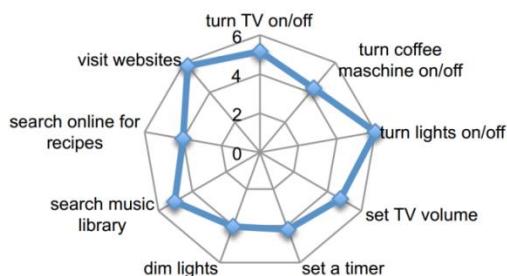


Figure 30: Mean usage frequencies of the intended uses performed without Heidrich et al.'s [112] system (max = 6.0). Figure from Heidrich et al [112].

Users were asked if they thought it would be advantageous for the displays to be available from any surface, or if controlling the system from the kitchen table is sufficient. A total of 25 European users (ages 19-62; μ 29 years) were surveyed. They found the system was perceived as easy to use and they had an overall positive attitude to the system as a domestic technology. They found that users had a "*high*

2.5 Content Development

intention to use the system if it would work on any surface of the home" and that the intention to use the system was highest in a living room scenario than a kitchen scenario. The data regarding which actions the users typically undertake without the assistance of the system is reported in Figure 30. They report no significant correlation between the usage frequencies and the intention to use the system. The authors highlight this as an interesting finding as it suggests that the participants were open to new technologies in domestic spaces. They note more research is needed to corroborate these findings with more users, applications, and to test similar systems in different scenarios.

2.5.1.2 *Framed vs Frameless Interfaces*

In 2005 Pinhanez et al. [12] discussed the role of framed and frameless interface designs. A *frameless* display is a display with no perceptible boundaries. A *framed* display is a display with clearly distinguishable borders. These are compared in Figure 31. Pinhanez et al. propose that frameless displays are a better way of integrating into surrounding environment than framed displays because borders, frames, and whitespace are normally used to define boundaries. However, they make the observation that it can take professional designers a number of design iterations before they are typically able to start "*thinking outside the frame*".



Figure 31: A framed display (left) and a frameless display (right). Photographs from Pinhanez et al. [12].

2.5 Content Development

Acknowledging that they require more scientific verification, Pinhanez et al. [12] also report parameters they identified that impact the design of projected interfaces. In total, 15 guidelines are given (summarised in Figure 32). They conclude that it is difficult to associate an object with functions or properties that are not directly related to their natural usage. They speculate that visual mechanisms for contextualising and decontextualizing information are a more fundamental research issue for pervasive computing than previously thought.

General Case:

1. Use the environment, its objects, and surface elements as part of the interface.
2. Design, if possible, the real world together with the interface.
3. Be aware of the surface being projected onto and its effects.
4. Eliminate the ‘middle’ symbol¹² whenever possible.
5. Avoid implicit framing¹³.
6. Be cautious when using cinema-inspired visual techniques that rely on reference frames.
7. Avoid using scrolling.
8. Be careful when using navigation mechanisms and consider framing information from non-contextually relevant objects such as links.
9. Shoot video against a black background to keep actors’ figures whole.
10. Be cautious when using imagery with perspective.
11. Use sound effects wherever possible.

Applications embedded within an environment:

12. Do not use frameless displays when the information is disconnected from the environment.
13. Be careful when jumping content between surfaces discontinuously.

Applications connected to objects:

14. Be careful about the distance from the object to the display to avoid confusion.
15. Have mechanisms and sensors that are able to reliably confirm that an object is there.

Figure 32: List of design guidelines presented by Pinhanez et al [12].

¹² Symbolic representations of a bridge between the physical and the virtual. If overlay text is projected next to a coffee mug, an additional coffee mug icon is a middle symbol.

¹³ In Figure 31 the character is cut below the shoulders. This creates an implicit framing effect. A better solution would be a floating head.

2.5 Content Development

2.5.1.3 Dynamic and Flexible Form Factors

Displays with dynamic form-factors (i.e. those that can change shape, size, orientation, position in physical space, etc.) present a new range of challenges for content designers (i.e. handling different display sizes, resolutions, aspect ratios, and physical placements). Traditionally research exploring this area restricts itself to hard-coded graphical interfaces as enabling a more open content development process is not the object of study [69] [116].

Lee et al. [69] explore four foldable display designs (Figure 33) including: newspaper, scroll, fan, and umbrella. Simple hard-coded vector based content is responsive to a range of physical indicators such as surface orientation and surface angle. In all of these configurations, the content can be interacted with using an infra-red stylus. The FoldMe project [116] builds on Lee et al's approach to offer more advanced interactions (such as tilting a display to move a slider) although content remains hard-coded.

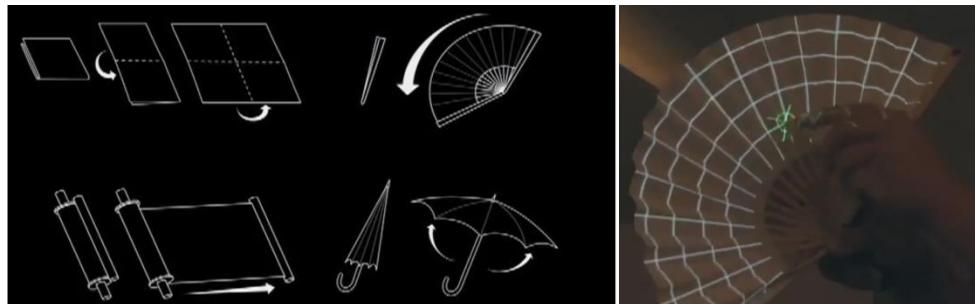


Figure 33: The four foldable display designs (left) and the fan design (right). Figures taken from Lee et al [69].

Kjeldsen et al. [26] explore decoupling the information describing what capabilities an interface provides (i.e. semantic components and spatial layout) from where it appears in an environment/camera image. They argue this provides a straightforward abstraction for the interface designer while facilitating (A) the porting of an application to a new environment where the imaging geometry and interaction surfaces are different, (B) the use of one surface for multiple applications, and (C) the use of the same interface on multiple surfaces. Although

2.5 Content Development

they initially expected to have to support the current GUI style paradigms they saw no reason to confine the interaction to rectangular frames as we are forced to do with monitors.

The future of content design for interactive projected displays remains an open question. For instance, if it is usually better to tailor content for specific physical installations, who should be responsible for doing so? Or should content be designed so that it is generic enough to transfer between displays of different sizes, geometries, interaction modalities, and social settings? Given that most dynamic displays currently exist in research labs, it will likely take more standardisation and experimentation before dominant design contenders emerge. One approach used to create web pages that are able to display themselves appropriately for different form factors is responsive web design [117]. This involves querying device characteristics¹⁴ and then applying conditional styles.

2.5.1.4 Physical Addressing

When projection is no longer restricted to a single planar image (i.e. a presentation on a flat wall) more advanced ways describing how the content should appear in the physical space are required. Physical addressing refers to the way that a content developer can define this information in physical space (i.e. position, orientation, geometry). For instance, one approach would be to have the content developer interact with a large global 3D coordinate system. Three main ways of physical addressing interfaces appear in the literature:

- *Perspective*: This model enables developers to highlight areas of a 2D sensor feed where they want to place interactive content [87]. A strength of this approach is that the developer is directly aware of where they are placing the content from the perspective view of the sensors. A weakness of this approach is that it is difficult to automatically and accurately add new areas.
- *Modelled World*: This approach uses a 3D model of a physical space and developers are able to programmatically [118] [8] or physically [27] place

¹⁴ CSS Media Queries: http://en.wikipedia.org/wiki/Media_queries

2.5 Content Development

content by interacting with the model. A strength of this approach is that can be used programmatically, but a weakness is that it relies on an accurate model that can be complex to query and require many resources to maintain.

- *Named Areas:* This model enables specific objects or areas to be pre-defined so that content can appear on them [119]. A strength of this approach is its simplicity: allowing applications easily select and appear in logical locations (i.e. cooker, sofa, or desk) that are defined by hand to suit the aesthetic of a space. A weakness of this approach is that it restricts content to places defined by the owner of a space. It is also harder to achieve free-form effects such as joining displays together.

Depending on the nature of the application content development complexity can be greatly increased if these abstractions do not cater to their requirements. For instance, a modelled world approach works well for widget based interfaces that need to be aware of each other's location at a room scale. However, when developing more intricate content that interoperates at an object or table scale, positioning large numbers of buttons and sliders relative to a 3D world is very time-consuming process. A better approach in this scenario would be to model a region that automatically generates relative content placement constraints. These issues also impact on the transferability of content and applications between environments. Indeed, is it better for the computer to infer the areas that users would prefer for certain kinds of interaction, or is it better to allow a developer responsible for the space to declare which areas should be used for certain kinds of content? Naturally, the answer depends on the application and usage context.

2.5.2 Development Languages

Due to the complexity of implementation, the content for interactive projected displays is often developed by the same group who developed the technical aspects of the system. Instances where research projects separate the two or offer a simplified content creation process are discussed below.

2.5 Content Development

Kjeldsen et al. [26] envisaged projected interfaces that were composed of individual widgets. They were similar to GUIs and were composed of controls such as scroll bars, buttons, and menus. Each widget would provide a basic type of (optionally parameterised) interaction event, such as touch or slider. Widgets may or not have a projected graphical representation located in the physical space. Furthermore, when the user's task changed, the widgets would also change; just as with current interfaces.

In order to simplify the creation of 'everywhere displays' graphical user interfaces, Kjeldsen et al. [26] proposed decoupling a functional definition of a projected interface from its location in the physical environment. To do this, they created a bespoke XML based mark-up language (VIML) that could be used to define widgets, spatial placement, regions for interaction detection and events that map these interactions onto system functions. The VIML shown in Figure 34 directs the system to set the parameters of a configuration called 'cfg' to create a button named "done" (located at x = 200, y = 200, 50 units large) and a track area named "T1" (located at the origin of the configuration 100 units large).

```
<set id="uniqueID1001">
  <VIconfiguration name="cfg" left="0" right="0" top="500" bottom="500">
    <VIbutton name="done" x="200" y="200" size="50" />
    <VItrackArea name="T1" left="0" right="0" top="50" bottom="50" />
  </VIconfiguration>
</set>
```

Figure 34: A sample VIML configuration as specified by Kjeldsen et al. [26].

When a widget detects a user interaction, it generates an event that reports the event type, configuration, and widget by name. Events are created as XML strings that can be parsed and handled by the application to control the flow of execution. Figure 35 shows a typical VIML event sent by the vision system to the application.

```
<event id="002">
  <VIconfiguration name="selector">
    <VIbutton name="showWhere">
      <VIeventTouch />
    </VIbutton>
  </VIconfiguration>
</event>
```

Figure 35: A sample VIML event as specified by Kjeldsen et al. [26].

2.5 Content Development

More recently, Weigel et al. [119] created ProjectorKit: a lightweight programming library that enables users to place content and interactions using a C# application deployed to a properly instrumented space. It provides support for dynamic objects and addresses projection issues such as pixel density. An example application is specified in Figure 36. Their approach centralises the design of display to a single application. Xiao et al. [27] in WorldKit adopt an approach that enables users to define the spatial location of interactions by ‘painting’ them around their environment. An example application is specified in Figure 37.

```
var book = env.World.Get("Book");

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

var shaking = new ShakeGesture(book, 0.5);
shaking.Recognized += (object sender, ShakingEventArgs e) => {
    /* Code to handle shaking event. */
};

var overlap = new OverlappingDisplays(projector, p2);
overlap.OverlappingChanged += (object sender, OverlappingEventArgs e) => {
    /* Combine views of two projectors. */
    if (e.Display1.PixelDensity <= e.Display2.PixelDensity)
        e.Display1.BlackoutOverlapWith(e.Display2);
    else
        e.Display2.BlackoutOverlapWith(e.Display1);
};
```

Figure 36: Example ProjectorKit C# application which attaches a projected image to a physical book and listens to shake events and resolves overlapping displays by blacking out regions of lowest pixel density. Code taken from Weigel et al. [119].

2.5 Content Development

```
import worldkit.Application;
import worldkit.interactors.Button;
import worldkit.interactors.ContactInput.ContactEventArgs;
import worldkit.util.EventListener;

public class OneButtonApp extends Application
{
    Button button;
    public void init() {
        button = new Button(this);
        button.contactDownEvent.add(
            new EventListener<ContactEventArgs>() {
                @Override
                public void handleEvent(Object sender, ContactEventArgs args) {
                    System.err.println("Got a button event!");
                }
            });
        button.paintedInstantiation("OneButton");
    }
    /* Boilerplate */
    public static void main(String[] args) {
        new OneButtonApp().run();
    }
}
```

Figure 37: Example WorldKit Java application code consisting of a single button. Code from Xiao et al. [27].

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. Furthermore, designers experimenting with interactive projected displays (such as Burrell-Saward et al's [110] Display Cabinet in Figure 38) often make use of advanced animated graphics and various forms of connectivity.



Figure 38: Showing Display Cabinet. Placing different RFID tagged tokens on a physical surface triggers changes in smooth animated graphics. Photographs by Burrell-Saward et al. [110].

2.6 Existing Toolkits

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 [120], 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. Although this clearly makes them an attractive option for content creation, web browsers do not have access to the native platform and thus cannot easily be used to access the underlying hardware.

In terms of content interaction, TUIO is an open framework that defines a common protocol and API for tangible multi-touch surfaces [121]. Despite being a community standard, a drawback of TUIO is that developers must implement the support at a low level. A challenge for those developing content languages for interactive projected displays will be to move away from the bespoke, complex implementations that require the use of ‘involved’ and compiled programming languages.

2.6 Existing Toolkits

This section describes existing toolkits from the academic, industrial, and hobbyist communities. Over the last 10 years many tools and platforms have emerged that can be used to prototype pervasive computing concepts faster, and in more detail than was previously possible [11] [15]. Although aspects of pervasive computing are open to community driven design processes [7] interactive projected displays have consistently lacked support. Existing toolkits tend to focus on supporting specific scenarios (i.e. projection mapping). Figure 39 shows the focus of the toolkit presented in this thesis positioned relative to the key areas it draws together.

2.6 Existing Toolkits

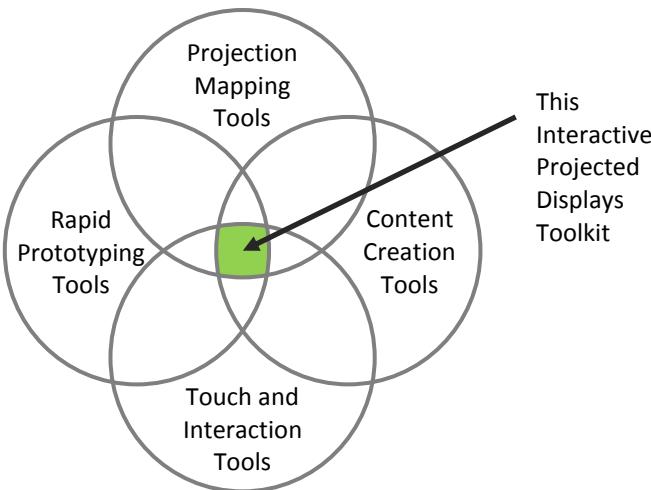


Figure 39: Showing how this toolkit is positioned relative to other important tools in the area.

Table 3 identifies three classes of popular toolkits: combined projection and interaction, touch sensing, and projection mapping. The toolkits were included in the table based on their capabilities and their significance to the goals of this thesis. Tools for choreographing interactive spaces (such as Rockwell Group's Space Brew [122] and Open TSPS [123]) are not included as they do not specifically address projection. Similarly, content creation frameworks such as Open Frameworks [124], Processing [125], and Max [126] (tools that simplify advanced graphical concepts and promote 'creative coding') are not included for the same reason. All of the aforementioned tools enjoy active support communities and are worthy of note.

Syphon [127] is a particularly noteworthy software tool that enables graphical output to be captured from one source (i.e. a popular media viewer or slideshow application) and channelled into other software that implements the Syphon protocol (i.e. a projection mapping tool). Although it is not without limitations, it allows non-programmers to easily share rich visuals between different applications without prior knowledge of each other's existence.

Table 3: Comparison of existing toolkits grouped by type. This table focuses on the most popular and fully featured relevant tools and is by no means exhaustive.

Toolkit	Appeared	Main User	Available	Main Interaction	Content Support	Community	Download Link
<i>Combined Projection and Interaction</i>							
WorldKit [27]	2013	CS Academic	Private	Hands	Java Code	No	chrisharrison.net/Research/WorldKit
ProjectorKit [119]	2013	CS Academic	Unknown	Gestures	C# Code	No	grouplab.cpsc.ucalgary.ca/cookbook/
dSensingNI [84]	2011	CS Academic	Unknown	Multi-Touch	C# Code	Private	dsensingni.de/
Wiimote Whiteboard	2007	CS Hobbyist	Public	IR Pen	OS Shell	Yes	johnnylee.net/projects/wii/
<i>Touch Sensing</i>							
TESIS [128]	2011	CS Academic	Private	Multi-Touch	TUIO Events	No	Link Not Available
CCV	2009	CS Academic	LGPL	Multi-Touch	TUIO Events	Yes	ccv.nuigroup.com/
Ludique's Kinect Bundle	2012	CS Hobbyist	zlib	Multi-Touch	TUIO Events	Yes	code.google.com/p/lkb-kinect-bundle/
<i>Projection Mapping and Content Creation</i>							
VVVV	1998	Artistic / Pro	Commercial	Via Extensions	C# Code	Yes	vvvv.org/
Mad Mapper	2011	Artistic / Pro	Commercial	Via Extensions	Video Source	Yes	madmapper.com/
VPT 6.0	2011	Artistic / Pro	Unknown	Via Extensions	Video Source	Yes	hcgilje.wordpress.com/vpt/
Multi Projector Mapper	2013	Artistic / Pro	BSD	Via Extensions	Java Code	Yes	github.com/arisona/mpm

2.6 Existing Toolkits

2.6.1 Combined Projection and Interaction

2.6.1.1 *WorldKit*

WorldKit¹⁵ [27] allows users to ‘paint’ interactive features onto everyday surfaces. The system uses a Microsoft Kinect depth camera and projector to create a paired pre-calibrated unit. Unlike most of the other toolkits presented in this section, it is not publically available either as source code or a binary. The painting interaction involves a user brushing their hand over a surface in order to instantiate controls. This operation is shown in Figure 40. Once controls have been painted onto a surface, users are able to interact with them in a number of different ways.

Although it appears that the widget selection process and Java applications that run on it are hardcoded (see Figure 37), it is flexible enough to explore a range of scenarios.



Figure 40: Showing photographs of the WorldKit system in operation [27]. Left: TV programme, volume and light controls. Centre: painting a radial control onto a workbench. Right: painting a presence detector onto a door. Related Video: <http://www.youtube.com/watch?v=BBQPA5fsLTA>

2.6.1.2 *ProjectorKit*

ProjectorKit¹⁶ eases rapid-prototyping of interactive cross-device and multi-display applications with mobile projectors [119]. The toolkit addresses the problem that applying projector-based techniques within an application is cumbersome and time-consuming. To do this they identify five interaction primitives that serve as building blocks for a large set of applications. These primitives are implemented

¹⁵ WorldKit: <http://www.chrisharrison.net/index.php/Research/WorldKit>

¹⁶ ProjectorKit: <http://grouplab.cpsc.ucalgary.ca/cookbook/index.php/Toolkits/ProjectorKit>

2.6 Existing Toolkits

using automated jitter and keystone correction, projection mapping of textures, hotspot and targeting events, projector and object gestures and overlapping events. The toolkit can be used to build C# applications that listen to high-level interaction events.

In terms of hardware the current ProjectorKit implementation requires high-end external tracking hardware to track the positions and orientations of mobile projectors. The authors note that this is sufficient for prototyping and testing such applications, but does not allow for real-world deployment. Although there is no official community, the authors have made the code open source and are corresponding.

2.6.1.3 *Wiimote Whiteboard*

Wiimote Whiteboard¹⁷ is a software tool that uses the Nintendo Wii Remote¹⁸ to create low-cost interactive whiteboards. The system uses the Wii Remote's infrared camera (with built-in hardware blob tracking of up to 4 points at 100Hz) to locate light emit from an IR-pen from the 2D perspective of the Wii Remote. These 2D points are then transmitted to a computer via Bluetooth where they are transformed by a known calibration in order to generate a mouse event for the Windows Desktop. This transformation step enables the WiiRemote to be positioned with a non-perpendicular view of the interaction surface. Due to the lack of support for multi-touch in operating systems of the day, a separate multi-touch demo was provided as a custom C# DirectX program.

Although a relatively simple concept, the tool was adopted widely. The impact was particularly felt in the hobbyist, education and, academic communities as it reduced the requirements for creating an interactive surface to a projector, Wii Remote, and a simple IR-LED circuit built into a marker pen. The tool has a

¹⁷ Wiimote Whiteboard: <http://johnnylee.net/projects/wii/>

¹⁸ Nintendo Wii Remote: http://en.wikipedia.org/wiki/Wii_Remote

2.6 Existing Toolkits

discussion and support community called WiimoteProject¹⁹ that has operated since January 2008 and has almost 30,000 registered members²⁰.

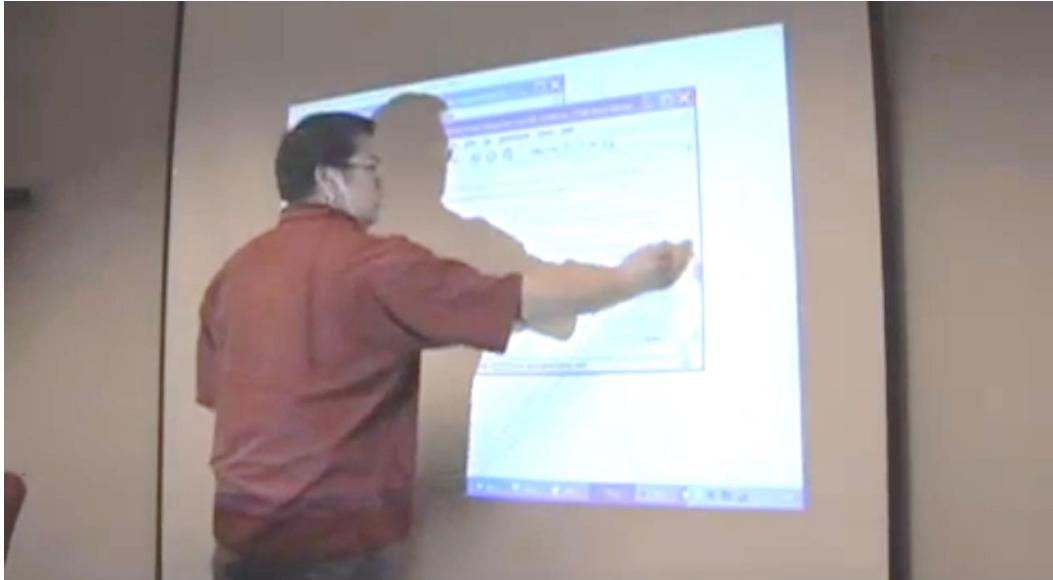


Figure 41: Demonstrating the Wiimote Whiteboard software. Screenshot taken from: <http://youtu.be/5s5EvhHy7eQ>

2.6.1.4 dSensingNI

The dSensingNI²¹ framework [84] supports multitouch on arbitrary surfaces, freehand gestures and tangible interaction using a depth camera. Although primarily developed for use with tabletops it can also be used to create vertical installations such as interactive walls and white boards. Limitations of the system include that it does not support projection mapping (i.e. the projector needs to be directly above the interaction surface) and only supports single surfaces.

dSensingNI is a powerful system due to the broad palette of interactions it exposes to application programmers. To transmit the tracking data between dSensingNI and a client application, the software uses the TUIO protocol. Although this reduces the hardware requirements to a PC, a projector and a commercially

¹⁹ WiimoteProject: <http://www.wiimoteproject.com/>

²⁰ Internet forums are prone to spam and robotic accounts. For that reason it is difficult to confirm the accuracy of this number.

²¹ dSensingNI Framework: <http://www.dsensingni.de/>

2.6 Existing Toolkits

available depth camera, it still requires advanced programming skills in order to be able to develop applications. A C# library is provided that can present the decoded TUIO messages as application events. However, these events are not tied into the .NET events stack.

dSensingNI has been published academically [84] and is now available (on request) for academic and non-commercial use (see screenshot in Figure 42). There is also a closed support forum available for users of the framework.

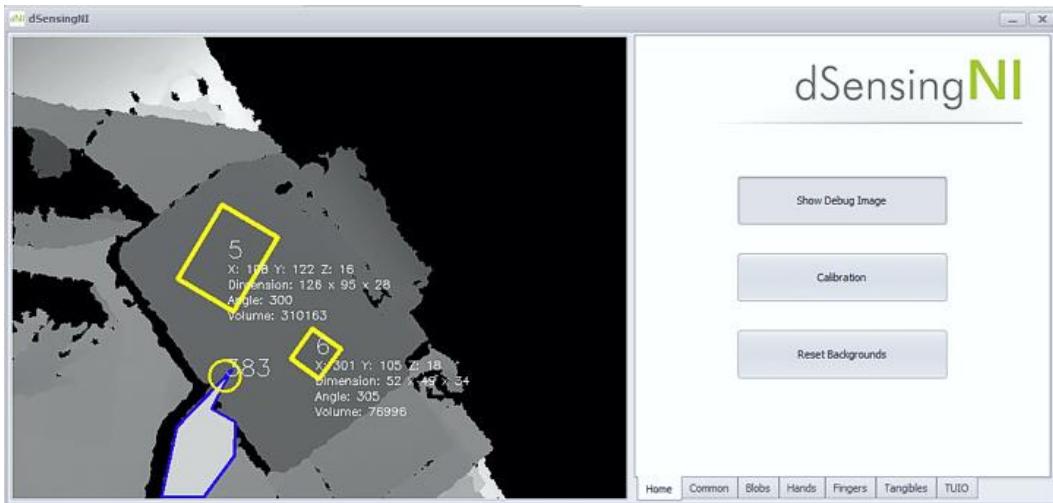


Figure 42: Screenshot of the dSensingNI framework taken from: <http://www.dsensingni.de>

2.6.2 Touch Sensing

2.6.2.1 TESIS

TESIS (Turn Every Surface into an Interactive Surface) is a portable device demonstrated at ITS2011 [128] that enables every surface beneath it (both flat and non-flat) to be turned into an interactive surface. The device integrates a pico-projector and depth camera into a lamp-styled object that enables the projector and sensor to appear directly above or in-front of the interaction surface. The projector is connected to a computer in order to display a user interface.

2.6 Existing Toolkits

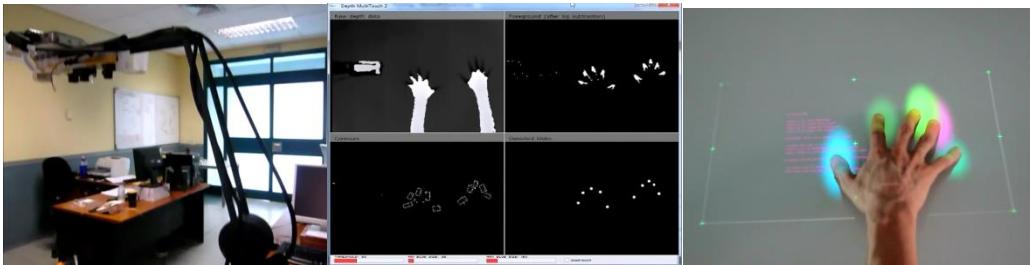


Figure 43: Left: TESIS lamp device. Center: Showing the touch points extracted from the depth image. Right: The detected multi-touch points. Screenshots taken from: <http://youtu.be/wWg-CKj5Dmo>

Internally, this uses the CCV multi-touch tracking solution in order to process touch events. With little effort, this can be used generate operating system touch events that can be used with any application. Limitations of the system include that it does not support projection mapping (i.e. the projector needs to be directly above the interaction surface) and only supports single surfaces. Although the author has demonstrated the device at a number of events, the project is not available for download either as binary or in source code format.

2.6.2.2 Community Core Vision (CCV)

Community Core Vision²² is a general purpose open source and cross-platform solution with a particular focus on touch sensing. It is very popular within the touch table community. The software operates by processing a video input stream (typically a view from an IR camera) and outputs tracking data (such as touch coordinates and blob size) as TUIO events (Figure 44). The main limitation of CCV in a pervasive projection context is its lack of support for multiple unconnected surfaces. However, it does enable users to stitch multiple camera views together.

CCV is known for its ability to interface with a variety of cameras and supports many multi-touch lighting techniques including: FTIR, DI, DSI, and LLP. Expansions are also available for the Microsoft Kinect that use Wilson's [37] method of touch sensing. It is primarily intended for use within the academic community, although

²² CCV: <http://ccv.nuigroup.com/>

2.6 Existing Toolkits

the NUI Group Community is a popular home for many hobbyists and a source of touch sensing discussion with interests beyond the CCV tool.

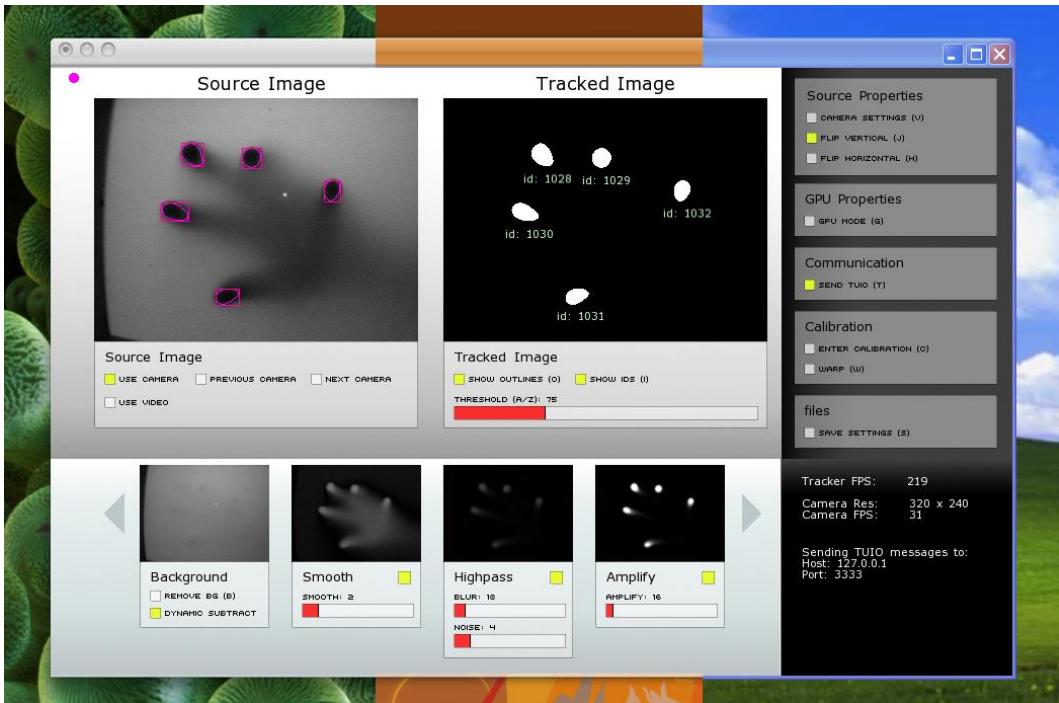


Figure 44: The CCV tool running on OSX, Linux, and Windows. Screenshot source: <http://ccv.nuigroup.com/>.

2.6.2.3 Ludique's Kinect Bundle

Ludique's Kinect Bundle²³ is a hobbyist toolkit that uses a variety of sensing methods to create multi-touch surfaces using the Microsoft Kinect. The toolkit is able to communicate with other applications by transmitting touch events using the TUIO protocol. Unlike other touch toolkits, LKB does not require the user to position the depth camera directly above or in-front of the interaction surface. However, like the others, its main limitation is that it only supports the use of a single surface and requires programmers to implement applications that support the TUIO protocol.

The toolkit was released under the open source zLib licence in May 2012 and has received over 1000 downloads. There is a community discussion group where

²³ LKB: <https://code.google.com/p/lkb-kinect-bundle/>

2.6 Existing Toolkits

users can request support, although this is not as active those offered by other toolkits.

2.6.3 Projection Mapping and Content Creation

2.6.3.1 VVVV

VVVV²⁴ is a multi-purpose hybrid graphical and textual programming environment for easy prototyping and development of graphics. It first appeared in 1998 in response to a need to simplify the programming process for interactive media installations and is particularly adept at handling large media environments. It has support for real-time graphics, audio, and video that has led to it being popular in the television, music, and arts communities. It is free for non-commercial use and offers a range of commercial licensing options. The toolkit can be extended with new ‘nodes’ that control external devices such as lights, switches, and touch screen monitors. A screenshot of its use is shown in Figure 45.

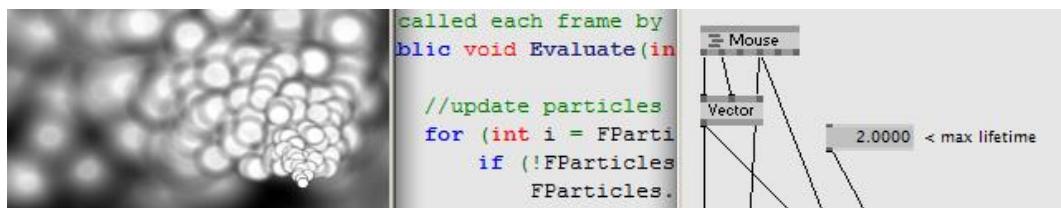


Figure 45: The VVVV code editor. Left: output window. Centre: code editor. Right: visual program structure. Screenshot taken from <http://vvvv.org/screenshots>.

Although VVVV is able to create a wide range of installations, the process for doing so can get complex and users require training in order to create more complex structures and projection mapped content. The software itself enjoys an active developer community. External modules exist that support the use of the Microsoft Kinect as a gesture, skeleton, and image mask sensor. However, it does not readily support touch interaction with the projection.

²⁴ VVVV: <http://vvvv.org/>

2.6 Existing Toolkits

2.6.3.2 Mad Mapper

Mad Mapper²⁵ aims to provide a simple and easy tool for projection video mapping. Its focus is on simplifying the projection mapping process (using tools such as VVVV, can be quite complex) so that artists and designers can focus on creating content. A screenshot of this process is shown in Figure 46. Users are able to select regions of an input stream, and transform them onto surfaces in a projection.

Mad Mapper relies on the Syphon framework [127] as a source of real-time video content. By outsourcing the content it is possible to support a variety of different types of interactivity.

Like VVVV, Mad Mapper offers a free non-commercial license as well as a commercial license. They offer free email support to all users and also maintain a forum for community support.

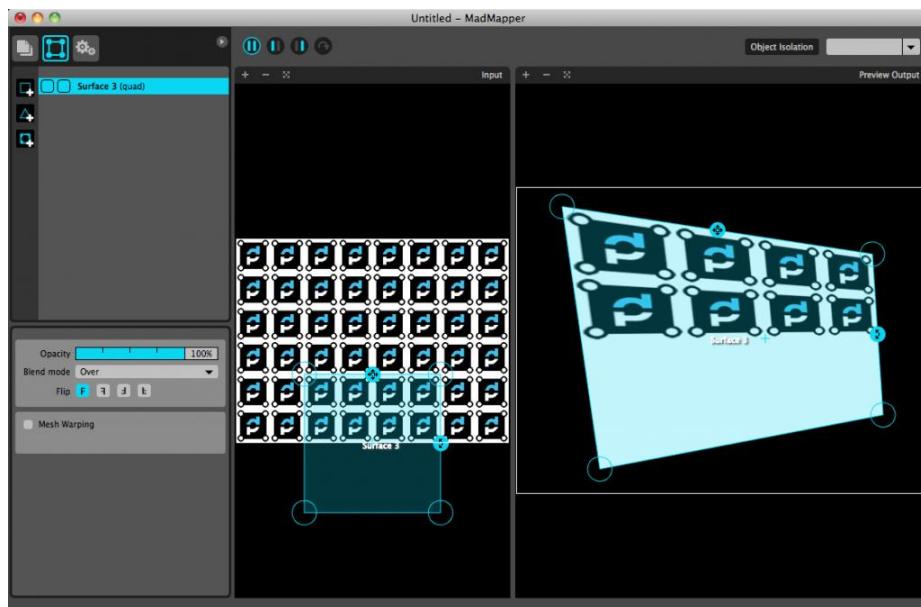


Figure 46: Showing the Mad Mapper tool. Here a region of a surface is selected (blue rectangle of the centre frame) and the output transformed to align with a physical surface in the projection window (right frame). Screenshot taken from: <http://www.madmapper.com/basic-introduction/>.

²⁵ Mad Mapper: <http://www.madmapper.com>

2.6 Existing Toolkits

2.6.3.3 VPT 6.0

VPT 7.0²⁶ is a free multi-purpose real-time projection tool that is popular within the theatre and arts communities. Like Mad Mapper, it features a graphical interface for positioning, scaling and distorting up to 32 projection layers. VPT can support up to eight video sources and two live Syphon [127] sources, in addition to a number of others, such as a noise source. A number of extensions are available that make it easier to work with serial devices such as physical switches. These can act as video triggers for interactive elements. Although VPT's interface is graphical (Figure 47) it is relatively complex and requires training to understand.

Unlike most other tools, VPT is completely free for both non-commercial and commercial use.

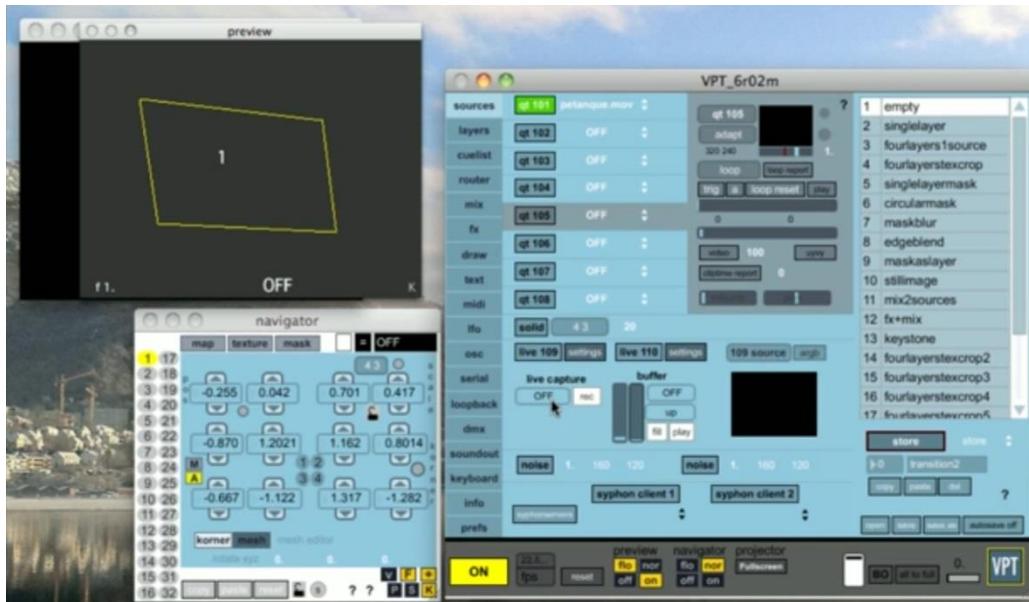


Figure 47: Screenshot from the VPT getting started tutorial: <http://youtu.be/atR6c0R0xKM>

2.6.3.4 Multi-Projector-Mapper

Multi-Projector Mapper²⁷ is an open-source software framework for 3D projection mapping using multiple projectors. It attempts to close the loop between

²⁶ VPT 7.0: <http://hcgilje.wordpress.com/vpt/>

²⁷ Multi-Projector Mapper: <http://www.arisona.ch/web/mpm/>

2.7 Chapter Summary

3D projection mapping and 3D scanning using the Microsoft Kinect. It contains a basic rendering infrastructure and interactive tools for multi-projector calibration. To achieve this calibration, six circular calibration points in 3D space need to be matched to their physical counterparts for each projector. This relies on a physical cube of a known size being placed in the scene (Figure 48).

Applications that use this framework are developed using Java code. Although the calibration process is relatively user friendly, the application development process is still complex and requires an in-depth understanding of the technical processes involved. The software is available under a BSD license and is actively supported by its authors at ETH Zurich's Future Cities Laboratory in Singapore.

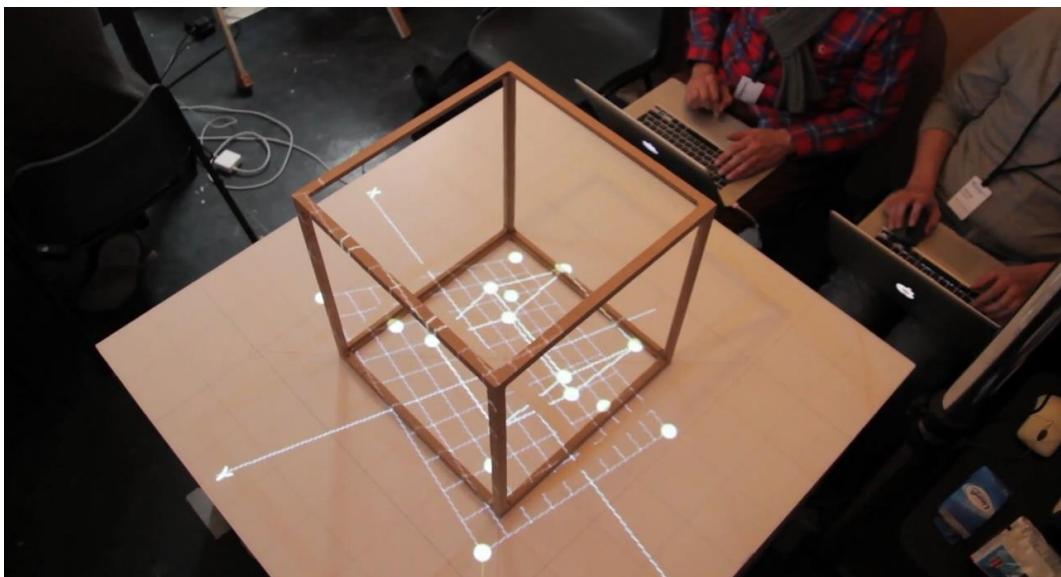


Figure 48: MPM 3D projection mapping calibration using a physical 3D cube. Screenshot taken from: <http://vimeo.com/65130490>

2.7 Chapter Summary

This chapter describes interactive projected displays and introduces a range of implementation technologies, user interactions, and content development issues. Section 2.2 describes influential systems and visions that motivate the use of

2.7 Chapter Summary

interactive projected displays. Rapid prototyping tools [32] [19] can encourage user innovation in this space [15] by simplifying the concepts and removing the technological barriers that prevent people from engaging.

The projection and interaction technologies sections (Section 2.3 and 2.4) discussed hardware and software technologies that can be used to create interactive projected displays. These focus on low-cost commodity hardware and scenarios that do not require user or surface augmentation. Although many of the technical challenges are now well understood, creating systems that combine them remains difficult—even with existing toolkits. More research is needed to improve the quality of hand interaction with the projected imagery and the quick calibration and deployment of instrumented projection mapped displays. In the toolkit’s design, the complexity of many of these technologies would be abstracted away from the users. Most works in this space focus on technical contributions and short interaction scenarios (i.e. those in controlled lab environments). This motivates research that understands applied and long term use of interactive projected displays.

The content development section (Section 2.5) covered the design challenges facing content designers for interactive projected displays. It also discusses existing specialist development languages and their limitations in the context of a toolkit. More work is needed to improve the decoupling between the content and underlying implementation technologies. Furthermore, interactive projected displays appear in many different configurations, sizes, locations, form-factors, and often interoperate with external systems. More research is needed to create systems that enable content to select its own configuration and interaction modality based on its surroundings and user context.

The last section presented existing toolkits (Section 2.6). Although the discussion is not exhaustive, it aims to be representative of what is available in the academic, industrial, and hobbyist communities. A comparison table of these tools is presented in Table 3.

The breadth of this chapter reflects the range of interaction styles and application scenarios that interactive projected displays enable. Naturally, such a wide range is difficult to support with a single toolkit, and attempting to do so would

2.7 Chapter Summary

likely result in conflicting requirements and a confused design. To address this, a requirements scope is needed in order to focus the toolkit. However, to derive an appropriate scope, the next chapter explores a range of applied interactive projected display characteristics and application scenarios based on the works presented in this chapter. This will inform considerations, requirements, and challenges for applied interactive projected displays and assist reasoning about valuable toolkit design.

Chapter 3. Research Probes

3.1 Overview

Chapter 2 identified a diverse range of interactive projected displays that enable a wide set of new interactions and application scenarios. However, to determine important features for a toolkit focused on enabling user innovation more research is needed. This chapter provides rationalisation for the requirements and design decisions in the next chapter through two generative and evaluative application driven research probes. The first probe explores a new collaborative software development environment and the second explores long term use of an interactive office desk. Both apply interactive projected displays to application scenarios and were selected as they address a wide range of display characteristics described in the previous chapter.

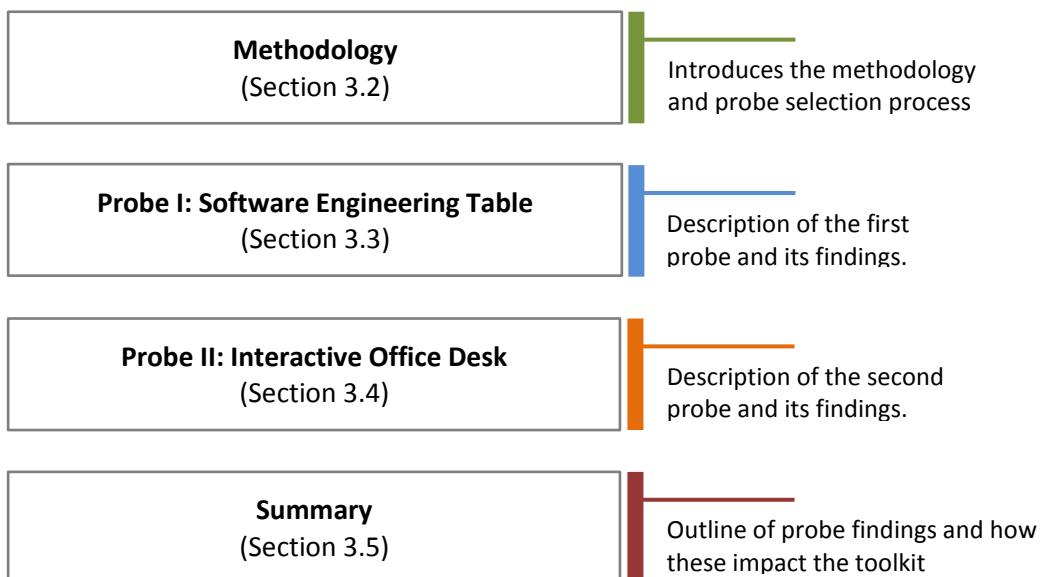


Figure 49: Structure of the research probes chapter.

3.2 Methodology

The contributions of this chapter are: (1) experience of building and deploying interactive projected displays, (2) a deeper understanding of important display characteristics in the aforementioned real world application domains, and (3), research findings in targeted application domains enabled by the introduction of an interactive projected display. The chapter structure is shown in Figure 49. Section 3.2 discusses methodology, including a description of the probes, justification for their selection, and a format for their presentation and analysis. Sections 3.3 and 3.4 each describe a research probe and discuss its findings. The chapter concludes with a summary of contributions (Section 3.5).

3.2 Methodology

This research informs the toolkit design by examining interactive projected displays in application driven research scenarios. A probe based methodology is adopted based on its ability to concurrently address two main challenges:

1. *It is unclear which features of interactive projected displays will result in valuable toolkit features.* Probes reduce the need to speculate about how different display and application characteristics generate value in application scenarios through grounded examples.
2. *There are many display and application scenario features that require exploration.* Examining every combination of display and problem domain is not practical. Probes sample the problem space and divide it into manageable areas of study.

3.2 Methodology

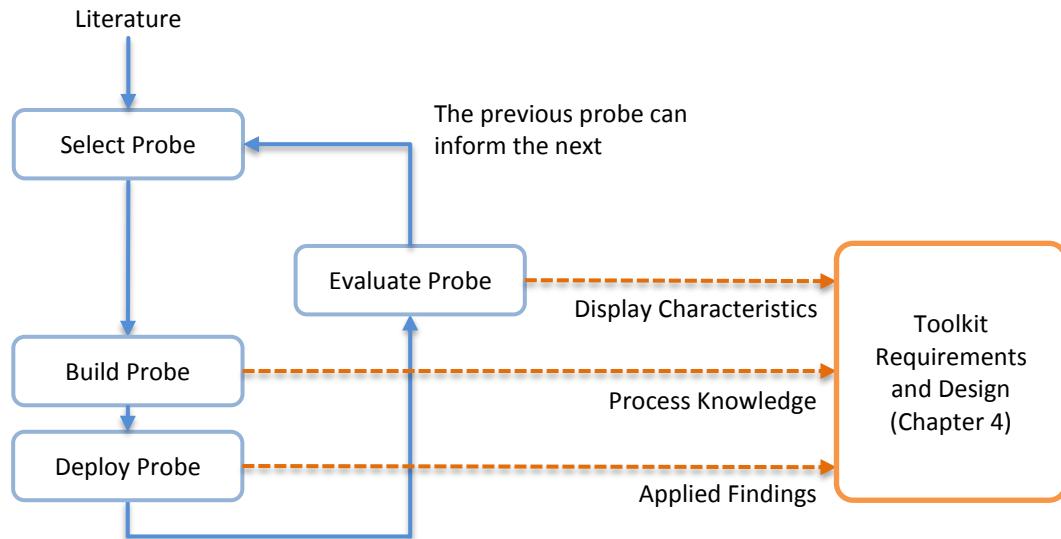


Figure 50: Illustrated overview of the probe based methodology used in this chapter. Solid lines indicate the process of iteratively conducting probes. Dashed lines indicate the generated knowledge that rationalises design decisions in the next chapter.

The methodology presented in Figure 50 addresses these challenges simultaneously. The process of producing and deploying application driven research generates valuable information about the process which is often lost [16], identifies important toolkit features, and the findings from the probes themselves can also provide insights which translate into toolkit requirements. All of this information is captured and fed into the design decisions in Chapter 4. A limitation of this approach is that it relies on the representativeness and quality of the knowledge generated by the probes. If this knowledge is not ecologically valid, it is harder to use it when reasoning about toolkit designs. Resolving this imposes certain characteristics on the probes which are discussed in the next section.

3.2.1 Probe Anatomy

To provide ecological validity, each probe is a fully in-depth application driven research project. To that end, the internal methodology of each probe is defined by the needs of the application it supports. Treating each probe as an applied thesis

3.2 Methodology

contribution—as opposed to simply emulating the application driven research process—has several advantages which are available through two primary outputs:

1. *Display Artefacts*: Novel interactive projected displays that are representative of a selection of display and application scenario characteristics (see Section 3.2.2). These generate applied research findings and knowledge about the development process.
2. *Publication*: An item of peer-reviewed academic literature that reports the findings of the probe to its respective problem domain. These describe how display characteristics generate value for the academic community.

These outputs are advantageous because physically developing and deploying prototypes grounds concepts, technologies, and development processes in reality. The depth of these outputs warrants that the knowledge gained is ecologically valid as application driven research. However, increasing the complexity of each probe reduces the number that can practically be conducted, thus making a representative probe selection is important.

3.2.2 Probe Selection

To increase the extent to which the probes represent the characteristics of interactive projected displays in the literature, distinguishing interface and application characteristics from Section 2.2.7 are cross-referenced in Table 4 with the influential systems from Section 2.2.

It is possible to reduce the number of probes by discounting characteristics which could limit the flexibility of the toolkit. Spatial AR (highlighted red in Table 4) is one such characteristic. It uses interface metaphors that are based on phenomena in the physical world such as lights and shadows [11] [5] [2]. Toolkit users (and content designers) are likely to have considerably more experience with the elements of symbolic AR (i.e. icons, graphics, and text). To avoid a confused design and minimised applicability the probes concentrate on exploring symbolic AR.

3.2 Methodology

Table 4: Cross reference of applied projected display characteristics with visions described in Section 2.2. Spatial AR systems are highlighted and frequencies of each characteristic are shown at the bottom.

Vision Characteristics	Interface						Application					
	Multi Device	Frameless Design	Dynamic Geometry	Body Interaction	Device Interaction	Tangible Elements	Symbolic AR	Spatial AR	Public Use	Private Use	Collaborative	Task Specific
TUI metaDESK [2]						✓	✓		✓	✓		✓
TUI ambientROOM [2]	✓	✓				✓	✓		✓			
Office of the Future [3]	✓				✓		✓		✓	✓		✓
Shader Lamps [11]	✓	✓	✓		✓		✓		✓			
Luminous Room [5]	✓	✓	✓			✓	✓		✓			
Everywhere Displays [4]		✓	✓						✓			
Digital Desk [10]				✓	✓				✓			
Augmented Surfaces [9]	✓	✓			✓	✓			✓	✓		
Total Frequencies	5	5	3	1	4	4	7	3	3	8	3	3
Non Spatial AR Frequencies	2	2	1	1	2	2	5	-	2	5	2	3

Of the symbolic AR systems shown in Table 4, all are suitable for private or semi-private spaces (5/5 cases). This is followed by a task specific design (3/5 cases). The next most common characteristics are: multiple devices, device based interaction, tangible elements, frameless projection designs, and a collaborative function (2/5 cases). The least common are dynamic geometry (1/5 cases) and direct body interaction (1/5 cases). Although the frequency of a characteristic does not necessarily equate to its importance, it can be used as a basis for representative probe selection. For instance, almost half of the systems are collaborative, thus one probe could examine a collaborative context, and the other a single user context.

3.2 Methodology

3.2.3 Probe Overview

Probe I examines the use of interactive projected displays in a co-located collaborative software development scenario. This addresses task-focused and collaborative characteristics through a controlled experiment.

Probe II examines the long term use of interactive projected displays in an office computing scenario. It uses involved observation to help communicate a rich account of user experience. Probe I and Probe II have the most in common with the Augmented Surfaces [9] and Digital Desk [10] projects respectively, as described in Section 2.2.

Table 5: Mapping interface, application, and study process characteristics onto the chosen probes.

Probe Characteristics	Probe Name	Interface						Application				Project	
		Multi Device	Frameless Design	Dynamic Geometry	Body Interaction	Device Interaction	Tangible Elements	Public Use	Private Use	Collaborative	Task Specific	Long Term Study	Single Developer
Characteristic Number		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
<i>Probe I:</i> Software Engineering Table		✓	✓	✓		✓			✓	✓	✓		
<i>Probe II:</i> Interactive Office Desk						✓	✓		✓			✓	✓

Table 5 maps interface, application, and study process characteristics onto the probes. The ‘project’ category expands the range of research project characteristics that are covered. The ‘long and short term study’ characteristic was added as many systems discussed in Chapter 2 are evaluated in short-term lab studies rather than over extended periods of use. The ‘single and multiple developers’ characteristic considers scenarios when system construction involves collaboration.

Sections 3.3 and 3.4 each report probes as application driven research projects (i.e. motivation, goals, design, development, implementation, and evaluation) by describing the findings of the probe. Figure 51 shows the structure of these sections.

3.3 Probe I: Software Engineering Table

The first four sub-sections (green, top) describe the probe details and the last two (blue, bottom) discuss toolkit findings and open questions for the next probe. At the end of this chapter (Section 3.5) the findings are summarised in terms of a toolkit.

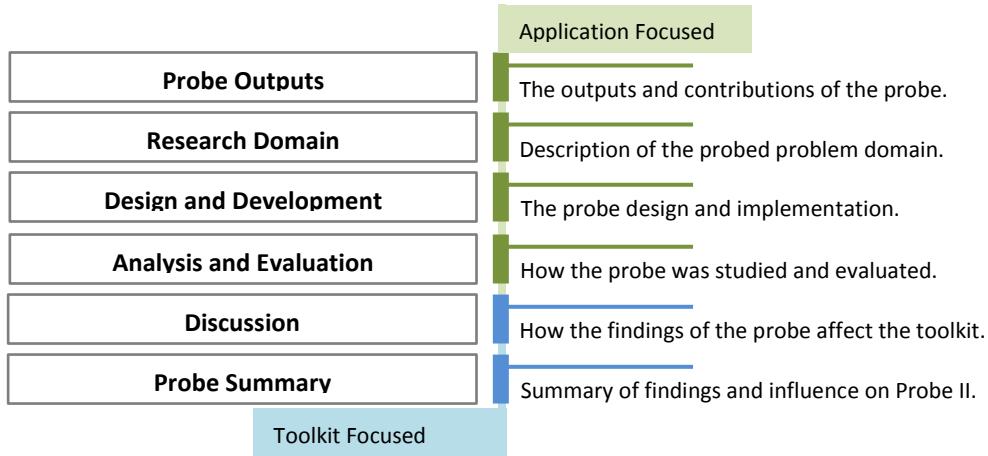


Figure 51: Probe structure. Green subsections are application focused. Blue sections are toolkit focused.

3.3 Probe I: Software Engineering Table

3.3.1 Introduction and Goals

The first probe is based on a large multi-user interactive table designed to improve the process of collaborative software development for co-located developers. Known as CoffeeTable, the system was fully functional and enabled developers to collaborate in the creation, compilation, and testing of Java²⁸ desktop applications (Figure 52). The probe was developed and evaluated over a total period of 6 months and involved two authors²⁹. The evaluation focused on a comparative study between classic individual programming, pair programming, and programming using the table. The outputs of this probe are shown in Table 6.

²⁸ Java programming language: <http://www.java.com/en/>

²⁹ The primary author of the CoffeeTable probe is the author of this thesis. The other author is Christopher Bull, a Ph.D student researching ‘playful’ software engineering and studio environments for software development [180].

3.3 Probe I: Software Engineering Table

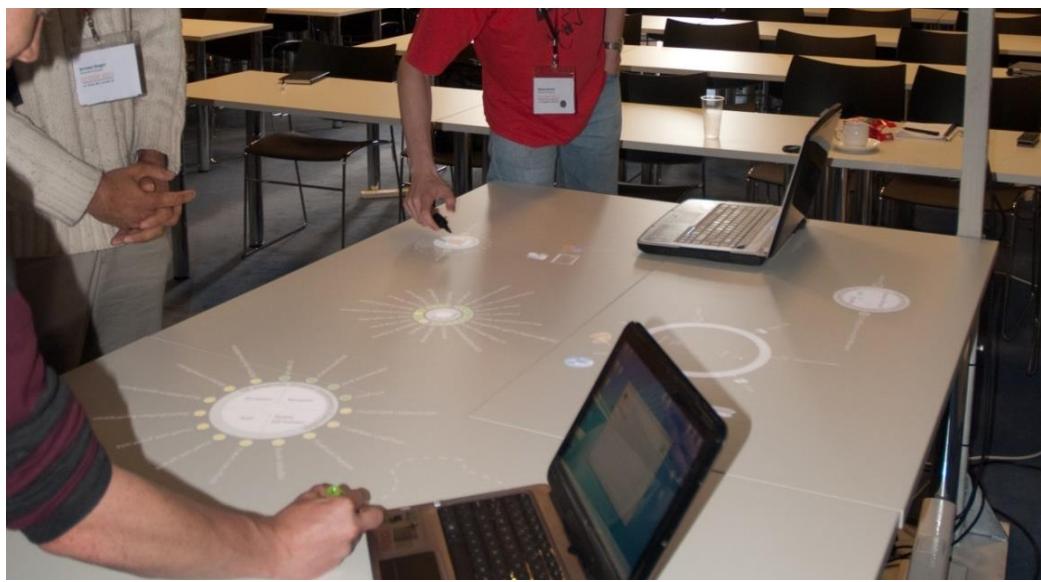


Figure 52: Photograph of the Software Engineering Table (CoffeeTable) taken at the ECOOP'11 conference.
Photo Credit: Christopher Bull.

Table 6: Overview of the outputs of Probe I.

	Name	Description	Picture
Display Artefacts			
1	CoffeeTable: Interactive Projected Display (Hardware and Software)	A large top-projection interactive table with bespoke distributed IDE software. http://highwire-dtc.com/coffeetable/	
2	WiiTUO Toolkit (Software)	An open source tool which enables IR stylus multi-point interaction on Windows 7. https://code.google.com/p/wiituo/	
Research Papers			
3	Digitally Annexing Desk Space for Software Development	Short Paper describing the display and findings published at ICSE'11 [129]. http://dx.doi.org/10.1145/1985793.1985910	

3.3 Probe I: Software Engineering Table

3.3.2 Research Domain

Interactive projection is used in this probe to examine a fundamental dilemma in modern integrated development environment (IDE) design: software engineering is a fundamentally collaborative activity, yet the programmer's key tools and training are designed for soloists [130]. Although researchers have investigated a number of desktop applications [131] [132] [133] [130], CoffeeTable uses an interactive projected display to create a new collaborative IDE designed to support co-located software engineering. The probe has three research goals:

Goal 1) *Minimise production bottlenecks through features that encourage the integration of agile and traditional practice.* Agile methods recognise the inevitability of change, emphasise active stakeholder involvement and use short iterations as a basis for rapid system delivery. Traditional models emphasise predictability, accountability, and control. Rather than thinking of these as separate practices, this goal investigates if and how the CoffeeTable workspace encourages developers to transition between different working practices; choosing the best mode of working to suit the task at hand.

Goal 2) *Investigate how developers interact with a large shared visualisation of software architecture and working process.* Studies have shown that programmers spend a significant amount of their time navigating code and other development resources [134] [135] [132]. CoffeeTable aims to transform previously individual resources into shared inter-personal boundary objects [136]. This goal investigates the impact a large shared interactive visualisation (showing both architectural and workflow process information) has on the software design process.

Goal 3) *Examine the impact of a collaborative workspace on developer performance, quality, and project awareness.* Sharing a physical workspace typically invokes social interactions. Visualisations in this space are able to act as a reference frames for shared understanding [137] [138]. This goal investigates how combinations of these factors can affect developer performance, quality, and awareness of the actions of others.

3.3 Probe I: Software Engineering Table

3.3.3 Design and Development

CoffeeTable is physically composed of a standard white 1.8x1.4m table situated in an office environment (semi-private use, Table 5, C8). Using an Infocus™ IN1503 short throw projector mounted in the ceiling alongside two Nintendo™ Wii Remotes™ to sense stylus input, the table hosts a large interactive visualisation of software architecture (Figure 53). Pairs of developers use laptops on the table (multi-device, C1) and synchronously collaborate (collaborative, C9) on a single live revision of a software project (task specific, C10). The desk space serves a dual purpose: functioning as both a place to put laptops in a collaborative form factor and as an interactive visual representation of the software architecture and workflow.



Figure 53: Top: Ceiling mounted InFocus IN1503 short throw projector. Left: Table in software studio context. Right: Table visualisation, developer laptops, and participant experimental task brief.

Developers use IR styluses to literally drag elements of the interactive visualisation around the table and onto their laptops so that they can work on them in a private space (device based interaction, C6). The same dragging technique is also used to freely arrange items on the table (dynamic geometry, C3) using a set of

3.3 Probe I: Software Engineering Table

uniform rotate-scale-translate control graphics. The visualisation can be spread over the entire visible surface with no explicit borders other than the edge of the table (frameless design, C2) although some of the visualisation items use symbolic frames (i.e. circles) to indicate logical boundaries (i.e. code windows).

3.3.3.1 Hardware and Software

The two main computational components: the *table* and the *developer laptops* are shown in Figure 54. The *table* software contains a network server, a model of the source code being developed, and the CoffeeTable visualisation. This is hosted on a commodity desktop PC running Windows 7™. The *developer laptops* are standard Windows 7™ laptops running CoffeeTable client software which is able to communicate with the table network server. Developers can use this software to physically write code and interact with the visualisation.

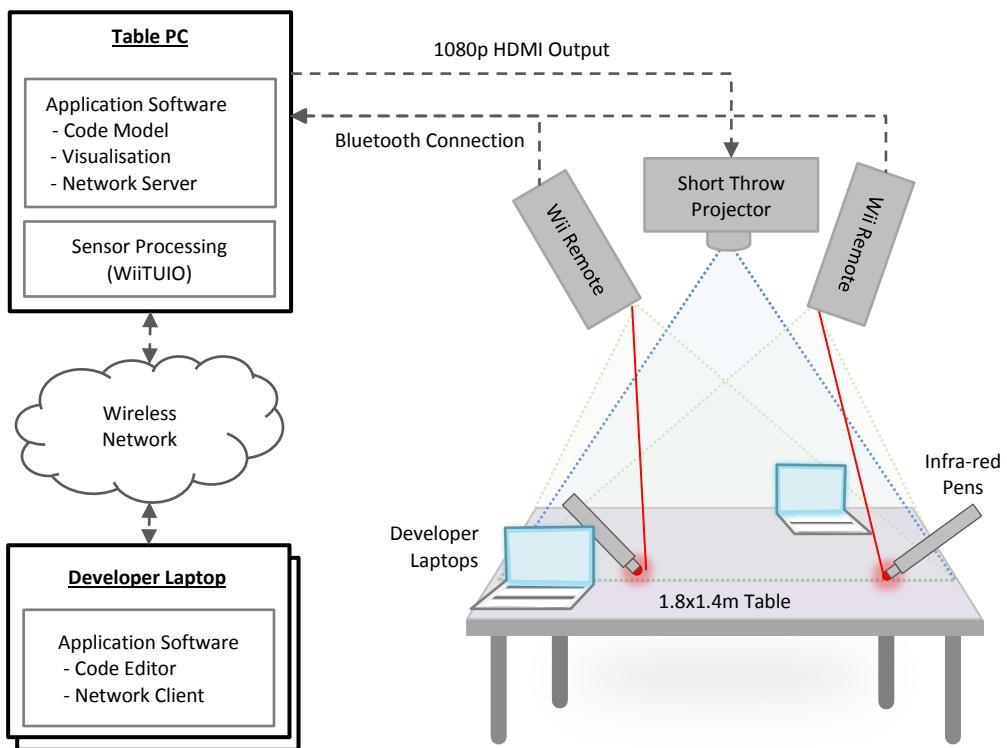


Figure 54: Overview of the CoffeeTable hardware and software architecture.

3.3 Probe I: Software Engineering Table

All of the software was written using C# and the Windows Presentation Foundation³⁰ (WPF). There are two distinct layers to the software: *application software* and *sensor processing*. These are discussed below.

Application Software

The *application software* is responsible for maintaining a model of the software being developed, user interaction with that model, compilation of the software, distribution to the laptops, and rendering an interactive visualisation of the model and those actions. The system architecture is based on a centralised distributed system and updates are synchronised through a distributed event based model-view-controller pattern. The developer laptops wirelessly connect to a table server that hosts the model.

The model manages the source code being developed and a range of metadata (i.e. authorship, access rights, etc.) used in the visualisation. This is implemented in a separate library to the table view which is responsible for rendering the visualisation and user interaction. The same is true for the code editor which handles interaction with the model on developer laptops. Both views communicate to the model though a custom network server (TCP socket based) which acts as a controller.

To populate the visualisation, key information about the source is extracted by parsing raw blocks of Java code contained in the model (i.e. method names, parameters, comments) which have been created by developers working at the table. Compilation and execution of the source code is performed on the developer laptops by executing the standard '*javac*' and '*java*' commands distributed with the standard JDK³¹. As the structure of the source code is not stored in files, or thus line-numbers, this limits the ability to detect and report certain code errors. To address this, the output of these commands is redirected to the CoffeeTable client editor,

³⁰ WPF is a unified programming model for building user interfaces on the Microsoft Windows platform: <http://msdn.microsoft.com/en-us/library/ms754130.aspx>

³¹ Java Development Kit: <http://www.oracle.com/technetwork/java>

3.3 Probe I: Software Engineering Table

which locates the line numbers that refer to the location of any errors and replaces them with an on screen button in the code editor that accesses the erroneous code.

The code editor on the developer laptops takes the format of a traditional desktop application. It is integrated into the operation of the table visualisation such that when a developer takes content from the table and physically drops it onto their laptop, that content (i.e. a method) will appear in a fully editable text field with syntax highlighting. When content is being edited in this way, it is locked to other developers. To inform others that this is the case, when another user attempts to access locked code, the table visualisation renders a line which points to the editing user.

Save and load support are provided through serialisable wrappers at the model level. Visualisation features such as syntax highlighting and web browsers compatible with affine transformations (required for rotation and scaling) were provided by AvalonEdit³² and off-screen web page rendering library Awesomium³³. Support for these features is particularly challenging due to the complex rendering architectures of the external libraries [139]. In terms of table interaction, the multi-touch support provided by Windows 7™ focuses on multi-finger gestures for single users rather than multiple users each with a single interaction point. Subsequently, a new set of simple interaction controls were developed and implemented into the CoffeeTable application software that provide selection, translation, scaling, and rotation functionality.

Sensor Processing

The *sensor processing* layer (WiiTUIO) is responsible for turning the standard table surface into a large multi-point interactive surface. This operates by calculating the position of custom-built IR light pens using multiple Nintendo™ Wii Remotes™. This data is then classified, transformed, and smoothed for conversion into Windows 7™ compatible multi-touch events. This is based on the method demonstrated by Lee [22] with two major enhancements: (1) simultaneous input

³² AvalonEdit editor: <https://github.com/icsharpcode/SharpDevelop/wiki/AvalonEdit>

³³ Awesomium off-screen Web Renderer: <http://www.awesomium.com/>

3.3 Probe I: Software Engineering Table

from multiple pens to enable multi-user interaction, and (2) multiple Wii Remote sensor support to extend the range of the interactive surface to larger surfaces. The first was achieved using a spatio-temporal point coherence ranking system and the second was achieved by synchronising and mapping multiple sensors directly over the same coordinate system.

3.3.3.2 Visualisation

The interactive visualisation is based around a series of symbolic elements with specific purposes and functionalities. These are overviewed in Figure 55 and described below.

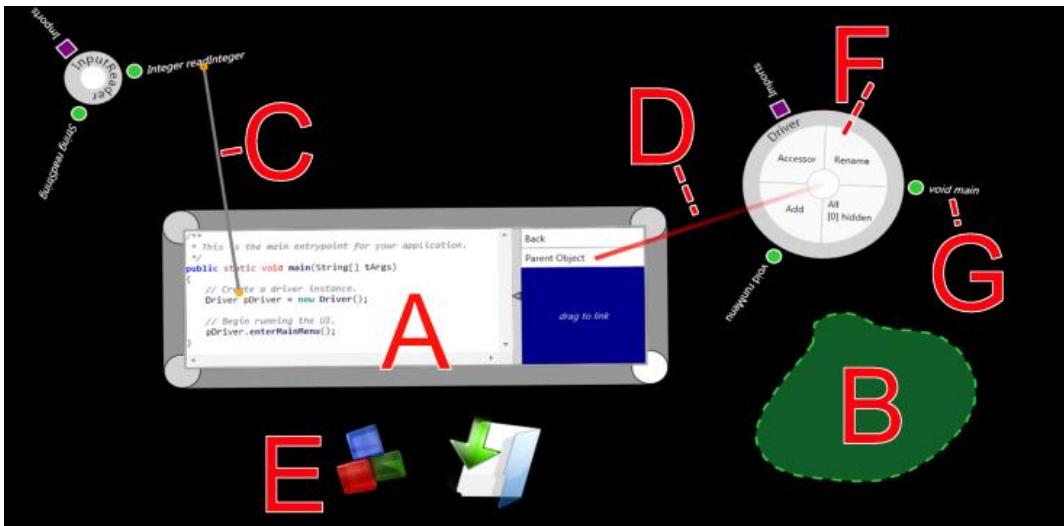


Figure 55. Screenshot of widgets in the CoffeeTable visualisation projected onto the table. The letters correspond to widget descriptions below. Black background reflects the lack of implicit borders.

(A) Window – Windows are divided into two sections: a content panel and a semantic link panel. The content panel displays rich media (i.e. code editor, web browser, or drawing canvas). The semantic link panel references more content of relevance to the content panel. This acts as an interface to a bi-directional weakly connected content graph that is automatically constructed as the programmer works.

3.3 Probe I: Software Engineering Table

(B) User Areas – User areas are a mechanism for exchanging information between the visualisation and specific developer laptops. For example, a developer drawing a shape around their laptop using the stylus and then presses a ‘connect’ button to bind the two together. Elements of the table visualisation can then be ‘dropped’ into this area and accessed through the application software on the laptop.

(C) Linker Band – These visualise the process of information exchange within the system as a direct result of stylus interaction. Uses include moving items (i.e. methods, objects, and documentation) to, from, and around the visualisation.

(D) Highlight Lines – A simple glowing line used to illustrate the state and location of an action or some information. Red lines show the location of ‘locked’ items. Green lines show the movement of an item to and from a user area. Yellow lines highlight requests for input, and blue lines indicate compile requests.

(E) Icons – Represent ‘significant actions’ within the visualisation. For example, the block icons in Figure 55 are used to create new software objects. Other examples include: close, delete and compile.

(F) Object – Represents a software object within a project. Protruding arms show internal members such as fields, methods, and documentation. Objects can be moved by dragging the centre circle or resized and reoriented by manipulating the border. Centrally located accelerators are used to filter members, create accessor methods, and refactor details.

(G) Internal Members – Internal members (i.e. fields, methods, documentation, or even inner classes) symbolise the contents of parent software objects. They can be edited in private (locked) by being dragged into a user area or in a public space by being dragged onto empty table space to open a window.

3.3.4 Analysis and Evaluation

To achieve the probe goals it was necessary to create a scenario which would challenge developers to collaborate on a single piece of complex source code. A controlled repeated measures single-factor experiment compared developer performance and behaviours when using three collaborative coding techniques

3.3 Probe I: Software Engineering Table

(Figure 56): pair-programming (Eclipse IDE), classic programming (Eclipse IDE + SVN), and CoffeeTable.

3.3.4.1 Experimental Setting

A total of six developers participated in groups of two, paired based on the organisation they worked in. The participants were all software developers from local organisations with at least one year of Java development experience in either an academic (Group 1) or industrial (Group 2 and Group 3) context. All developers were male and signed a consent form.



Figure 56: Experimental conditions. Left: CoffeeTable. Centre: Classic programming. Right: Pair programming.

Each group undertook in three programming tasks (one for each collaborative coding technique). Each task lasted for 45 minutes and was presented to the developers as a mock requirements specification. This was designed to help emulate a real world task and contained a mixture of simplistic and complex features as well as basic user interface design. The tasks were balanced to be similar in complexity but challenging enough to require multiple developers to complete most features in the allotted time³⁴. To reduce carryover effects a latin square was used to counterbalance the tasks and groups.

The analysis (undertaken by both researchers) is based on quantifiable code metrics such as the number of features completed and the complexity of their implementation [140]. These are used to give an impression of their performance in the different conditions. A custom tool was used to code video footage to extract periods of time spent typing, talking, and engaging with the table. These records are

³⁴ Tasks were derivative of challenges set by the British Informatics Olympiad:
<http://www.olympiad.org.uk/>

3.3 Probe I: Software Engineering Table

supplemented by follow-up interview feedback in addition to an expert code quality evaluation (conducted by an academic software engineer) that assessed factors such as architectural decisions, maintainability, and readability.

3.3.4.2 Findings

Task Completion Level and Quality

The classic programming style offered the best overall feature completion level at the expense of code quality. Pair programming produced the overall highest level of solution quality, at the expense of fewer fully completed complex features. The table provided a middle ground where fewer basic features were completed but the more complex features were completed to a higher standard (Figure 57). In terms of modularity, readability, and weighted-method-count complexity (Figure 58) table solutions were generally of a higher code quality than those produced using classic or pair programming methods.

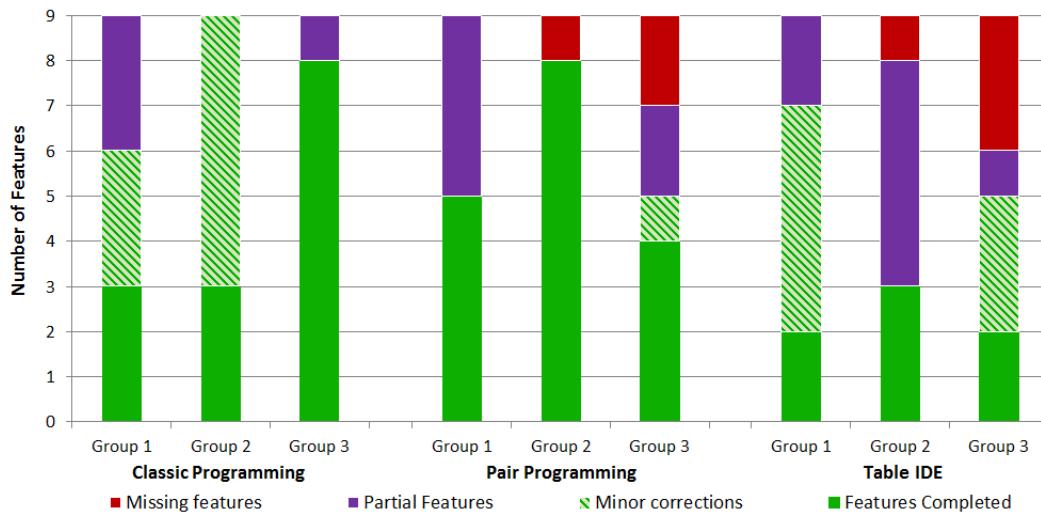


Figure 57: Feature completion levels for each condition. Green: Fully completed feature. Green Shade: Feature is not integrated into the interface or requires minor corrections. Purple: Feature has errors or does not compile. Red: Feature is missing.

According to the expert code review, the most common code errors made when using the table were: (1) uninstantiated variables, (2) lack of integration into the

3.3 Probe I: Software Engineering Table

user interface and, (3) out of order syntax errors. In the pair and classic programming tasks developers made extensive use of in-line debugging and autocomplete features which were not present in the CoffeeTable system. This could account for many of the minor and partial errors in Figure 57 and yields an important design lesson for further experiments in the area.

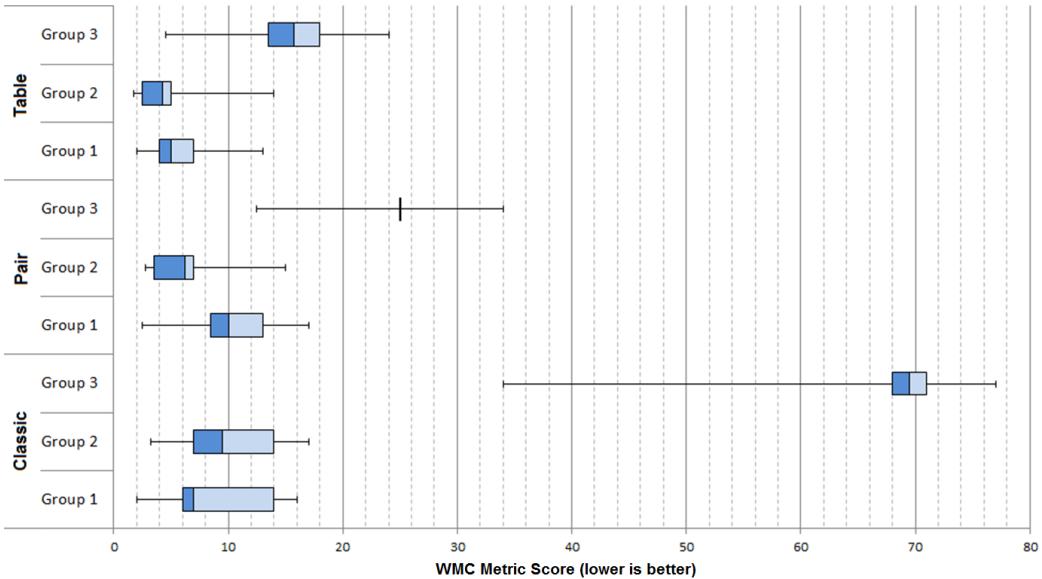


Figure 58: The average weighted method counts per-class for each group and condition. The WMC metric is a numeric indicator (the sum of the complexities of all class methods) of how much effort is required to develop and maintain a particular class. A higher WMC indicates that the class is likely to be harder to reuse and maintain [140].

Impact on Code Structure

The expert code review uncovered a correlation between the frequency of a given software pattern and the time it took to implement. For example, CoffeeTable featured an accelerator for generating variable assessors, and code produced on the table saw increased use of these methods. In comparison, code developed in the classic or pair style saw more direct variable access as developers would have to scroll around the source page to add them as they worked. By identifying and managing these overheads, it may be possible to create tools which encourage certain coding practices.

3.3 Probe I: Software Engineering Table

CoffeeTable did not support concurrent method editing. This reduced the productivity of the team when working in monolithic software structures (i.e. the main menu). If one developer wanted to edit code which was being accessed by another developer, production would stall until the first had finished working or through conversation they negotiated a resolution. While adding collaborative method editing would remove this bottleneck, it demonstrates that tools can be counter-productive if their design does not account for the culture of those who use them.

Visualisation Usage

Figure 59 renders 45 minutes worth of table usage (Group 2) as a heatmap. This shows the visualisation was used most intensively by the developer on Laptop 1. The main visualisation activity focused on the centre strip of the table. Developers primarily used this space to lay out the elements that represented the software architecture. The area directly behind the laptops was not used extensively, neither was the opposite side of the table which was difficult to reach and typically used for storage. Developers tended to reduce the size of elements which were not being used and move them out of the way. However, Group 3 did not do this as much as the other two groups and their visualisation became cluttered and (speculatively) harder to interpret and work with. This could suggest that the CoffeeTable visualisation does not readily scale to large developer working sets but could also encourage developers to maintain focused on specific areas.

The dynamic geometry of the visualisation elements (position, size, orientation, and shape) was used to add meaning to specific items or groups of items on the table. For example, items clustered around a person typically indicated that those items were ‘owned’ by that person. There are no analogues for this in development tools beyond documentation and specific organisational software. If a developer wanted to modify or access visualisation elements controlled by the other, they would have to reach into the other developer’s personal space and literally take that item of code and drop it on their laptop. Developers were observed socially

3.3 Probe I: Software Engineering Table

negotiating access to this code and using the conversation to confirm their partner developer was aware of relevant and useful information.

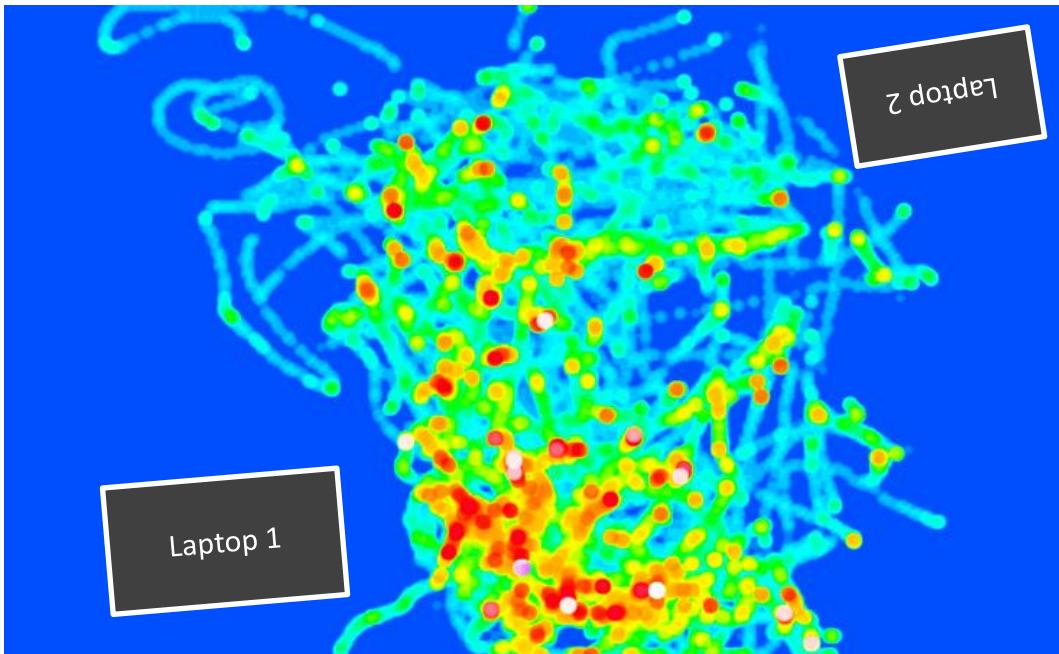


Figure 59: Heatmap showing table interaction events during Group 2's experiment. Dark blue indicates no usage, cyan, green, yellow, orange, red and white indicate increasingly more usage.

Conversation and Shared Awareness

Figure 60 visualises interaction patterns of typing, talking, and table use for Group 3 in all experimental conditions. Green blocks illustrate time spent interacting with the developer laptop (i.e. typing and mouse usage), purple blocks show time spent interacting with the table (i.e. stylus use and pointing), and red blocks illustrate time spent speaking (i.e. conversation and statements).

During pair programming and table use conversation was in-depth and relevant to the on-going work. During classic programming conversation tended to be more irrelevant (e.g. personal topics). When using the table participants often communicated in bursts with the visualisation as a subject of conversation. These bursts of conversation helped maintain shared awareness. Developers would qualify assertions and questions by pointing and interacting with visualisation elements.

3.3 Probe I: Software Engineering Table

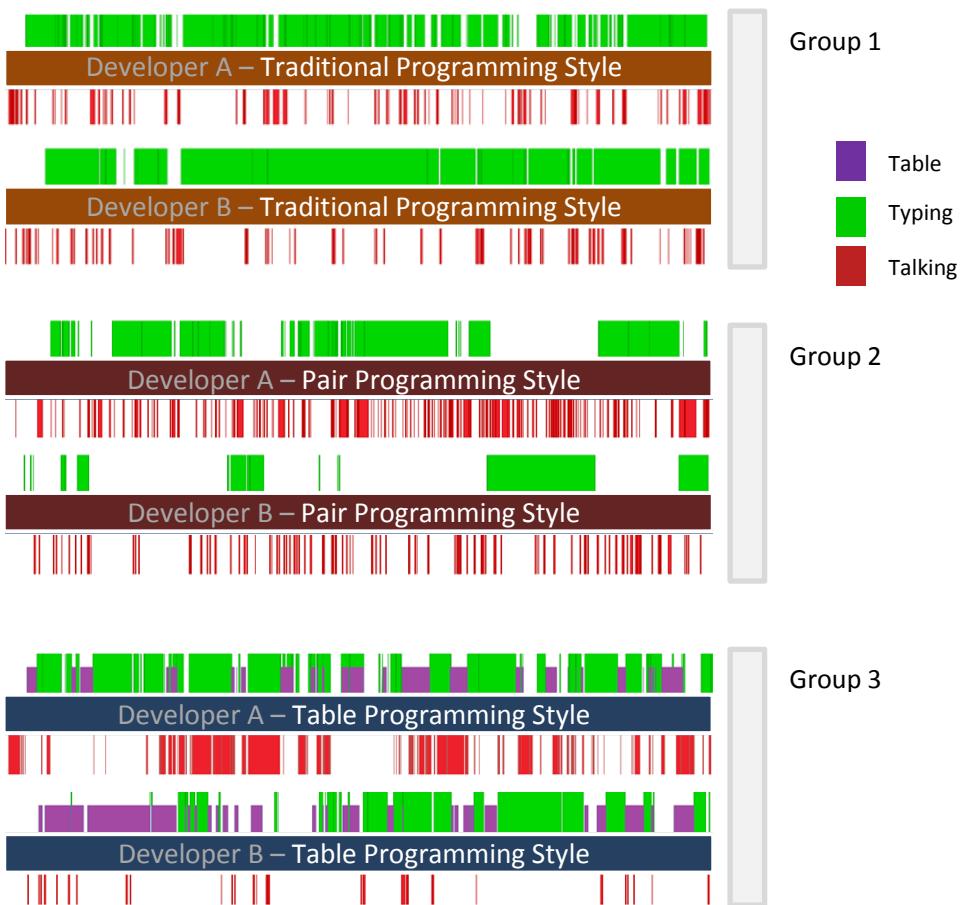


Figure 60: Visualisation of developer interaction and conversation patterns across the three experimental conditions (Group 3). Top-to-bottom: Traditional programming, pair programming, and table programming. Left to right illustrates time (45 mins).

Performing large physical actions required to interact with the table (i.e. taking an object from another developer's personal space) would stimulate conversation (i.e. development plans and the state of a given code item). This encouraged developers to socially negotiate design choices in-line which might have otherwise caused later conflicts if handled offline, or evolution of the software module without the support and understanding of other team members. Developers were able to interpret information from the way others interacted with the visualisation. For example, moving an element closer would indicate ownership. Scaling down and

3.3 Probe I: Software Engineering Table

pushing a software object away would indicate that it is not important to the current work focus.

3.3.5 Discussion

3.3.5.1 *Probe Characteristics*

Programming is an activity rich in culture and niche. Post-experiment interviews found that all developers understood how CoffeeTable addressed problems in IDE design. However, all felt that it required further development and integration into existing tools (i.e. Eclipse) before adoption would be possible. While all the participants enjoyed using CoffeeTable: “...*most [software engineering tools] are boring, and this made it fun*”, the majority were also sceptical about the benefits and suitability of an interactive projected display in longer term scenarios; after novelty effects have worn off.

Although care was taken to make this emulate the real-world pressures of time and complexity in the space of a 45 minute controlled experiment, the main limitation of the probe is the artificial setting. To improve the transfer of this technology (and the benefits it unlocks) to real-world productivity scenarios, more studies are needed with larger numbers of users over longer time periods. It would be interesting to use this development environment in an education or prototyping setting.

3.3.5.2 *Development Challenges*

The primary use case for most graphical rendering systems is a computer monitor: a rectangular pixel matrix with a vertical orientation. Applying rotations, scaling, or non-affine transformations to content typically increases the complexity of the supporting user interface code. The rendering process used in CoffeeTable was relatively simple as assuming a single flat projection plane (i.e. a single horizontal table aligned with the projector) meant that each visualisation element required only one transformation to manage its position, scale, and orientation. In scenarios with multiple projection surfaces, or surfaces which are not co-planar

3.3 Probe I: Software Engineering Table

with a projection device this assumption is not valid. Supporting visualisation elements on multiple projection surfaces would have required a much more complex renderer.

The ease with which content could be moved between the different devices (i.e. public table, private laptop) meant that exchanging content was fast and easy to accomplish. However, the underlying distributed system required network socket access and the implementation of a common protocol on all devices. If a toolkit does not allow content to communicate with external systems using existing standards it could restrict or prevent use in multi-device scenarios.

A challenging aspect of CoffeeTable's development was the creation of a large multi-user interactive surface. The decision to create a separate sensor processing toolkit (WiiTUIO) in combination with an existing multi-touch stack (to inject stylus manipulations as WM_TOUCH³⁵ messages) simplified the application software. Naturally, the WM_TOUCH API does not specify fields for all the inputs the IR stylus could provide and assumptions also had to be made for fields which the API required but the stylus did not provide (e.g. assuming a constant 'touch' pressure). Furthermore, the single-user multi-point interaction assumptions made by the .NET controls meant that additional interface controls had to be developed. In ubiquitous computing scenarios users may not necessarily think of objects, surfaces, and application content in terms of standardised interactions as they do with particular devices (i.e. touch with tablets, remotes for televisions). A generic solution would enable ubiquitous application content to directly interface with the input device without the need for underlying platform support.

3.3.5.3 *WiiTUIO Toolkit Adoption*

Following the completion of the probe, WiiTUIO was released as an open source project under a GNU GPL v3 licence: (<https://code.google.com/p/wiituiio/>). It supports both WM_TOUCH and TUIO³⁶ interaction events and has been adopted by a

³⁵ WM_TOUCH message specification:

[http://msdn.microsoft.com/en-us/library/windows/desktop/dd317341\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/dd317341(v=vs.85).aspx)

³⁶ TUIO message specification: <http://www.tuio.org/?specification>

3.3 Probe I: Software Engineering Table

number of different users and communities. As of writing it has been downloaded over 3400 times, used to create demonstrations [141], and elements of it have been integrated into a number of other 3rd party applications, most notably TouchMote³⁷.

Figure 61 shows a month-by-month site traffic graph taken from Google Analytics³⁸. Of those visiting the site—adjusting for users who immediately left the page—by their first web interaction, the majority of through-traffic (69.52%) visited the download list in comparison to (0.05%) of users who visited the source browsing / checkout page. This loosely suggests that the web audience is more focused on using the tool rather than understanding how it works.



Figure 61: Traffic statistics for the WiiTUIO toolkit since release. Graph produced 28th October 2013.

3.3.6 Summary

CoffeeTable used an interactive projected display to achieve the probe research goals described in Section 3.3.2. It contributes a fully functioning prototype of a collaborative software development environment, an open source toolkit for supporting stylus interaction on large flat surfaces, and the publication of application driven research findings [129]. Developers were observed working alone on simple tasks and transitioning to a collaborative style when necessary (Goal 1). The visualisation acted as a boundary object to help organise development, facilitate discussion, and cultivate a shared awareness of project factors (Goal 2). The shared physical space and digital content effectively combined affordances from the physical world with those of digital visualisation to assist the software development process (Goal 3).

³⁷ TouchMote: <http://touchmote.net/>

³⁸ Google Analytics: <http://www.google.com/analytics/>

3.3 Probe I: Software Engineering Table

Collaboration requires coordination, thus CoffeeTable inevitably increases the amount of information exchanged during development. CoffeeTable shows that interactive projected displays can be designed to capitalise on human-factors, and harness physical behaviours in a collaborative context to help communicate information. The collaborative workspace was a synthesis of these three types of information exchanged between developers:

- *Architecture Elements*: Interactive virtual representations of elements in the software architecture (i.e. classes, methods, fields). These are kept in a shared space and used as reference items in discussion. Interaction with these elements (i.e. project structure changes) is visible to all in the workspace.
- *Significant Actions*: Physical and virtual behaviours that are easy to notice by all in the workspace (e.g. leaning over to take some code, creating a new software class). These help to promote awareness of changes within the project which often leads to spontaneous conversation and in-line conflict resolution.
- *Workflow Indicators*: Virtual displays that help developers understand, at a glance, the different states, foci, and responsibilities within a workflow (e.g. code proximity to a developer indicating ownership). They also help to contextualise the virtual content in a physical space (i.e. a highlight line which indicates which developer is working on a particular code item).

As CoffeeTable was a fully functioning interactive projected display, it invited others to critique its design. Participants offered many design improvements and new ideas that would have otherwise remained hidden. Examples include: converting the visualisation into a call graph for debugging, conversion into a prototyping and design environment, and a ‘testing table’ where developers could pass completed modules to others who would test them in-line. Providing people with a toolkit which they can use to create, communicate, and refine ideas is supportive of the core thesis objective: *effectively supporting user innovation*.

3.4 Probe II: Interactive Office Desk

In terms of toolkit design decisions, CoffeeTable raised concerns about the long term suitability of interactive projected displays—overcoming the novelty factor and the limitations of projection (i.e. fan noise, brightness). These issues are explored extensively in the next probe. It also highlights assumptions in rendering systems, the limitations of standard desktop interaction APIs, and questions how a toolkit would support transferability to task or content-generic scenarios.

3.4 Probe II: Interactive Office Desk

3.4.1 Introduction and Goals

This section describes the second probe. It takes the form of a hybrid interactive office desk used for general computing tasks and computer science research (Figure 62). Its construction combines a standard Windows 7™ desktop environment with a monitor and top-down interactive projection onto a standard white 1.8x0.9m table. The outputs of this probe are shown in Table 7. These include the desk itself, a full conference paper (DIS'12) [52], and a magazine article (ACM Interactions'12) [142].

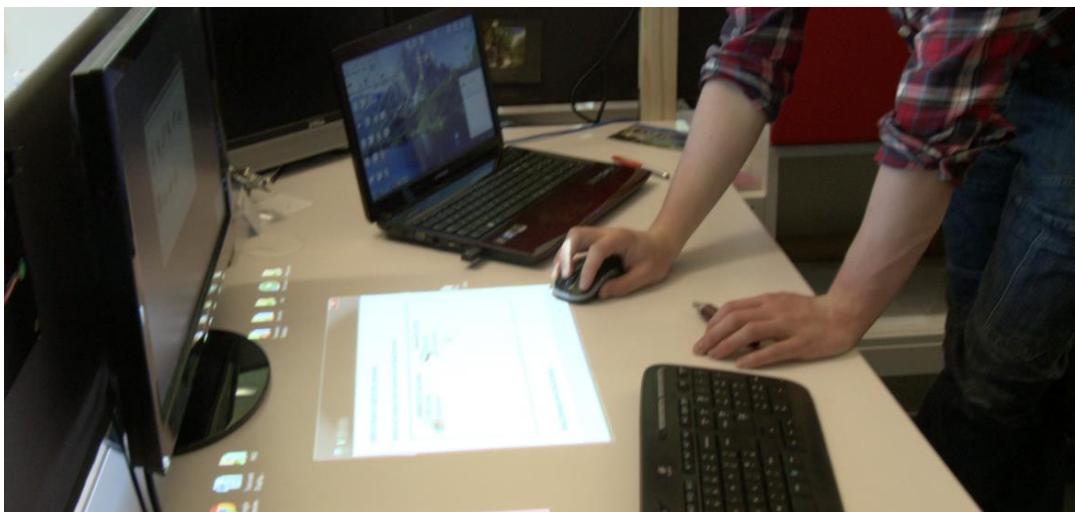


Figure 62: The hybrid interactive office desk which is the subject of Probe II.

3.4 Probe II: Interactive Office Desk

This probe builds on the previous probe by studying an applied interactive projected display for the period of one year. The study methodology is *longitudinal involved observation*. A year was selected to allow enough time for novelty effects to wear off and domestication of the technology [143] to occur. This approach is inherently subjective as experiential case studies seek verisimilitude rather than generalizable objectivity [144] [145].

Table 7: Overview of the outputs of Probe II.

	Name	Description	Picture
Display Artefacts			
1	Hybrid Interactive Office Desk (Hardware and Software)	A large top-projection interactive table with bespoke distributed IDE software. http://highwire-dtc.com/coffeetable/	
Research Papers			
2	Experiences: A Year in the Life of an Interactive Desk	Full paper at DIS'12 describing the experiences of using the desk [52]. http://dx.doi.org/10.1145/1985793.1985910	
3	Reflections: A Year Spent with an Interactive Desk	Reflective summary published in ACM Interactions (2012) [142]. http://doi.acm.org/10.1145/2377783.2377795	

This type of longitudinal qualitative approach is able to explore emerging meanings of technology, changing routines, habits, and conflicts that would not normally be accessible to quantitative methods. Involved observation is able to surface aspects that might not be obvious to external observers, such as internal rationale and muscle memory. A criticism of this descriptive approach is its reliance on detailed case studies, which make it difficult to draw the prescriptive lessons usually expected by HCI practitioners. However, a rich descriptive output is also its

3.4 Probe II: Interactive Office Desk

strength: enabling processes, artefacts, and values to be explored in more depth than individualistic quantitative methods. Further methodological details are provided in '*Experiences: A Year in the Life of an Interactive Desk*' [52].

3.4.2 Research Domain

There are many studies that enumerate, debate, and describe aspects of interactive surfaces [146] [147] [148] [149]. However, a majority of research focuses on short walk-up-and-use scenarios that evaluate specific interactions. It is rarer to find studies that consider longer term impact (i.e. +1 month) of interactive desks applied for general productivity [50] [51]. As a result, researchers and practitioners lack a contextualised understanding of interactive desks in application scenarios and their potential roles in the modern office. To address this question, this probe provides qualitative insights that can help describe the challenges associated with long term use. It also describes changes that can occur in both working process and working environment as a result of using an interactive desk. The research goals of this probe are twofold:

Goal 1) Develop, deploy, and use an interactive office desk for day-to-day computing and research tasks for the period of a year.

Goal 2) Communicate a rich account of desk usage over the year; addressing immediate design issues (i.e. ergonomics) as well emergent habits (i.e. user customisation).

These result in an evaluation of the desk by identifying factors that have the potential to limit, assist, or require development, before similar interactive projected displays can be adopted or support integrated into the toolkit design.

3.4.3 Design and Deployment

The installation itself was opportunistically constructed from a standard office desk, a desktop PC, and a surrounding wooden frame (single-device, C1). The desk had a surface area of 1.8x0.9m which was almost entirely covered by the projection (fixed geometry, C3). Output is provided using a standard 24" PC monitor and a

3.4 Probe II: Interactive Office Desk

table-top projection, while user input is provided using multiple IR styluses³⁹, a wireless keyboard, and a mouse (device based interaction, C5).

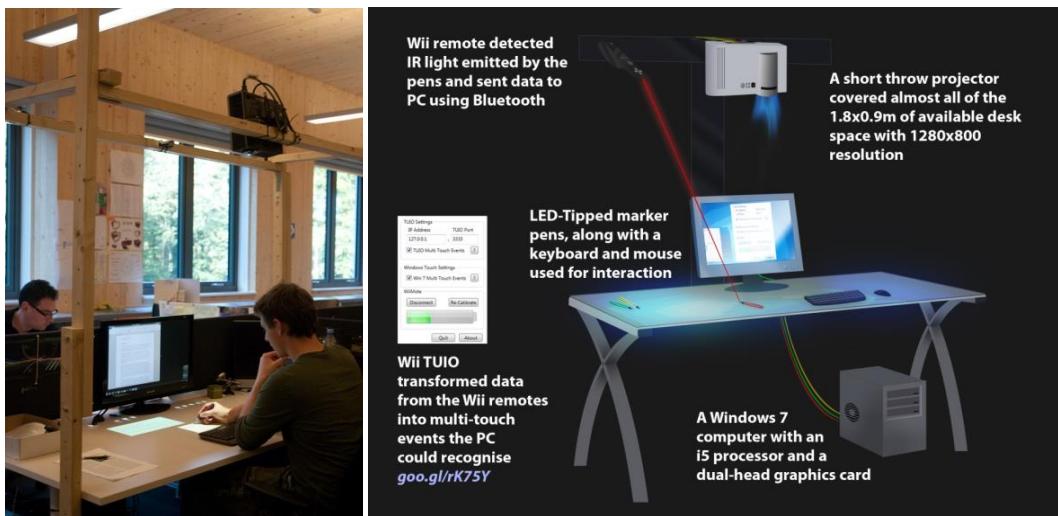


Figure 63: Left: Photograph of the interactive office desk and wooden frame. Right: Graphic which illustrates the key components of the desk. Graphic provided by Mark Ward.

A fixed-size virtual workspace was constructed by arranging the monitor and projection in a vertical stack so that the mouse and applications can move seamlessly between the two displays (framed design, C3). This is achieved using the embedded multi-monitor support provided by the operating system. The large flat nature of the desk meant that physical objects (i.e. research papers, desk toys, drinks, etc) could mix into the interaction with the virtual workspace (tangible elements, C6).

The deployment took place over a year beginning December 2010 (long term study, C11) during the author's first year as a PhD student (single developer, C12). It was used as an exclusive addition to a standard office computer. Throughout the deployment, the desk was situated in an open-plan office containing approximately 20 PhD students on Lancaster University's HighWire programme⁴⁰. The space is divided into two parts that were separated by lockers. At one end of the room is a

³⁹ Stylus support is provided by a Nintendo Wii Remote™ and calibrated to transform infrared light emitted by each stylus into Windows 7™ compatible multi-touch events. Uses WiiTUO created for the previous probe.

⁴⁰ HighWire Doctoral Training Programme: <http://highwire.lancs.ac.uk/>

3.4 Probe II: Interactive Office Desk

collaborative working space and at the other are individual desks (private use, C7), which is where the interactive desk was situated. The interactive desk was not primarily used in collaborative scenarios (non-collaborative, C9), and was used to undertake a variety of day-to-day research and computer use tasks (non-task specific, C10). These required a variety supporting applications, including: reading and writing text, internet browsing, programming, file management, listening to music, watching video, terminal operations, web development, rapid prototyping, graphic design, photo editing, and 3D modelling.

3.4.4 Analysis and Evaluation

To achieve the probe goals, this section is a reflective account of desk use over a year. The findings are reported within three categories: usability considerations—ergonomic and technical issues which impacted usage habits, user interface considerations—design limitations, behaviours, and interaction with the projected graphics, and the role of personalisation and decoration—exploring the role of customisation and integration into the physical environment.

3.4.4.1 Usability Considerations

This subsection describes factors (i.e. resolution) which influenced acceptance and usage of the desk. As the amount of time spent using the desk is extended, it becomes increasingly important that its design remains physically comfortable and efficient [150].

Readability

The limited pixel density of the projection (~20 DPI relative to the ~96 DPI of a standard monitor) was a governor for how the different surfaces (i.e. desk and monitor) were used. While it was possible to adjust to reading short bursts of text on the desk, it was uncomfortable to use for focused reading. Studies that compare reading on screens to reading on paper have suggested that at least 300 DPI is required for them to be comparable [151] [152]. Zooming in on text helped increase

3.4 Probe II: Interactive Office Desk

character legibility but did not help overall readability as the relative increase in line length required additional eye and head movement which made it difficult to maintain a flow between the words. However, this effect was not reported by Wigdor et al. [50], which suggests this may be dependent on the individual user or viewing position (i.e. seated or standing).

The differences in DPI between the monitor and the desk also meant that the size of user interface elements was inconsistent between the output surfaces. When moving content between the monitor and desk surfaces the content underwent unexpected jumps in size; making it harder to suspend disbelief that the digital content was part of the physical space.

Brightness

Prolonged exposure to bright projected light could become very uncomfortable. It was especially noticeable in the evenings under lower lighting conditions. However, turning down the brightness of the projector (~40% of 3000 lumens) and selecting a black desktop background largely alleviated the problem. However, this meant that when a window was maximised to the desk, its white background would again flood-fill the visual field with light. This was unpleasant enough to cause consciously minimisation and resizing of all windows before moving them from the monitor to the desk.

Occlusion

Occlusion is a clear drawback of top and front projection systems as objects in the projection frustum (i.e. users or coffee mugs) block out light. Although a seated user did not cause any shadowing, when standing up and leaning forward slightly, light from the projector would be blocked. To address this, placing the projector at an angle which minimises occlusion relative to the position of the user is possible; such that the shadow cast by objects (i.e. a hand) is in parallel with the users line of sight.

3.4 Probe II: Interactive Office Desk

Interaction Modalities

For most tasks, the fast and precise interactions offered by the mouse and keyboard superseded the stylus. This is in no small part due to years of user practice and the design of the GUI applications that were used on the desk. However, the stylus was also subject to more immutable physical considerations. Firstly, after long periods of use, it became tiresome to repeatedly use the big muscle groups in the back, arm, and shoulder to perform tasks over a large area that could be achieved with a mere flick of a wrist using the mouse. Secondly, there is a convenience factor or ‘momentum’ that built up using a particular tool whereby the overhead of swapping would be greater than changing to a more suitable tool (e.g. not swapping from the mouse to the stylus when drawing simple shapes as the mouse is already in the hand). This is not necessarily a conscious choice, as the mouse and keyboard were applicable to both the vertical and horizontal display planes, and adaptable enough to be suitable for the majority of applications, they were the dominant interaction method. A further awkward aspect of stylus use was that the stretch of an arm would not always be sufficient to reach important areas (i.e. minimise, maximise buttons etc.) from a comfortable seating position.

Initialisation Overhead

Starting the software to enable stylus interaction involved launching the WiiTUIO application and occasional recalibration. This process took approximately 5 minutes. Subsequently, it was easier to get straight to work and activate the stylus when necessary; leading to less serendipitous use. To counteract this, it will be important to minimise any barriers preceding interaction and ensure that there is no recurring user involvement required to start different interaction modalities.

3.4.4.2 User Interface Considerations

This subsection discusses the projected content on the desk and how this influenced usage of the desk. As collaborative computing form factors are developed, many of the assumptions made in the design of the graphical user

3.4 Probe II: Interactive Office Desk

environments are no longer valid. This subsection explores the impact of these assumptions.

Content Transformation

To improve the transfer of GUI components to the desk, the ability to apply affine transformations to certain windowed applications was developed. This enabled windows to be scaled, rotated, and translated independently of the resolution at which they are initially rendered. This attempted to address the DPI differences between output devices but also turned out to be useful from an aesthetic and layout point of view as it enabled more creative window layouts.

Shrinking windows made some applications unusable as they would be too small to interact with and text too small to read. However, it was advantageous in situations where only an overview of the content was required. For example, media applications (i.e. video where the brain can interpolate missing detail) and applications with large interface controls (i.e. play/pause, icons) retained most of their usability, even at smaller sizes.

Dual Plane Challenge

In their study of how knowledge workers make use of horizontal displays, Morris et al. [51] report an effect they called the ‘dual plane challenge’. They observed that users experience difficulty noticing windows on a horizontal display when looking at a vertical display and vice versa. In the case of this desk the dual plane challenge was particularly noticeable for modal dialog boxes which unexpectedly locked focus to another plane. However, it was rare to forget where a particular window was located given the larger desk space, but more common to forget that windows relevant to another task were open on another plane.

Multi User Support

The desk was able to accept multi-point input, but this was not useful in a multi-user context as the GUI applications did not support multiple users. For

3.4 Probe II: Interactive Office Desk

example, given two people simultaneously working on a drawing, they would be unable to assign different colours to different styluses. Thus any collaborative interaction required one person to stop interacting with multi-touch controls (e.g. drawing canvas) before another can use the single touch controls (e.g. colour selections bars). Furthermore, lack of multi-user support in the window manager⁴¹ meant that it is also only possible to interact with one application at once, even if those applications supported multi-user or multi-touch interaction.

Both of these factors restricted collaborative interaction on the desk; necessitating a negotiation for control over the interface such that all users were constantly aware of each other's interactions. As a result, this made it hard to maximise the usefulness of the desk space. Although various approaches address this have been proposed [153] [84] [104], until one is adopted by existing interface frameworks, software written for desktop computers will not transfer to multi-user computing devices.

Organisation and Layout

Large display sizes are often considered a high-value feature [149]. As a large display, the desk was typically used as a space for peripheral awareness of information (i.e. task lists), peripheral applications (i.e. music players), organisation, sub-task triage, and as a temporary store for files and notes. The monitor was typically used for focused tasks such as reading, writing, programming, and web-browsing. This task distribution is similar to what has been observed with virtual desktops or multi-monitor solutions [154] [148]. However, unlike a virtual desktop, the large desk space allowed increased use of peripheral applications, and unlike multiple monitors, the desk space offered more creative ways to arrange the contents of the focal and peripheral zones. For example, when writing a document the desk space would be used to organise and sort through content (i.e. document notes, images, and files) and then the monitor would be used to integrate content into a more complete form where there was more control over details.

⁴¹ This limitation applies to both Windows 7 and Windows 8.

3.4 Probe II: Interactive Office Desk

Spatial transitions between the different planes were used to reflect different task stages or degrees of logical separation. Features like colour, size and position provided natural ways to construct classifications or represent a shared identity across related items [155]. Similarly, the large desk space meant that items of relevance to the current task could be stored in helpful places within the visual field rather than beneath a virtual window. Popular places included alongside the keyboard, directly between the keyboard and monitor, and also off to the far left or right.

Occasionally it was also useful to force a separation between the planes by turning one off. Doing so created a minimalistic view of one or the other that helped to focus on a specific task rather than dealing with distracting aspects of a user interface or the content presented within it.

3.4.4.3 Personalisation and Decoration

People have personalised and decorated their physical environment since early cave dwellings, and office desks are no exception. By projecting into the physical environment, the desk expands the palette of decorations to include digital content. This subsection discusses how the desk was personalised and the affordances of doing so.

Epistemic Actions

Part of the value of being able to creatively arrange the working environment stems from epistemic actions: the act of modifying your environment to put yourself in a better position to think, solve a problem, or extract information from your surroundings [156]. The desk space expanded the palette of these actions through a mix of physical and digital items. The juxtaposition of the two (i.e. through layout, size, position, and grouping) could be used to create ‘fun’ and compelling interfaces. For example, projecting icons into a physical in-tray was a reminder that they should be dealt with soon, and placing the recycle-bin icon over a hole in the desk enabled files to be deleted when they were dropped into it.

3.4 Probe II: Interactive Office Desk

Self-Expression

Decoration is a way of expressing taste, creating visual appeal, storing memories, highlighting function, and expressing ownership or belonging. At the end of the year, the desk was decorated with various combinations of physical objects and virtual content, all of which served some form of expression, ownership, or function. The extra space afforded by the desk allowed my digital interests to spill out into the physical world—much like keeping a to-do list in front of a monitor.

Interactive Decorations

The desk played host to an array of physical and virtual clutter: bottles, mugs, paper, icons, sticky-notes, and digital fish (Figure 64). Although it could be argued that these are trivial issues, as computing services become more important to the lives we lead and computers themselves become more integrated into the physical spaces we inhabit, digital decorations could become as important and popular as the physical decorations that currently adorn homes, offices, and public spaces.

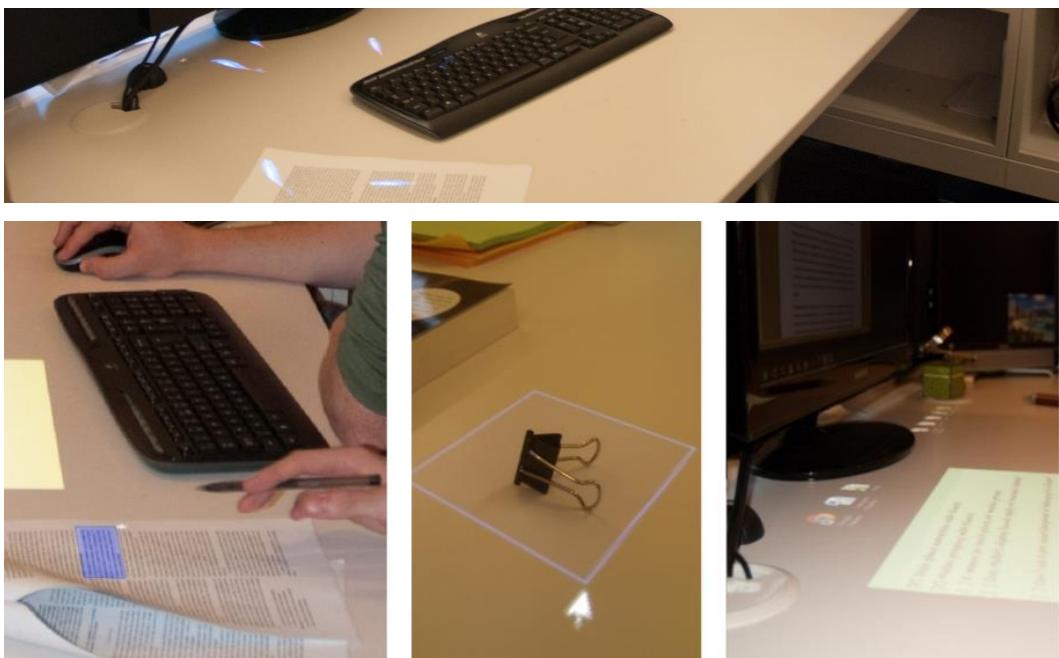


Figure 64: Digital desk decorations. Top: Digital fish. Left: Highlighting a physical research paper. Centre: Highlighting physical items to point them out. Right: arranging digital icons around physical objects.

3.4 Probe II: Interactive Office Desk

Unlike physical decorations, interaction with digital decorations could easily trigger functionality or visually update to reflect a variable state. In the physical world, this is often not possible without specially designed mechanics or electronics. However, in the digital world of the desk it was comparatively simple to bind arbitrary functionality to digital objects. For example, an icon picturing a small man placed on the far side of the desk (nearest the colleague at the adjacent desk) would email files to that colleague when they were dropped onto it. Given the extra space the desk offered, this did not take up extra prime focal space and was not often covered by other content.

Interactions sought but not implemented included an ability to assign actions to certain gestures or object arrangements. For example, dragging a finger along the top of the monitor to lock the computer or placing an object over an icon could mute my music. As it stands, implementing such features are too complex to construct and it is difficult to know ahead of time if required investment would be worth the result. Other examples included drag-dropping files onto other physical devices (similar to CoffeeTable [129] and Augmented Surfaces [9]) along with an ability to select an area of the table using the mouse and then take a photograph of the items in that area (i.e. selecting and photographing paragraphs of a paper or drawing).

3.4.5 Discussion

This section discusses how the findings of the probe could impact the design of the toolkit. Unlike the previous probe, the development of the desk was relatively simple (i.e. combining WiiTUIO with the Windows 7™ desktop environment). It discusses immediate issues that a toolkit could address and how effectively the digital desktop metaphor transfers to the physical desktop environment and implications for the toolkit design.

3.4.5.1 Immediate Issues

There is immediate scope for improvement in terms of display technologies, multi-user support, and graphical window management. From a practical

3.4 Probe II: Interactive Office Desk

perspective, the experience may not have been as positive had it not been for the ‘failsafe’ of the vertical monitor that offered high resolution display of text, etc. However, given improved display technologies this may not have been the case.

Beyond issues of readability, the projection technologies required specific hardware placement to achieve a usable projection surface. This was oriented to align with the desk and was not angle invariant. The creation and deployment of a projector mount is not something which is always possible for a number of reasons. Furthermore, mounting distance affected DPI and this had a number of influences over usability on the desk. Firstly, it restricted readability and thus what it was possible to use the different parts of the desk for. Secondly, transferring content between the devices resulted in a jump in size which required the user to resize the content to prevent it taking up too much space. Thirdly, this made it difficult to suspend disbelief that the digital content was part of the physical environment. A toolkit which is able to support different mounting points and specify the size of content items in physical size units rather than virtual pixel units would be able to support a wider range of display scenarios.

In terms of user support, the single user assumption made by the desktop environment is generally suitable in the context of a personal computer or mobile device. However, as computers integrate with physical spaces or increase in size like the desk, this assumption is no longer valid. Firstly, the desktop window manager only supported interaction with one window at once. Secondly, the interface controls are designed assuming that only one user is interacting at a given time. These are not appropriate abstractions for future interactive projected displays in physical spaces with multiple interaction modalities and multiple user contexts. For a toolkit to support wall, room, or even desk sized displays, it will be important to not assume single or multi-user interaction; allowing support for both.

In terms of window management, the graphical elements of the desk operated remarkably well in the physical environment. Not only did they function as expected digitally (i.e. easy to copy, paste, create, and delete) but their juxtaposition with physical objects (i.e. digital icons combined with physical letters in a physical in tray) added them to the tacit understanding already present in the physical space.

3.4 Probe II: Interactive Office Desk

Bringing the digital content into the physical world brings with it the assumption that it will also behave like the physical content. For instance, rotating a physical ornament or choosing a desk lamp of a certain size is an important part of aesthetic arrangement. A toolkit which supports content in a range of physical spaces could respect the behaviour of physical objects by supporting per-object and per-surface (i.e. object on shelf) translation, rotation, and scaling independent of projector DPI.

To help blend the projection into its physical surroundings, digital content could also be given the ability to understand and respond to its physical context. The desk illustrated a number of cases where this was not handled well: Firstly, opening modal dialogue boxes on the opposite display plane to the one the user was using. Secondly, the jump in size and lack of readability created when content was moved between the monitor and projection. Addressing these issues with modern windowing frameworks adds considerable complexity to the application design. Giving content an awareness of its physical properties and surrounding digital content and physical items (i.e. physical orientation, size, and projection resolution) would allow content to present itself in a manner which befits its physical setting. This is referred to later in the thesis as physical responsive design.

3.4.5.2 The Desktop Metaphor

Although the desk's technical limitations governed the how different parts of it were used for different tasks, none of these limitations were severe enough to harm productivity or prevent the completion day-to-day work. In that sense, the desk was not a productivity panacea: not noticeably better or worse than a normal computer. Indeed, productivity involves skills and creativity honed through education and experience. The desk is a tool which facilitates these processes by expanding the range of epistemic actions that could take place. Work processes adapted to the limitations of the desk and made the most of the advantages it offered. However, it took time to develop an understanding of how these could be useful. For instance, sometimes spreading out would suit the work process (i.e. collaboration, brainstorming). However, in other cases it was useful to be able to turn the monitor

3.4 Probe II: Interactive Office Desk

or the desk off entirely—downsizing the interface in order to focus and minimise distractions (i.e. writing or drawing).

An interesting point of reflection is that many positive aspects of the desk experience were dependant on an enthusiasm for customising and experimenting with the interface. Not everyone would be so inclined or in a position to do so. It remains to be seen if people who are hesitant to use technology are less likely to embrace the customisation of physical spaces like the desk.

The pervasiveness of physical decorations throughout the environments we inhabit could indicate that digital decorations represent a valuable interface metaphor for ubiquitous computing. This is suggestive of short and specific—what Nakatani and Rohrlich [157] would describe as machine-like—interactions with individual items as opposed to sustained interaction with a generic computer-like device. Lui et al. [158] argue that such interactions will be key to supporting the next generation of office workers. They demonstrate how a USB stick embedded within a glowing ball can be used as a way of playfully sharing files in an office. Heidrich et al. [112] indicate that similar short interactions are useful around domestic spaces. In another example, Wilson et al. [8] explored moving content by literally carrying projecting light and using the body as a conduit. These kinds of interaction are suggestive of a much broader design space for computer interaction; one which sees the re-physicalisation of computing metaphors to expand the set of computable interactions.

3.4.6 Summary

This probe investigated the research goals described in Section 3.4.2. To that end, a hybrid interactive office desk was constructed, deployed, and used in a day-to-day research context for the period of a year (Goal 1). The experience was captured and communicated through a rich account of desk usage (Goal 2). This presents a set of immediate technical and usability issues as well as emergent and habitual considerations.

Most of the perceived benefits of the desk stemmed its affordances as an output device rather than an input device. Factors such as brightness and resolution

3.5 Chapter Summary

are an immediate usability problem that could be addressed through the use of more advanced projector technologies. However, use of lower quality devices affected how the projection and monitor displays were used. Attempting to resolve this by increasing the size of text (on a large, close quarters display) increased the legibility of words, but not the readability of prose. The different visual planes were used for different purposes: the monitor was an area for focused tasks while the desk was an area for peripheral awareness, organisation, sub-task triage, group review, and the buffering of files and notes. The role of decoration, clutter, and personal expression should not be ignored in the design of future interactive projected displays as they integrate with physical spaces as they are important elements of a comfortable and pleasing working environment.

There is immediate scope for the improvement of window management and the inclusion of affine-transforms in table-top user interfaces such as making content able to react to its surroundings to respect DPI independent sizing. The lack of a multi-user assumption in modern multi-touch frameworks and windowing toolkits prevents more than one application from being used simultaneously, regardless of multi-touch or multi-user design.

In comparison to the previous probe, this probe was easier to construct as it adapted existing tools and established technologies to create an interface that was viable for long term use. This form of bricolage is common in the hacker and maker communities. Supporting a number of different hardware configurations (as discussed in Section 3.4.5.1) is important. The subjective and descriptive methodology used in the analysis is difficult to objectively generalise to other scenarios. However, the subjective and descriptive nature are also its strength as they allow consideration of personal factors such as decoration and aesthetic which are difficult to reason about through purely objective and quantitative terms.

3.5 Chapter Summary

This chapter informs the toolkit design by improving the understanding of how interactive projected displays can be built and applied to application scenarios in order to generate value. It studies a range of interface, application, and project characteristics through two in-depth research probes. These probes are conducted iteratively such that the second answered questions raised by the first regarding the suitability of the technology to generic use in a long term context. The core contributions of this chapter are:

1. *Experience building and deploying interactive projected displays.* The research probes generate knowledge about the technical and practical issues involved in system construction. They also draw out how effective existing solutions are at supporting interactive projected displays in applied scenarios. The development of these probes also resulted in the construction and deployment of the open source WiTUO toolkit (Section 3.3.5.3).
2. *A deeper understanding of important display characteristics in real world application domains.* Given the characteristics described in Section 3.2.2, Probe I demonstrates how a multi-device design can be used to create an effective working environment by combining shared visualisation with private development spaces. It also shows how frameless design and dynamic geometry can be used to effortlessly capture features of a working process in a way that helps improve shared understanding in collaborative scenarios. Probe II describes how digital projections and physical objects can be combined to decorate a personal space. Both probes indicate that collaboration and coordination in physical spaces leads to concurrent multi-user interaction requirements that current operating systems do not support.
3. *Research findings in targeted application domains enabled by the introduction of interactive projected displays.* Together the probes resulted in the creation of two interactive projected displays. The application of these displays led to a total of two conference papers [129] [52] and one magazine article [142] in the

3.5 Chapter Summary

domains of software engineering (ICSE'2011 New Ideas and Emerging Results Track) and the organisation and productivity domain (DIS'2012 Organisation and Productivity Session).

In a typical design process the initial conceptual stage has no distinct conclusion. Its role is to explore a solution space (i.e. requirements scope). In terms of this thesis, the work in this chapter enables deeper reasoning and justification of the design decisions in the next chapter. The role of next chapter is to concentrate these findings on a single toolkit design.

Chapter 4. Toolkit Requirements

4.1 Overview

The last chapter presented two in-depth application driven research probes in order to inform the toolkit design. The purpose of this chapter is to present a set of toolkit requirements based on those findings. All the requirements are intended to support the objective of *creating a software toolkit that supports user innovation with interactive projected displays*.

This requirements chapter draws on three main sources of information: the background work described in Chapter 2, the lessons and findings of the research probes in Chapter 3, and the descriptions of stakeholders in Section 4.2. To ensure that the requirements are sensitive to the innovation and adoption processes that motivate the thesis (Section 1.2), von Hippel's *five criteria for toolkits in user innovation* [32] are used as a framework to structure the requirements. As requirements address each of these criteria, it is possible to be more confident that the toolkit design will support, encourage, and stimulate user innovation with interactive projected displays.

Figure 65 outlines the structure for this chapter. It begins by describing characteristics of key stakeholders (Section 4.2) and the general constraints placed on the toolkit design (Section 4.3). The next section presents the requirements themselves (Section 4.4). The requirements are organised into groups (based on their ability to meet von Hippel's five criteria for toolkit innovation [32]) and ordered within these groups based on the number of relevant stakeholders. The last section (Section 4.5) summarises the chosen requirements which are implemented in the next chapter.

4.2 Stakeholders

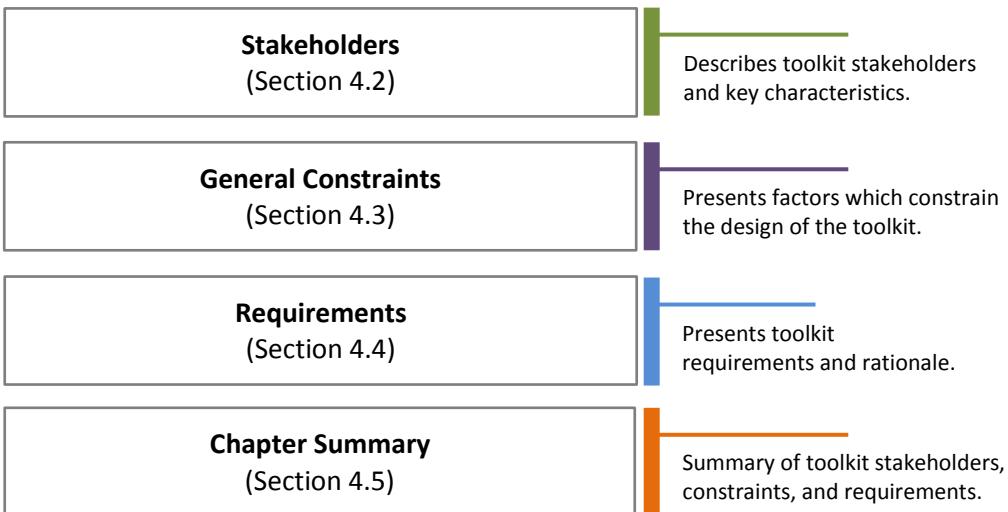


Figure 65: Structure of the toolkit requirements chapter.

4.2 Stakeholders

This subsection identifies the toolkit stakeholders—describing their role and interest in the toolkit or the displays it can create. Stakeholders may interact with the toolkit directly (i.e. as a user or developer) or indirectly (i.e. project sponsor or space owner). Particular focus is given to the demographic of would-be *toolkit users*, as these are the ‘innovators and early adopters’ [29] who will download, use, and deploy displays. A brief analysis of their demographic helps to contextualise their culture and skillset.

4.2.1 Stakeholder Roles

Table 8 identifies the major toolkit stakeholders and characteristics. The purpose is to identify clear boundaries between the roles by defining responsibilities and goals. In practice, roles may overlap and be the same person.

Table 8: Toolkit stakeholders: relationship to toolkit, role, and stake.

Stakeholder Name	Interaction ⁴²	Role	Stake and Interests
User	Direct	Directly interact with displays created using the toolkit (i.e. touching or gesturing). May ignore displays entirely [114]. Users may complain to display or space owner if not satisfied. They have different age ranges, physical abilities, and attitudes to technology.	Specific display installations may target different users. Users prefer systems with a superior user experiences (both in terms of form and function [57]).
Toolkit User	Direct	Develop and deploy interactive projected display installations and content. May be a professional or academic researcher. May be a member of the maker, hobbyist, or DIY technology community. May have established processes and technologies. Basic technical and deployment skills required.	A simple and easy to use toolkit with as few barriers to usage as possible. Different toolkit users may require different types of display. To use the toolkit the needs of these users must be met. May be an organisation interested in profiting from the technology. May be a hobbyist or maker interested in 'playing' with the technology.
Content Creator	Indirect	Produces content for interactive projected displays. Similar to the toolkit user, although content creators may not have knowledge of the circumstances under which their content is deployed. May not necessarily be aware of toolkit's existence.	Easy content development process. Control over how their content appears in different configurations and the interaction modalities that are used. May be representing a 3 rd party or publishing content not necessarily designed for specific deployments.
Toolkit Developer	Direct	Develops extensions and / or functionality for the toolkit. The author of this thesis is included in this category.	A flexible and clean codebase that is easy to extend, develop, and maintain. May be developing specific enhancements for the technology for personal reasons and sharing with a user community.

⁴² A direct stakeholder has direct contact with the system whereas as indirect stakeholder does not.

4.2 Stakeholders

Space Owner	Indirect	Responsible for managing the space that contains an interactive projected display.	Interested in catering to the needs of those in the space, managing the aesthetic of a space, ensuring that content and interaction with the display is appropriate.
Display Owner	Indirect	Responsible for the projected display in a space. May own the hardware and software used in the deployment.	Interested in multiple ways of deploying and configuring the display as easily as possible. Also interested in toolkit reliability.
Community Member	Indirect	Engages with a community of toolkit users, asking questions and exchanging ideas. Can become a toolkit user or content creator.	A low learning curve to engaging with the toolkit. A friendly and welcoming user community.
Project Sponsor	Indirect	Is motivated to solve a problem or explore an application scenario through the introduction of an interactive projected display. Rather than engaging with the toolkit themselves, they sponsor others to do it.	Not necessarily interested in the inner workings of the toolkit or even an awareness that it exists. Interested in technologies that solve their problems and can be introduced into application scenarios with speed and minimal problems.

Table 8 firstly distinguishes between a *user* and a *toolkit user*. *Users* are individuals who interact with the displays created by the toolkit; perhaps by passing through a space which contains a display or interacting with content. *Toolkit users* are people who literally use the toolkit to create displays and deploy content. *Content creators* are defined as those who develop digital content which can be displayed, although they may not necessarily be aware of the deployment conditions where their content will be displayed (i.e. videos, copy, etc.) *Toolkit developers* are individuals who extend or change the core functionality of the toolkit (i.e. porting to a new platform). The author of thesis is included in this category as a special case.

4.2.2 Influence and Demographic

The roles and interests described in Table 8 are intentionally coarse as stakeholders may come from many different backgrounds and have many different reasons for engaging with the toolkit. Figure 66 subjectively plots the stakeholders according to their influence and interest in the toolkit; helping to reason about the extent to which they should be considered in the requirements specification.

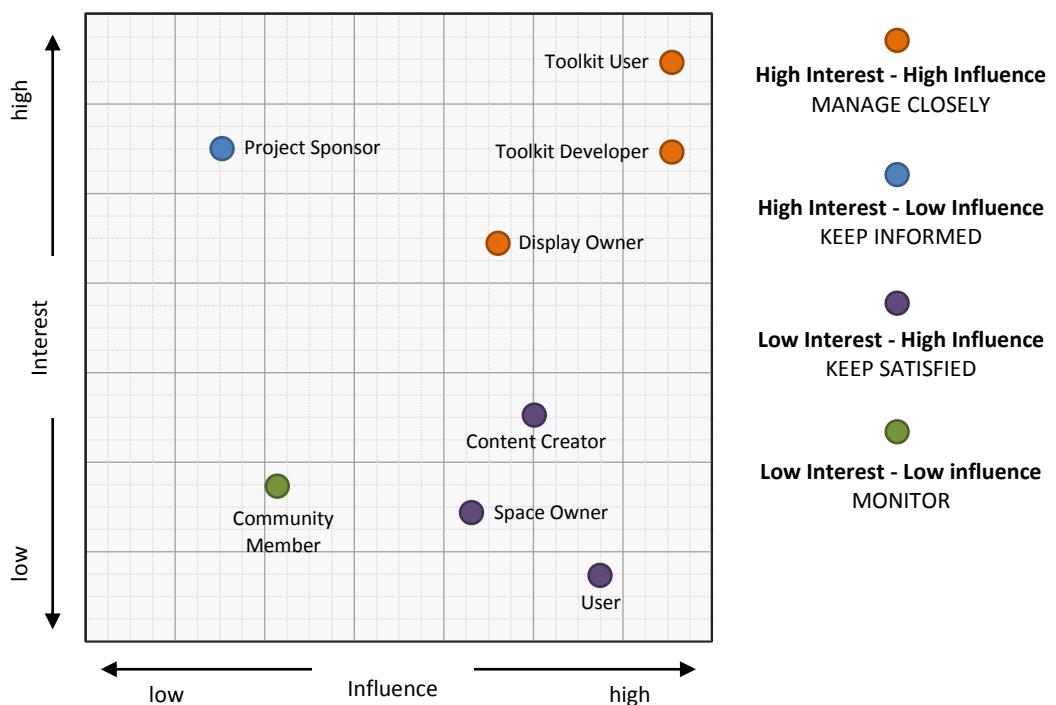


Figure 66: Interest-influence diagram of toolkit stakeholders. Placements are illustrative estimates.

4.2 Stakeholders

The primary toolkit stakeholder is the *toolkit user* as they have the highest interest and influence of all the stakeholders and are the target for the consumption and use of the toolkit. The *toolkit users* fit within the innovators and early adopter's category on Rogers' diffusion of innovations theory [29] (Figure 2, Section 1.2). These people are willing to experiment, take risks, and be early consumers of new technology. They typically provide considerable and candid feedback which can be used to help refine the technology. For these people to become toolkit users, they must be able to have access to relevant consumer sensing and projection hardware, an awareness of the technology, and a willingness to engage and experiment with technology.

Outside of potential academic and cooperate toolkit users, this characterisation resonates with the maker, hacker, and DIY computing enthusiast communities. Analysis of these communities through surveys can be difficult as respondents are usually self-selecting. Attendees of the 'Maker Faire Bay Area'⁴³ event in 2012 [159] were primarily male (66%) with a median age of 46.5 years. In terms of education, virtually all (98%) attended or graduated university and (43%) hold postgraduate degrees. In terms of background, the most popular self-descriptions are: hobbyist (58%), tinkerer (39%), engineer (31%), programmer (31%), and beginner (30%). In a large-scale survey of over 2600 individuals involved in DIY communities, cultures, and projects, Kuznetsov and Paulos [160] found that values such as open sharing, learning, and creativity are often placed higher than other common motivators like profit. They hypothesize that unlike communities which revolve around artefacts (i.e. software repositories and scientific articles) DIY communities revolve around meta-information, such as personal experiences and knowledge gained from creating physical objects, that is then projected into the public sphere.

⁴³ Research Report [159] (<http://cdn.makezine.com/make/sales/maker-faire-bay-area-survey-09-2012.pdf>) summarises 2740 complete survey responses. Faire took place between the 19th May 2012 and 20th May 2012 at the San Mateo Event Centre, San Mateo, CA.

4.3 General Constraints

4.3 General Constraints

This section describes general constraints (and assumptions) that are built into the toolkit design. These constraints are immutable factors that influence the development of the toolkit and should be recognised ahead of time. The constraints cover: toolkit distribution (i.e. software), available resources, time frame, and performance. They are derived from ‘common sense’ project issues and the need to cater to the toolkit user demographic identified in Section 4.2.

Constraint 1: Online Distribution

Rationale	The speed and accessibility of online resources make them an effective method of software distribution for DIY communities and academic researchers alike. To encourage adoption, the distribution mechanism shall not present a barrier for engagement.
Implications	The toolkit hardware will be sourced by those who wish to use it. The toolkit will only be available as software. To cater to toolkit user expectations and established distribution mechanisms it will be important to create an online presence, support network, and promote of the toolkit. For instance, an online open source project.

Constraint 2: Use only commodity, readily available, or inexpensive hardware

Rationale	To prevent the toolkit requirements from exceeding the reach of would-be toolkit users, the hardware required should be easy to find and cheap to purchase. Kuznetsov and Paulos [160] found that the typical spend on a hobbyist project was \$11-\$50USD. However, they report a correlation between project cost and project completion time. If projects are able to reuse existing hardware, they can also be opportunistic and do not incur delivery times for components. If possible it could even be recycled from previous projects.
Implications	The toolkit cannot rely on expensive or specialist hardware. The hardware must use off-the-shelf components such as a PC, office projector, and sensors like the Microsoft Kinect™ or Nintendo Wii™. High performance projection equipment can be supported, but not required. This may restrict the size of the displays that can be created to the range and capabilities of the hardware (i.e. display resolution). Further, the toolkit shall be designed to achieve its goals in a way which maximises the resources available, such as processing power.

4.4 Requirements Specification

Constraint 3: Support uninstrumented surfaces

Rationale	If users are required to instrument each projection surface the range of potential interface locations becomes limited to: (a) the number of instruments which are available, (b) the locations where the user has permission to instrument, and (c) locations where instrumentation does not interfere with the aesthetic of the space.
Implications	The toolkit must not rely on surface instrumentation, and thus must use optical sensing methods (as described in Section 2.4) to detect interaction.

Constraint 4: Allow time for toolkit adoption and evaluation

Rationale	To evaluate toolkit adoption and usage patterns, time must be allowed for the adoption process to take place. This must also be done within the timeframe of the doctoral programme.
Implications	Work must be carried out which promotes the toolkit within the DIY user communities. The toolkit should be released with at least 6 months (minimum) to allow for adoption to take place ⁴⁴ . A backup strategy if this should fail should be in place.

4.4 Requirements Specification

This section specifies the behaviour and properties of the toolkit through requirements organised around von Hippel's criteria for toolkits in user innovation. This helps to ensure the requirements provide the necessary support to toolkit users.

Table 9 summarises all the requirements. Each requirement features a numeric identifier, name, description, type⁴⁵ and rationale. Traceability is provided by cross-referencing each requirement with relevant literature and probe findings to justify

⁴⁴ Assuming a similar adoption pattern to WiiTUIO (Probe I, Section 3.3.5.3) 6 months of time would be enough to engage approximately 500 users.

⁴⁵ Functional requirements define specific behaviours (i.e. what the toolkit is *supposed to do*), whereas non-functional requirements specify criteria that can be used to judge the operation of a system, rather than specific behaviours (i.e. how a system is *supposed to be*).

4.4 Requirements Specification

its inclusion. The extent to which the toolkit meets each requirement is covered later in Chapter 6.

According to von Hippel [32], toolkits deliberately facilitate scenarios that the toolkit creator does not consider. As such, the requirements support a process of display creation, rather than a specific set of applications or use-case scenarios. The range of displays that can be created by this process falls within the approximate design space laid out by the visions of interactive projected displays (Section 2.2) and the characteristics explored by the research probes (Section 3.2.3). In '*User Toolkits for Innovation*' von Hippel [32] observed that toolkits which effectively support user innovation meet five key criteria. These have been shown to be applicable in a variety of software related scenarios [161] and consider different aspects of the toolkit lifecycle. The five criteria are summarised below:

- *User Friendly Operation*: This addresses the interactions toolkit users have with the toolkit. Users need to be able to operate the tools using customary languages and skills without much additional training.
- *Trial and Error Learning*: This asserts that toolkit users should be able to learn through experimentation. Allowing toolkit users to quickly see the consequences of design decisions helps to precisely identify what they want.
- *Appropriate Solution Space*: This encompasses support for the range of displays that toolkit users might create. Limiting factors define the solution space users are able to address. Supporting a larger solution space expands the set (and thus chance) of innovation.
- *Common Modules*: This asserts that tools should be provided with libraries of commonly used modules which a toolkit user can incorporate into their designs. This allows the toolkit user to focus on the unique parts that are the focus of their design.
- *Easy Transfer to Production*: This asserts that the outputs of the toolkit should be easy to convert into the format required for a production (or in this case, real world deployment). If toolkit users are not able to deploy their designs, much of the effect of a toolkit is lost.

4.4.1 Requirements Summary

Table 9: Summary of toolkit requirements.

Requirement Details											
#	Name	Description	Type	Toolkit User	User	Content	Space	Literature	Probe I	Probe II	Demographic
<i>User Friendly Operation</i>											
1	Lower Skill Barriers	Lower the skill barriers required for toolkit users to create interactive projected displays.	NF	Y		Y		Y			
2	Easy Toolkit Interface Operation	The toolkit interface will be simple for toolkit users to operate.	NF	Y				Y			
3	Simple Abstractions	Toolkit abstractions shall be simple for toolkit users to understand and work with.	NF	Y				Y			
<i>Trial and Error Learning</i>											
5	Graceful Error Handling and Degradation	The toolkit shall be tolerant to errors with graceful degradation and debugging support.	NF	Y	Y	Y	Y	Y			
6	Fast and Simple Reconfiguration	The toolkit shall be easy to reconfigure to suit different scenarios.	NF	Y				Y			
7	Minimise Calibration	The toolkit shall minimise the overhead and process of calibration.	NF	Y	Y					Y	
<i>Appropriate Toolkit Solution Space</i>											
8	Variable Display Sizes	Support a range of interactive projected display sizes	NF	Y		Y	Y	Y	Y		
9	Programmable Aesthetics	The toolkit shall enable displays to appear, disappear, and move around their environment programmatically.	F	Y	Y			Y			
10	Projection Mapping	The toolkit shall support the application of graphical transformations to interface content.	F	Y		Y			Y		

4.4 Requirements Specification

11	Multiple Displays	Possible to create multiple displays simultaneously using the same hardware.	F		
12	Physical Responsive Design	The toolkit shall enable content to adjust its graphics in response to changes in its physical environment.	F		
13	Responsive Interaction Design	Content on the displays can select interaction modalities programmatically.	F		
14	Programmable Content	The toolkit shall support programmable display content.	F		
15	Multiple Concurrent Users	Multiple users shall be able to interact with multiple items of display content simultaneously without coordination.	F		
16	Walk Up and Use	No specialist interaction training or tools shall be required to interact with display content. Interaction should be accurate.	F		
4	Inter-display Communication	Provide a convenient mechanism for items of display content to communicate with one and other.	NF		
	<i>Common Modules</i>				
17	Standards Support	The toolkit shall support use of existing content standards and rich media.	F		
18	Decoupled from Platform	Interaction modalities should be decoupled from the underlying operating platform.	NF		
19	Documentation and Samples	The toolkit shall be released with documentation and samples which demonstrate common use cases.	NF		
	<i>Easy Transfer to Production / Deployment</i>				
20	Robust Hardware Placements	Support a range of different projector and sensor placements.	NF		
21	Public Deployments	The toolkit must be suitable for deployment public spaces.	NF		
22	Interoperable and Extensible	Toolkit shall support interoperation with external systems and new interaction technologies.	NF		

4.4.2 User Friendly Operation

The following requirements (*Requirements 1 – 3*) make it easier to construct interactive projected displays. The term ‘user friendly’ refers to the *toolkit users* described in Table 8. The requirements cover lowering skill barriers, easy interface operation, and simple toolkit user abstractions.

Requirement 1: Lower Skill Barriers

Description	Lower the skill barriers required for toolkit users to create interactive projected displays.
Rationale	Lowering skill requirements mean more people will be able to engage within interactive projected displays as toolkit users. Supporting a range of skills (i.e. programmers and non-programmers) is important. However, the capabilities of the toolkit should scale with the capabilities of the user. This means more advanced users to drill down into more complex functionality (i.e. creating content) whilst novice users can deploy existing content.

Requirement 2: Easy Toolkit Interface Operation

Description	The toolkit interface will be simple for toolkit users to operate.
Rationale	An effective toolkit interface will allow toolkit users to focus on their application development rather than the underlying projection and sensing technology. Existing toolkits are button dense and contain many domain specific references (Section 2.6).

Requirement 3: Simple Abstractions

Description	Toolkit abstractions shall be simple for toolkit users to understand and work with.
Rationale	Intuitive and simple abstractions (i.e. those which relate to known physical concepts, such as files) make it easier for toolkit users to express their goals to the toolkit. It will also simplify the training process for toolkit users.

4.4.3 Trial and Error Learning

The following requirements (*Requirements 4 – 6*) support trial and error learning with the toolkit. Trial and error learning is important because it encourages

4.4 Requirements Specification

experimentation through graceful error handling, robust deployments, and rapid reconfiguration.

Requirement 4: Graceful Error Handling and Degradation

Description	The toolkit shall be tolerant to content errors and varying performance demands with graceful degradation and debugging support.
Rationale	Graceful error handling is important to all the stakeholders. Toolkit users will be able to debug content more easily if they do not need to re-launch the toolkit every time it encounters an error. Users and space owners have a better user experience if degradation is graceful (i.e. performance of one display does not impact the other).

Requirement 5: Rapid Configuration

Description	The toolkit shall simplify and expedite the creation and reconfiguration of displays to suit different scenarios.
Rationale	Reconfiguring a display deployment to suit new physical circumstances (i.e. display size, etc) should be a short process; taking minutes not hours. Toolkit users are more likely to play with the toolkit and try out new ideas if they can be created and deployed within a short amount of time. In Probe II, the ease of layout changes meant that the desk could be quickly reconfigured to suit productivity needs. The focus was on the application rather than operating the underlying projection technology.

Requirement 6: Minimise Calibration

Description	The toolkit shall minimise the overhead and process of calibration.
Rationale	Probe II described a need for no recurring user involvement required to start using different interaction modalities. The same is true for toolkit users; calibration processes that take too long or are too complex (i.e. those requiring a special physical device or marker etc.) increase application development time, complicate deployment, and should be avoided.

4.4.4 Appropriate Solution Space

The following ten requirements (*Requirements 7 – 16*) outline the range of display characteristics that the toolkit can support. These include: multiple display sizes, programmable aesthetics, projection mapped graphics, multi-display

4.4 Requirements Specification

scenarios, content and interaction which are responsive to physical contexts, multiple concurrent users, and walk-up-and-use scenario support. These draw on the existing displays in the literature (i.e. the features of the visionary systems in Section 2.2) and the research probes conducted in Chapter 3.

Requirement 7: Variable Display Sizes

Description	Support a range of interactive projected display sizes.
Rationale	Both the probes and the literature demonstrate a range of different display sizes. To be applicable in the object, furniture, and wall and room sized displays (shown in Figure 6, Section 2.1.1) the toolkit must cater to these situations.

Requirement 8: Programmable Aesthetics

Description	The toolkit shall enable displays to appear, disappear, and move around their environment programmatically.
Rationale	Probe II highlighted the importance and opportunities in appropriate projection aesthetics (Section 3.4.4.3) as well as a need to ensure that a space is not overloaded with content (Section 3.4.4.2). Enabling content to appear, disappear, and move around a space programmatically (i.e. responsive to user location) caters to a wide range of design possibilities.

Requirement 9: Projection Mapping

Description	The toolkit shall support the application of graphical transformations to interface content.
Rationale	Probe I's use of dynamic geometry highlighted a limitation in traditional screen based content renderers: they are suited to a single output surfaces (i.e. screens) rather than multiple surfaces at different angles. To help simplify the content development process, it should be possible to map projected content to individual surfaces. However, this can increase the complexity of simple content development if each interface element (i.e. button) has to be given its own location in the physical space.

Requirement 10: Multiple Displays

Description	Possible to create multiple displays simultaneously using the same hardware.
-------------	---

4.4 Requirements Specification

Rationale	Multi display interactive projections are relatively common design choices (Section 2.2 and 2.3.3.4). To maximise the potential of the hardware, the toolkit should make it possible for toolkit users to simultaneously deploy multiple items of display content on multiple surfaces.
-----------	---

Requirement 11: Physical Responsive Design

Description	The toolkit shall enable content to adjust its graphics in response to changes in its physical environment.
Rationale	Interactive projected displays come in a range of sizes (Requirement 7) and on a range of different surfaces (Requirement 10). Effective content design for different locations requires an appreciation of the display surface properties (i.e. physical width, height, and orientation – see Section 2.5.1). Enabling content designers to access these properties allows content to be presented in a manner which is appropriate for the space (i.e. working in physical dimensions). In sections 3.3.5.2 and 3.4.4.2 content was not able to take account of the low DPI in its presentation. Unintelligent juxtaposition of digital and physical ‘items’ can negatively impact user experience (Probe II, Section 3.4.4.1).

Requirement 12: Responsive Interaction Design

Description	Content on the displays can select interaction modalities programmatically.
Rationale	Unlike traditional displays, projected displays can appear in a variety of physical settings and scenarios (Section 2.5). Allowing content to select an interaction modality that suits the circumstances (i.e. touch if close, gesture if far away, presence if a simple content trigger) means that toolkit users and content developers can design appropriately, independent of physical setting.

Requirement 13: Programmable Content

Description	The toolkit shall support programmable display content.
Rationale	Programmable content enables a broad range of applications to be created. It also allows content to be used to create task specific interfaces (Probe I) and be adapted to suit scenarios which the toolkit developers have not considered. This is particularly important in the exploration of new ubiquitous computing scenarios.

4.4 Requirements Specification

Requirement 14: Multiple Concurrent Users

Description	Multiple users shall be able to interact with multiple items of display content simultaneously without coordination.
Rationale	Probe I demonstrated how certain levels of coordination between users can be useful in certain task-specific design contexts (Section 3.3.4.2). However, Probe II highlighted that if the application scenario does not call for it, then the extra coordination can be harmful (Section 3.4.4.2). Subsequently, the toolkit design should enable the toolkit users and content creators to have the control to dictate this level of coordination.

Requirement 15: Walk Up and Use

Description	No specialist interaction training or tools shall be required to interact with display content. Interaction should be accurate.
Rationale	Walk-up-and-use scenarios which do not require training are an important aspect of public usability. Probe II illustrated that even in a private scenario, overheads of interacting with different tools can lead to reduced use of specialist equipment (Section 3.4.4.1). Content that supports walk-up-and-use interaction modalities (i.e. touch, Section 2.4.1) is applicable to a wider range of usage scenarios than content which does not.

Requirement 16: Inter-display Communication

Description	Provide a convenient mechanism for items of display content to communicate with one and other.
Rationale	The connectivity between the different displays in Probe I were a major development challenge. However, both probes demonstrate the value many graphical items. To simplify the development of complex display applications, the toolkit should provide convenient (i.e. easy to use) programmable mechanisms which enable items of display content to exchange data. This will reduce the effort and knowledge toolkit users require (i.e. RMI) to develop multi-display applications. <i>Depends on Requirement 10 for multi-display support.</i>

4.4.5 Common Modules

The following requirements (*Requirements 17 – 19*) support ‘common modules’—referring to standards and re-usable elements (i.e. touch interaction

4.4 Requirements Specification

modality) that can be shared between toolkit users and allow more focus on application content rather than the implementation details.

Requirement 17: Standards Support

Description	The toolkit shall support use of existing content standards and rich media.
Rationale	Rich media (i.e. sound, video, animation, and graphics) is an important aspect of digital signage as it helps attract user attention and communicate content clearly to users [114]. Probe II reflects on the potential importance of digital decorations (Section 3.4.4.3). Supporting existing content standards decreases training times and the toolkit learning curve, addressing limitations of existing systems (i.e. Section 2.5.2). People will be able to use existing work in new ways. Furthermore, non-programmers who wish to create interactive projected displays with existing rich media should not be excluded as potential toolkit users.

Requirement 18: Decoupled from Platform

Description	The content and interaction modalities should be decoupled from the underlying applications and operating platform.
Rationale	Decoupling content from the underlying applications (i.e. not compiled) and platform (i.e. not relying on OS mouse and touch events) makes it easier to swap content in and out, and offer consistent interaction behaviours regardless of platform capabilities. For instance, platforms make assumptions about from-factors (i.e. a screen) which are not always suitable for interactive projected displays. As shown in Probe I, these can needlessly limit the expressivity of an interaction modality (Section 3.3.5.2). A better solution allows content to interface directly with the interaction modality in a way which lets it customise the input to suit both the content design and the physical circumstances.

Requirement 19: Documentation and Samples

Description	The toolkit shall be released with documentation and samples which demonstrate common use cases.
Rationale	Toolkit users must be shown how to achieve basic interfaces with the toolkit (i.e. touch interface). To do this, a library of samples and common interaction techniques (i.e. touch and presence detection) should be provided with the toolkit.

4.4 Requirements Specification

4.4.6 Easy Deployment

The last three requirements (*Requirements 20 – 22*) contribute to the transfer between development and deployment. This is supported through robust hardware placement, support for public deployment, and interoperability with external systems.

Requirement 20: Robust Hardware Placements

Description	Support a range of different projector and sensor placements.
Rationale	Deploying technology into a space can be challenging due to restrictions such as hard-to-reach power points and aesthetic concerns [114]. Supporting a range of projector and sensor placements means that toolkit users will be able to (a) work around the needs of the space and display owners, and (b) maximise the utility of the sensors and projectors to suit the types of the display they are creating (i.e. long throw projector, high resolution camera view of a particular surface).

Requirement 21: Public Deployments

Description	The toolkit must be suitable for deployment in a public space.
Rationale	Although Probe I and Probe II examine semi-private and private scenarios, many of the displays described in the literature are suitable for use in public spaces (Section 2.2 and 2.4).

Requirement 22: Interoperable and Extensible

Description	The toolkit shall support interoperation with external systems and new interaction technologies.
Rationale	Toolkits in user innovation transfer need-related aspects of development to toolkit users, in this case, exposing projected displays to a wider range of scenarios [15] [32]. To support use in additional scenarios (i.e. desired but unrealised features of the desk in Section 3.4.4.3), the toolkit must be able to integrate with the external devices and services (i.e. home automation, web services, etc.)

4.5 Chapter Summary

This chapter outlined the major stakeholders, constraints, and requirements for the toolkit implementation in the next chapter. The requirements and constraints draw on the innovation literature [32] [30] [29], the characteristics of interactive projected displays in the pervasive and ubiquitous computing domains [16] [18] [17] (Chapter 2), and the experience and knowledge generated through the applied research probes (Chapter 3). To design for adoption the requirements are structured according to von Hippel's toolkits for user innovation criteria [32]. In total four general constraints are presented (Section 4.3). These are summarised below:

- **C1:** Online Distribution.
- **C2:** Use only commodity, readily available, or inexpensive hardware.
- **C3:** Support uninstrumented surfaces.
- **C4:** Allow time for toolkit adoption and user evaluation.

A total of 22 requirements are presented. Of these, three refer to *user friendly design*, three refer to *trial and error learning*, ten refer to the *toolkit solution space*, and three refer to support for *common modules* and *easy deployment*. Each requirement is presented along with a description and rationale for its inclusion. These requirements are summarised in Table 9 (page 126). A domain model diagram is provided in Figure 67 to visualise the key vocabulary, concepts, and relationships within the toolkit design domain. It illustrates three main types of entity: *stakeholders* (green), *software* (purple), and *physical hardware and space* (orange). Without a toolkit, the display content and common modules would simply be a single 'application' with a tight coupling to the hardware platform and the physical environment. By introducing a toolkit, the content is decoupled from hardware platform. The common modules (i.e. interaction modalities, input devices, network connectivity) are further decoupled from display content such that content creators and toolkit users can focus on developing applications which suit their space rather than underlying hardware platform.

4.5 Chapter Summary

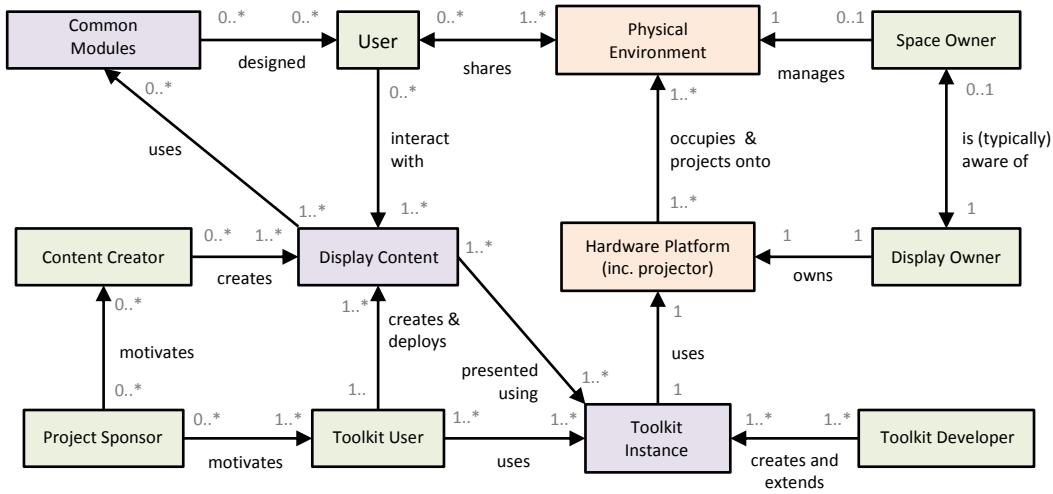


Figure 67: A UML-based class diagram which illustrates the different relationships between the stakeholders, hardware, and software relevant to the toolkit requirements. Green boxes indicate stakeholder. Purple boxes indicate software. Orange boxes indicate physical hardware and space.

The toolkit requirements aim to simplify the process of creating and deploying functional and aesthetically pleasing interactive displays; taking hours not days. The resultant toolkit will lower the barrier to entry for interaction designers and creative developers by allowing *toolkit users* to quickly experiment with designs and create real deployments.

Chapter 5. Toolkit Implementation

5.1 Overview

The previous chapter presented a set of requirements for a toolkit designed to support user innovation with interactive projected displays. This chapter integrates these requirements into a single cohesive design which respects the identified constraints. It also discusses challenging and novel aspects of the implementation.

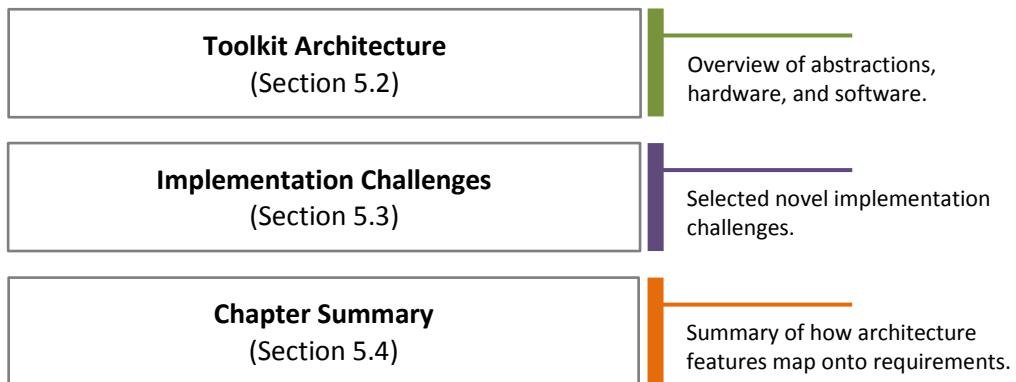


Figure 68: Structure of the toolkit implementation chapter.

This structure of this chapter is summarised in Figure 68. It begins with an overview of the toolkit architecture (Section 5.2) which describes major abstractions, hardware requirements, and software components used in the toolkit. The next section (Section 5.3) provides detail on novel aspects of the implementation, including: the user interface design, the multi-touch interaction support, and the design of a display content API that can query physical surroundings. The chapter concludes with a summary (Section 5.4) which maps implementation features onto the requirements in Chapter 4 and identifies the focus of the toolkit evaluation in Chapter 6.

5.1 Overview

5.1.1 Development Process

Figure 69 describes how the toolkit development process maps across the remaining thesis chapters.

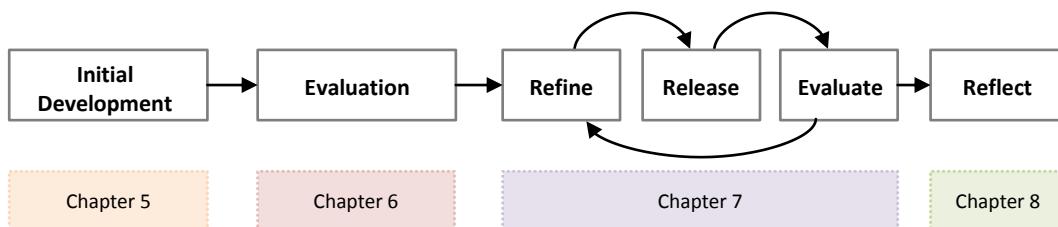


Figure 69: Toolkit development plan (top) mapped across the remaining thesis chapters (bottom).

In order to evaluate the toolkit implementation and validate its suitability for release, it is assessed in Chapter 6 through a series of controlled experiments and user studies. The lessons learned from these experiments are folded back into the toolkit in order to refine its design before release. Once the toolkit is publically released, lessons from its adoption are presented in Chapter 7. Here, a continuous-release approach is taken in order to maximise the time that the toolkits adoption can be studied (Constraint 4: Allow time for toolkit adoption and evaluation). Refinements, bug-fixes, and additional features added during this phase are discussed in that chapter and reflected upon in the conclusions (Chapter 8).

5.1.2 Release Strategy

As the toolkit is primarily a software contribution, before use, *toolkit users* are first required to:

- 1) Download the relevant toolkit software (Constraint 1: Online Distribution).
- 2) Acquire the relevant hardware (Constraint 2: Use only commodity, readily available, or inexpensive hardware).

5.2 Architecture

Following the precedent set by many open source projects—including the WiiTUIO prototype toolkit discussed in Section 3.3.5.3—the release of the interactive projected displays toolkit will be:

- 1) Distributed using the Google Code⁴⁶ open source project hosting platform.
- 2) Branded with the name “*Ubi Displays*” and released under that name to help cultivate a project identity.
- 3) Provided with thorough source code documentation, bug tracking software, video tutorials, and support forums.

Google Code was chosen as the project host as it provides many relevant project management features (i.e. bug tracking) and has a reputation as a safe host of open source content⁴⁷. After the toolkit’s public release, support and bug fixes are provided via the Google Code site. The toolkit will be promoted through online videos which demonstrate its capabilities and how to use it. Links and features on other websites will help to share these videos with wider audiences.

5.2 Architecture

Figure 70 shows the architecture of a single toolkit deployment (i.e. a single set of toolkit software and hardware used to create one or more co-located displays). There are three major hardware components (Section 5.2.1.1) and two major types of software involved: the *toolkit application* software (Section 5.2.3) and the *toolkit content* software (Section 5.2.4).

To create a deployment, a *toolkit user* obtains access to the required hardware, executes the *toolkit application*, configures the hardware, and deploys items of *toolkit content*. A deployment ends when either the *toolkit application* is closed or the hardware configuration disassembled. Save and load support is provided for each deployment, assuming the hardware remains in the same configuration.

⁴⁶ Google Code offers free hosting for open source projects.

⁴⁷ Alternative services are available.

5.2 Architecture

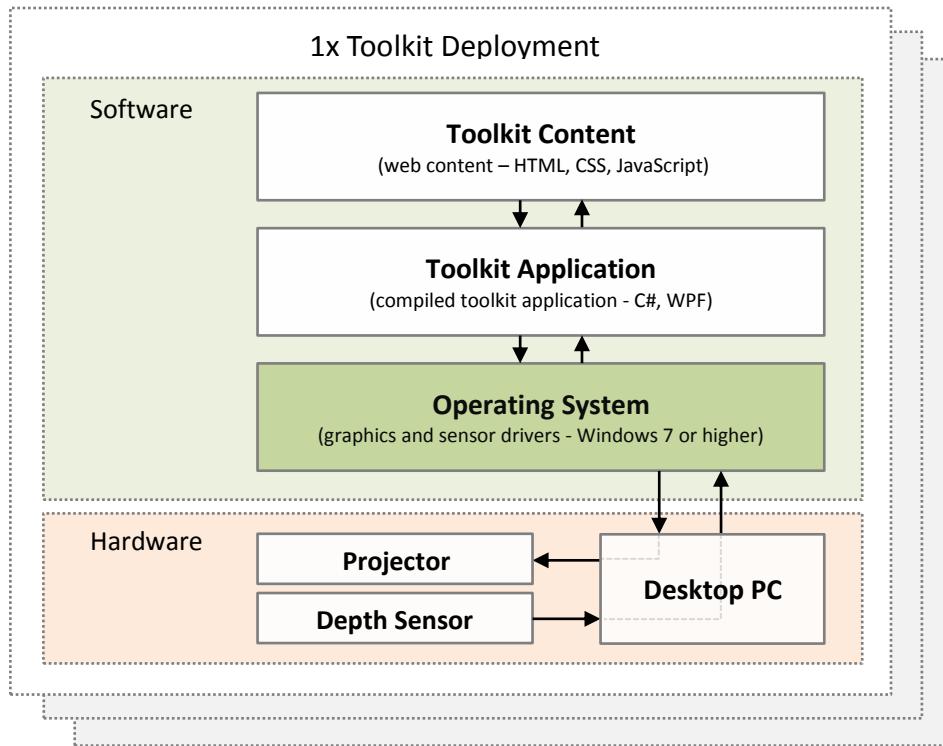


Figure 70: The requirements for a single instance of a toolkit deployment. The operating system is shaded as it is not discussed.

In brief, the *toolkit application* sends video frames to a *digital video projector* (via the operating system and desktop PC hardware) and receives frames of colour and depth data ($\approx 30\text{fps}$) from the *depth sensor* which can then be processed by the *display content* to detect user interaction. The *toolkit application* has a GUI which *toolkit users* interact with in order to calibrate the hardware and deploy content.

5.2.1 Abstractions

The novel capabilities of interactive projected displays (i.e. interaction with digital content across physical spaces) mean that new abstractions are required that enable toolkit users to effectively understand and operate the system (Requirement 3: Simple Abstractions). To achieve this, the toolkit is structured around two key abstractions:

5.2 Architecture

- 1) **Surface** – A named area of physical space designated for content display or as a reference point for interaction detection.
- 2) **Display** – An item of display content which can appear on a surface. A surface can only host one display at any one time, although displays can move between, configure, and query surfaces for their properties.

These abstractions were chosen due to their simplicity and referencing of existing concepts (e.g. people understand that a physical surface has a size and location in 3D space); helping to reduce the learning requirements of new *toolkit users*. Furthermore, they emphasise a separation between the content (i.e. display) and the location that it is deployed in (i.e. surface).

5.2.1.1 Surface

Surfaces are areas of physical space that are defined by toolkit users as being practically sensible for a *display* to appear upon. Each surface is capable of hosting one display at any given time, or can lay dormant, displaying no content at all. Surfaces are given unique names upon creation. Each surface automatically computes metadata such as its orientation and physical size, which is made accessible to the displays. Multiple surfaces can be defined within a single deployment (Requirement 10: Multiple Displays).

Having *toolkit users* pre-define projection surface areas lends itself to a *named-areas* physical addressing scheme⁴⁸. Although pre-defining areas limits the places content can appear, it is more expressive and easier for toolkit users to work with than a large virtual canvas 3D canvas. Furthermore, given that *space owners* often require a high degree of control over the management and aesthetic, manually specifying projected surfaces ensures that the deployment remains controlled by the entity responsible for the space, rather than items of content which may or may not come from trusted sources (Requirement 21: Public Deployments).

⁴⁸ Physical addressing refers to the way in which a toolkit user or item of content is able to programmatically be placed in a physical space (discussed in Section 2.5.1.4).

5.2 Architecture

5.2.1.2 Display

A *display* represents a single item of content. The most common type of display is a ‘web display’ which is composed of graphics, sound, and logic stored as web standard files (i.e. .HTML, .CSS, and .JS). Displays have the ability to perform functions such as querying the surrounding environment to find other display surfaces, and can use this information to alter their design (Requirement 11: Physical Responsive Design) or request an appropriate interaction method for its placement (Requirement 12: Responsive Interaction Design). For instance, content operating on surfaces of a similar size to a finger do not need to support multi-touch interaction.

Displays are not assigned an interaction modality automatically because the toolkit should be agnostic of interaction method (Requirement 18: Decoupled from Platform). The loose coupling between displays and the surfaces hosting them allows display content to ‘jump’ between surfaces (Requirement 8: Programmable Aesthetics).

5.2.2 Hardware

Following a review of implementation technologies (Section 2.3) that are readily available to the target toolkit users (Constraint 2) and do not require surface instrumentation (Constraint 3), three hardware requirements were selected:

- **Desktop PC:** A mid-range PC (circa 2012) with approximately the following specifications: Intel i5 Processor, 2GB RAM, Intel Integrated Graphics Card, 250GB HDD, USB 3.0 Support, VGA and, or HDMI video output (Constraint 2: Use only commodity, readily available, or inexpensive hardware).
- **Digital Video Projector:** Any commodity digital video projector (as described in Section 2.3.1). This includes a range of throw ratios, resolutions, and display technologies. Different types of projector are better suited to different display types (Requirement 7: Variable Display Sizes).
- **Depth Sensor:** A depth sensor can provide point-cloud representations of a physical scene in real time, which can be used to detect various forms of

5.2 Architecture

interaction (Section 2.3.4). The Microsoft Kinect™ was chosen as it is a widely available depth sensor with an active user community. It has a maximum useful sensing range of 3 meters with a 57 degree horizontal FOV⁴⁹.

This combination (Figure 71) inherently scopes the range of displays it is possible to create (Requirement 7: Variable Display Sizes) and interaction types it is possible to sense (Requirement 15: Walk Up and Use). This scope is defined by factors including projector and sensor range, resolution, placement options, available power outlets, and so forth. All three hardware components require a mains power source.

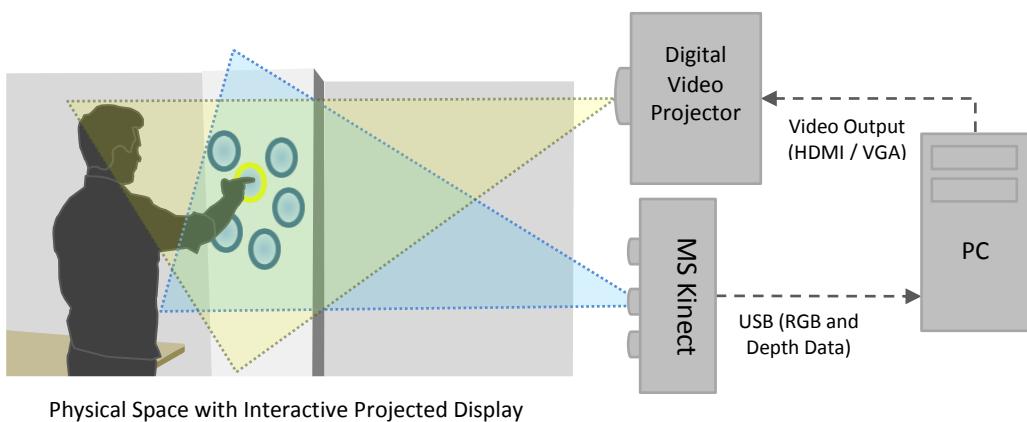


Figure 71: Toolkit hardware requirements deployed. Note sensing and projection frustum overlap.

Varying the hardware used can expand or contract that scope depending on specifications and how they are used. For instance, manually varying the hardware placement allows *toolkit users* to intuitively use hardware capabilities to achieve various effects, such as moving a projector closer to a physical surface to create a brighter, higher resolution, but smaller interface area (Requirement 20: Robust Hardware Placements). This helps the selected hardware cover floor, wall, table, or object sized displays (Requirement 7: Variable Display Sizes) in a range of public and private scenarios (Requirement 21: Public Deployments).

⁴⁹ Kinect™ specifications: <http://msdn.microsoft.com/en-us/library/jj131033.aspx>

5.2 Architecture

Support for additional hardware (e.g. object sensing with a high resolution webcam) can be achieved using toolkit extensions or content items (Requirement 22: Interoperable and Extensible). However, the minimum requirements reduce compatibility issues and ensure an acceptable standard performance (Constraint 2).

5.2.3 Application

There are two main types of toolkit software. The first type is the *toolkit application* software—responsible for hardware configuration, interactive surface layout, and content deployment. The second type of software is items of *toolkit content*—responsible for application specific graphics and behaviour.

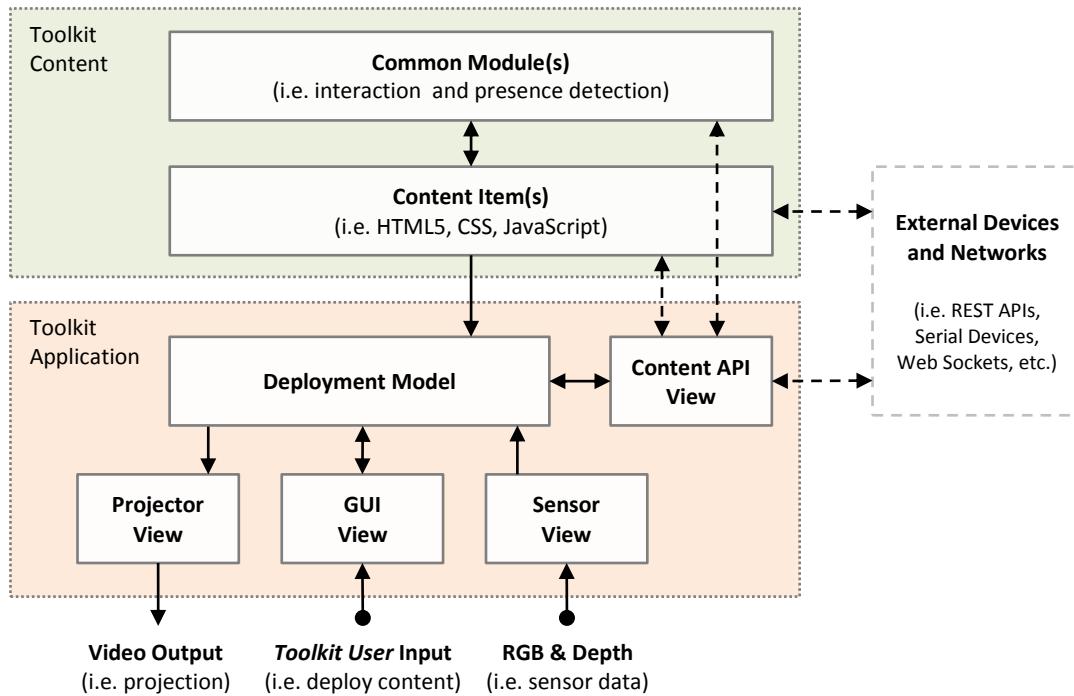


Figure 72: High level overview of toolkit software components. Solid lines indicate required communications. Dotted lines indicate optional communications.

The *toolkit content* is developed by toolkit users and content creators, whilst the *toolkit application* is provided by the author of this thesis and extended by other toolkit developers. Figure 72 presents these two types of software and illustrates their internal structures and the relationships between them. In any deployment,

5.2 Architecture

multiple items of *toolkit content* (i.e. HTML files) can be loaded into a single *toolkit application*. The *toolkit application* architecture is based on the model-view-controller pattern [162]. It features a single model of the deployment which is accessed and updated through four views. Each view is specialised to suit its purpose:

- **Projector View** – Transforms the content graphics and sends them to the projector hardware.
- **Sensor View** – Processes input data from the Microsoft Kinect™ and injects this into the deployment model.
- **GUI View** – Provides an interface with which to configure the toolkit deployment.
- **Content API View** – Provides a moderated mechanism for the toolkit content to access and manipulate the deployment model.

5.2.3.1 Deployment Model

The deployment model maintains a list of *displays* and *surfaces* active in the current *deployment*. Figure 73 shows the components and structure of the deployment model. It makes extensive use of the adapter pattern [162] to ensure extensibility from a *toolkit developer* perspective (Requirement 22: Interoperable and Extensible). Classes in the model implement an `IResource-IResourceOwner` pattern that allows toolkit content to dynamically create new resources, and have these resources automatically released if the content closes unexpectedly or does not implement appropriate resource management (Requirement 4: Graceful Error Handling and Degradation).

The `WebContent` class is an implementation of `IDisplay` that enables content with web-standards support (Requirement 17: Standards Support) and sandboxed JavaScript logic (Requirement 13: Programmable Content). This is achieved using a

5.2 Architecture

specialised Webkit control⁵⁰ capable of rendering the web graphics content to an off-screen texture.

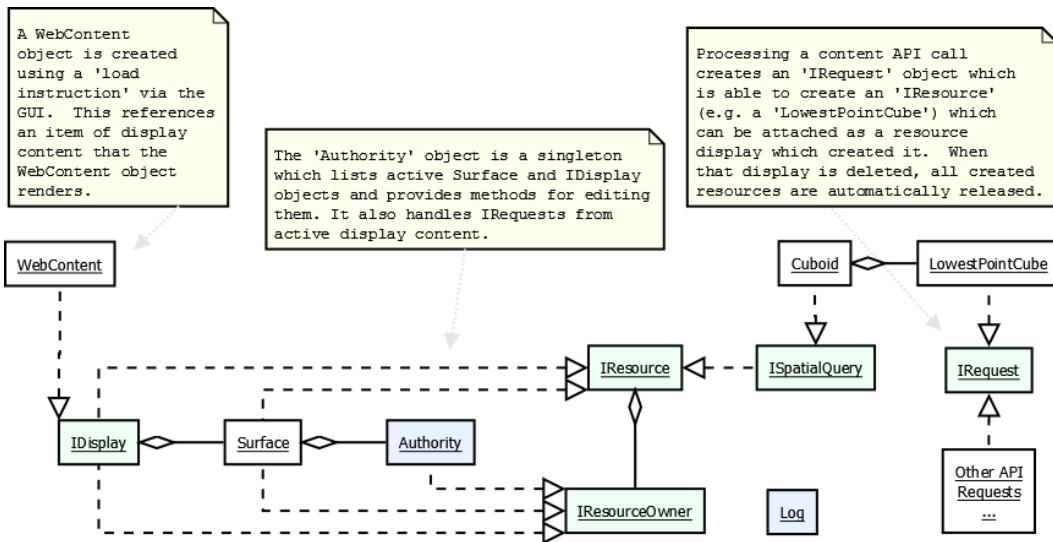


Figure 73: The internal structure of the deployment model as a UML class diagram. Green objects represent interfaces. Blue objects represent singletons. White objects represent standard classes.

To maintain a central list of active Surfaces and any IDisplays (Requirement 10: Multiple Displays) a singleton class called **Authority** is used. It provides methods for attaching and removing displays to surfaces, defining and arranging surfaces, and handling IRequest.

The **Surface** and **IDisplay** definitions reflect the abstractions in Section 5.2.1. A Surface contains zero or one IDisplay implementations at any given time. Only one IDisplay implementation (**WebContent**) is provided with the stock toolkit, although it is possible for *toolkit developers* to extend this to implement other types of content such as areas of the Windows Desktop, or custom WPF controls.

The *Content API View* is exposed to this JavaScript logic through a wrapper object named **Authority** (similar to the `document` and `window` objects available in the

⁵⁰ The Awesomium Web Engine awesomium.com is a wrapper for Google's Chromium v18. Rather than rendering to screen the graphics are rendered into texture memory which can be transformed by the *Projector View*.

5.2 Architecture

W3C specification⁵¹). This allows display content to directly query the `Authority`, `Surface`, and `Display` objects stored in the deployment model. This can be used to determine the physical properties of an interaction surface (Requirement 11: Physical Responsive Design and Requirement 12: Responsive Interaction Design).

Each API request made by an `IDisplay` is submitted to the `Authority` object as an `IRequest` object. The `ISpatialQuery` object is a notable `IRequest` that allows display content to interactively query areas of the physical space for point-cloud data. This interface is implemented by ‘shape’ objects (e.g. `Cuboid`) that process regions of point cloud data (i.e. `LowestPointCube`) or stream it directly into the display content (e.g. `Cuboid`).

A `Log` singleton is able to track the happenings within the deployment model and content. If errors are detected, their source (i.e. content file and line number, or internal toolkit application error message) and message can be captured and inspected in more detail (Requirement 4: Graceful Error Handling and Degradation).

5.2.3.2 Views

The four views as shown in Figure 72 are the *Projector View*, the *Sensor View*, the *GUI View*, and the *Content API View*. Each provides a specialised way of configuring, querying, or updating the deployment model. Their implementations are described in the subsections below.

Projector View

The *Projector View* is responsible for transforming the graphical output of an object implementing the `IDisplay` interface into the appropriate format for projection into the physical space (Figure 74). As not all physical surfaces lie planar and orthogonal to the projector, the graphics for each are distorted using a non-affine transformation matrix to achieve a projection mapping effect (Requirement 9: Projection Mapping, see Section 2.3.2). This technique allows multiple displays (Requirement 10: Multiple Displays) of various sizes and shapes (Requirement 7:

⁵¹ W3C Window Object 1.0: <http://www.w3.org/TR/Window/>

5.2 Architecture

Variable Display Sizes) to be created using a single projector which can be placed in a variety of scenarios (Requirement 20: Robust Hardware Placements).

By keeping this stage separate from the content rendering stage, it is possible to smoothly move the graphical displays around the physical space (Requirement 8: Programmable Aesthetics) without affecting the content rendering or content design process. To achieve this, the projection renderer is implemented using a WPF `Viewport3D` control⁵². Sample projector output is shown in Figure 74.

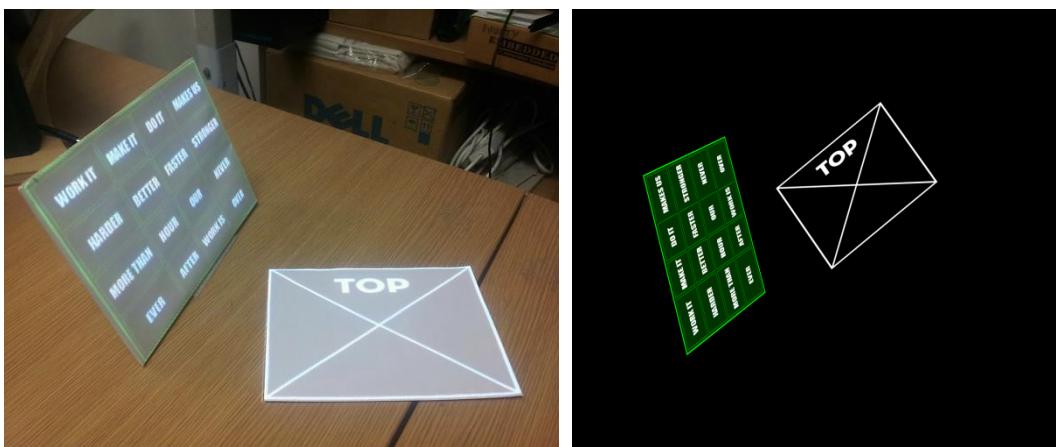


Figure 74: Left: Two displays are correctly projected onto two flat physical surfaces which do not lie planar to a projector frustum. Right: The display output sent to the projector. In this instance, the projector is located directly above the displays.

Sensor View

The *Sensor View* is responsible for processing the input data from the Microsoft Kinect™ and injecting this into the deployment model. The main challenge is to quickly poll the device for colour and depth frames (received as bitmaps), and then marshal this data into a format which enables it to be queried in real-time by the display content (i.e. point-cloud data). To achieve this, the sensor view uses a doubled-buffered multi-threaded pattern to allow simultaneous updates and spatial queries Figure 75.

⁵² Viewport3D renders 3D content within 2D bounds:
[http://msdn.microsoft.com/en-us/library/system.windows.controls.viewport3d\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.windows.controls.viewport3d(v=vs.110).aspx)

5.2 Architecture

To enable the `ISpatialQuery` objects to have concurrent access to the most recent version of point cloud data, whether a query is executed during this lock or not, the last good data frame is locked (L_{lock}), the query executed on that frame, and then released ($L_{release}$). When a new frame of sensor data is received, the ‘next’ item on a double buffer is locked (N_{lock}) and unlocked ($N_{release}$) once the data is written. After this, the buffers are flipped. This means that new frames of data can be processed while spatial queries are being executed and minimises the minimum wait time a thread.

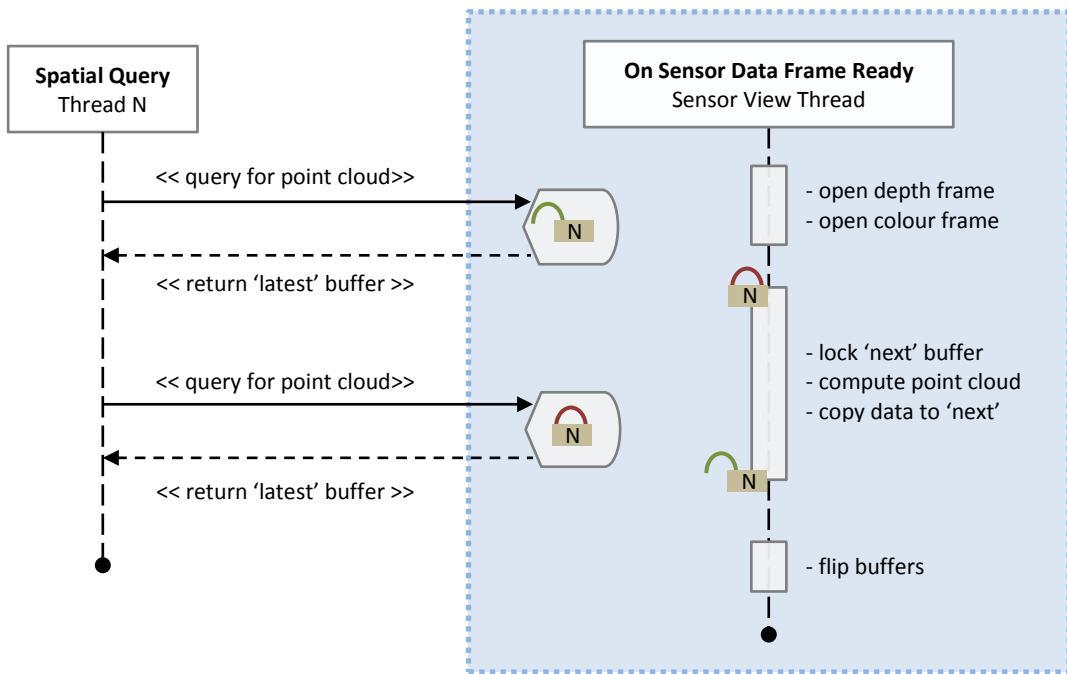


Figure 75: A double buffered threading model is used to perform spatial queries whilst concurrently receiving new data frames.

GUI View

The *GUI View* is a Windows desktop application implemented using C# and the WPF interface framework. Its purpose is to provide *toolkit users* with an interface to manage most aspects of a toolkit deployment, including:

- Calibrating the hardware for the deployment environment (Requirement 6: Minimise Calibration).

5.2 Architecture

- Defining, naming, and configuring the surfaces which displays are deployed onto (Section 5.2.1).
- Deploying and managing display content onto the surfaces (Requirement 5: Rapid Configuration, Section 5.3.1.3).
- Debugging display content (Requirement 4: Graceful Error Handling and Degradation).
- Providing feedback on the toolkit and the application scenarios that it is being used in.

The design of this interface connects the *toolkit user* to the physical environment in an intuitive and designed to cater to the *toolkit user's* context and capabilities (Requirement 1: Lower Skill Barriers, Requirement 2: Easy Toolkit Interface Operation). How this is achieved is discussed in more depth in Section 5.3.1.

Content API View

The *Content API View* grants display content access to the *Deployment Model* through a series of functions (Table 10) and properties (Table 11). These include being able to query the physical properties of display surfaces, manipulate the deployment model, and communicate with other display content via requests handled by the Content API that implement the `IRequest` interface. The `IRequest` interface also implements a static `HandleName` string property which acts as a unique identifier to the `Authority` object.

Table 10: Functions exposed by the Content API view.

<code>Authority.log</code>	Writes JavaScript objects (including strings) to the debug log, whilst referencing the display content which sent the message.
<code>Authority.request</code>	Invokes a specific <code>IRequest</code> function using a <code>HandlerName</code> string and a dictionary of arguments which are passed as parameters. If an <code>IRequest</code> object with the corresponding <code>HandlerName</code> is registered with the <code>Authority</code> object, an instance of the relevant <code>IRequest</code> is created, and its <code>Process</code> method called.

5.2 Architecture

Authority.call	Allows one item of display content to invoke a function on another item of display content hosted on another display surface (Requirement 4: Inter-display Communication). The content does not have to implement this function, as it is invoked on a best-effort basis. Any errors are written to the debug log. It takes two parameters: a ‘surface name’ string used to identify the surface hosting the content, and a set of ‘data’ objects passed as variadic arguments.
-----------------------	---

Table 11: Object properties exposed by the Content API view.

Surface.Name	The name of the surface this display is deployed on as used in the physical addressing scheme.
Surface.Width	A floating point value which contains the approximate width of the current display surface in meters.
Surface.Height	A floating point value which contains the approximate height of the current display surface in meters.
Surface.Angle	A floating point value which contains the approximate angle of the current display surface in degrees relative to a the calibration plane (Section 5.3.1.2).
Surface.AspectRatio	The aspect ratio of the current display surface given as width (meters) over height (meters).
Surface.TargetDPI	The render resolution of the content item as when processed by the WebDisplay (e.g. 800x600 pixels). Stored as an array in the format: [width, height].
Surface.ActualDPI	The estimated number of pixels actually used to render the content in the Projector View. Stored as an array in the format: [width, height].

The *Content API View* is designed to be extensible by *toolkit developers*. To make it easy to add or remove new API calls, when the application starts it scans the toolkit namespace for classes which extend the `IRequest` interface. These are then automatically registered with the `Authority` object using their static `HandleName` string property.

5.2.4 Content

As described in Section 5.2.3, *toolkit content* is loaded into the *toolkit application* where it is processed and rendered. There are two main types of content software: (1) *display content*—the graphics, logic, and sound for the application scenario, and (2) *common modules*—reusable libraries which make developing content easier.

Figure 76 illustrates the relationship between display content and common modules by showing a piece of display content which includes a module (presence-

5.2 Architecture

detection.js) in order to play a sound when a physical object is placed on top of the content.

```
1 <html>
2   <head>
3     <title>Presence Detector</title>
4     <!-- Import presence detection support from common modules. -->
5     <script src="common/presence-detection.js" type="text/javascript"></script>
6   </head>
7   <body>
8     <script type="text/javascript">
9       /** Once the page has loaded, create a presence detector. */
10      document.onreadystatechange = function()
11      {
12        // Ignore non-complete events.
13        if (document.readyState !== 'complete')
14          return;
15
16        // Create a new presence detector called 'area'.
17        // 0.1m high with a 0.02m offset.
18        var pd = new PresenceDetector("area", 0.1, 0.02);
19
20        // Play a sound if an object enters or leaves.
21        pd.onStateChange = function(state){ new Audio("sfx/there.wav").play(); }
22      };
23    </script>
24  </body>
25 </html>
```

Figure 76: A sample content item which uses a presence detector to play a sound when a physical object is placed on or taken off it. This demonstrates inclusion of a common module (line 5) and its use (lines 18-21).

5.2.4.1 Display Content

The toolkit *display content* is implemented using web-standard formats (HTML5, CSS3, JavaScript, WebGL, Flash, etc.) as they offer many of the required rich-media features (Requirement 17: Standards Support), support for programmable content (Requirement 13: Programmable Content), and remove the need for *toolkit users* to learn a specialist display language (Requirement 1: Lower Skill Barriers). Furthermore, it automatically qualifies a large pre-existing user base to develop content for the toolkit and capitalises on a pre-existing wealth of existing content, interface libraries⁵³, transferable skills, community support, and development experience.

A limitation of focusing on web standards is that it may force people to create new interface designs, rather than re-using existing Windows desktop applications in a projected context. However, this limitation has the virtue of forcing people to think outside the box and design interfaces tailored to physical spaces—the

⁵³ Many 3rd party libraries exist to simplify the process of developing engaging interactive web content. For instance, the JQuery library: <http://jquery.com/>

5.2 Architecture

exploration of which is a driving motivation for the toolkit. With that said, it is possible for toolkit developers to extend the types of supported content by implementing the `IDisplay` interface.

Display content items are stored as files, either locally or remotely. Using scripted rather than compiled content decouples *display content* from the underlying platform (Requirement 18: Decoupled from Platform). This allows content to be quickly loaded, unloaded, edited and, tested at runtime without recompilation⁵⁴ (Requirement 5: Rapid Configuration).

5.2.4.2 Common Modules

Common modules are reusable libraries (written with web standards) that make developing content for interactive projected displays easier. These serve a range of functions: graphical design, interface widget libraries, new interaction modalities, external service integration, and so forth. Examples of 3rd party common modules used in display content include JQuery, JQueryUI, GLSL.js, Adobe Flash player, and so on. The interactive projected displays specific modules provided with the stock toolkit focus on:

- Supporting graphics specification using physical dimensions (Requirement 11: Physical Responsive Design).
- Multiple interaction modalities such as presence detection and multi-touch (Requirement 15: Walk Up and Use).

Placing display-specific functionality in scripted modules rather than integrating it into the underlying toolkit or operating system (Requirement 18: Decoupled from Platform) has a number of effects:

It allows display content to have a very close relationship with the physical space that contains it. For instance, content can query its surroundings and use this information to make an informed decision about how to present itself to the user. Or

⁵⁴ This is different to the approach taken by existing toolkits such as dSensingNI [84], ProjectorKit [119], and WorldKit [27] where content is compiled into an application.

5.2 Architecture

similarly, choose an interaction modality which is more suitable to the display placement or user needs (Requirement 12: Responsive Interaction Design).

It allows the content to react to dynamic surface conditions (Section 2.3.2.3). For instance, should a display be moved from a large horizontal surface to a small vertical surface, the content could respond to these changes by switching from a table-top layout viewable from any angle to an ordered list with smaller text (Requirement 7: Variable Display Sizes and Requirement 8: Programmable Aesthetics).

It enables multiple items of content to be interacted with at once, using different modalities and methodologies. The single-user assumption built into the Windows operating system is not transferred into the toolkit content. However, if such a condition is required, it must be built in by hand.

The interaction sensing modality can be tweaked to suit particular and niche requirements. Content items can be optimised for detecting interaction in specialised circumstances without recompilation of the toolkit. For instance, detecting touch on water or through glass.

Performance degradation is isolated to the content item. As each item of content is executed in its own process, if one content item is processor intensive, surrounding content items are not affected (Requirement 4: Graceful Error Handling and Degradation). This is useful in cases where content is programmed poorly, uses lots of resources, or is interaction sensing intensive.

Extensible to new modalities. The toolkit is not restricted to the interaction modalities provided with the stock toolkit (Requirement 22: Interoperable and Extensible). For instance, it would be possible to script in new interaction modalities and share them with the community.

The major drawbacks of processing interaction sensing as part of the display content are lower performance and more difficult native hardware access (as the content must communicate with an external interaction event service, such as data streamed through a web socket). Typically, these are handled either by the operating system (i.e. a mouse) or a specialist application (i.e. WiiTUIO) that can balance performance and accuracy. Subsequently, providing responsive and

5.3 Implementation Challenges

accurate interaction sensing using JavaScript is challenging. The approach taken is discussed in the next section and is a focus of the evaluation in Chapter 6.

5.3 Implementation Challenges

This section presents the operation, challenges, and limitations of novel aspects of the toolkit implementation. This covers the interface design, the algorithms for the multi-touch and presence detection interaction modalities, the content threading model, and the features of an environmentally aware display content API. These were selected for discussion based on their novelty.

5.3.1 Application GUI Design

The toolkit application provides an interface for toolkit users to configure and manage interactive projected display deployments. Its design is intended to avoid domain specific language (Requirement 1: Lower Skill Barriers) and complex operations. To achieve this, the *toolkit user interface* adopts a wizard-based structure, with a strong emphasis on visual elements. The wizard-based structure was chosen as it is an effective method of simplifying serial complex tasks and guiding users with no previous experience through new processes [163] (Requirement 1: Lower Skill Barriers, Requirement 2: Easy Toolkit Interface Operation, and Requirement 5: Rapid Configuration).

The implementation presents *toolkit users* with a sequence of tab-screens which lead them through a series of well-defined steps: (1) hardware configuration, (2) interface calibration, and (3) surface and display content management. Advanced options relating to each step are presented in context rather than through a drill-down menu which would require *toolkit users* to know what to look for. Features which do not fit into these steps (i.e. save, load, view debug log, and close application etc.) are present as buttons along the bottom or tabs along the top of the application (Figure 77).

5.3 Implementation Challenges

5.3.1.1 Step 1 – Hardware Selection

To begin, the *toolkit user* must select the projector from a list of display outputs (i.e. projectors and monitors) and the depth sensor from a list of available depth sensors (Figure 77). Once these are chosen, the program immediately jumps to the next step. Although it is possible to perform this step automatically, it is shown to the *toolkit user* to help make them aware of the process and to give them the option of running multiple simultaneous deployments on a single Desktop PC.

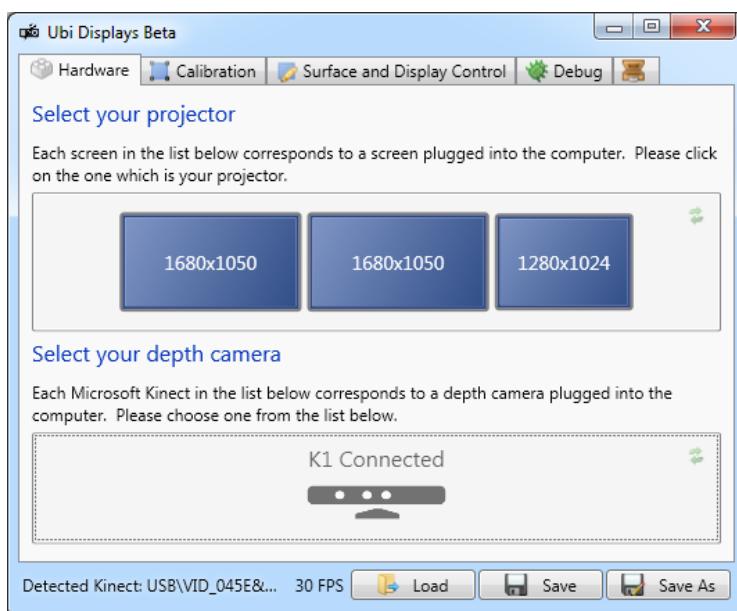


Figure 77: Toolkit User Interface hardware selection screen.

5.3.1.2 Step 2 – Interface Calibration

Next, the *toolkit user* calibrates the hardware to create a correspondence between the projected image and the view of the depth sensor. During calibration the projector will display four planar calibration points in sequence which the *toolkit user* must click on in the depth sensor video feed using the mouse (Figure 78). If they cannot be seen in by the sensor, they can also be dragged using the mouse. To more accurately select these points, the video can be panned and zoomed using the mouse scroll wheel and right mouse click.

5.3 Implementation Challenges

The calibration points are used to construct two homography matrices (described in Section 2.3.2.1) which map coordinates in the video image (2D) and sensor space (3D) into the area covered by the projector. Although this has the drawback of not accounting for the parallax distortion of the projector lens, this approach was favoured over an eight-point non-planar calibration as this would add complexity and is unnecessary for most planar display configurations.

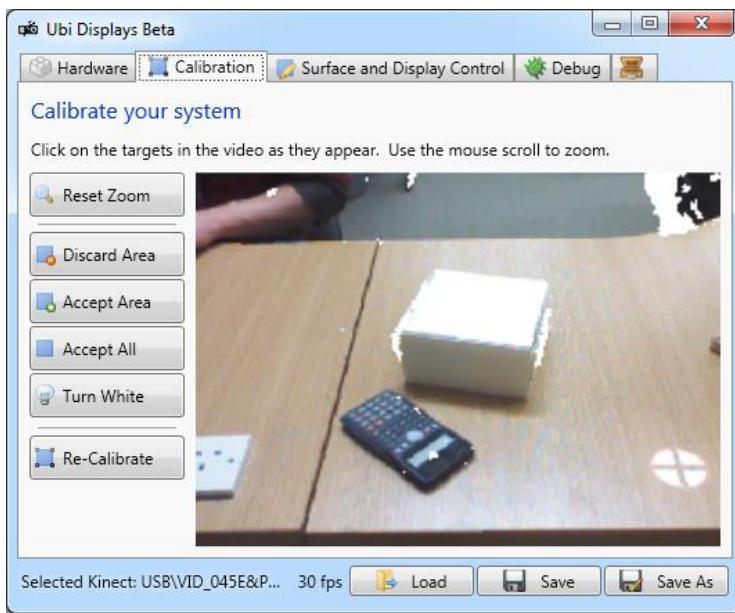


Figure 78: Toolkit User Interface hardware calibration screen. Showing the first of four projected calibration points in the bottom right of the image.

5.3.1.3 Step 3 – Surface and Display Management

The last step is dedicated to the creation of surfaces and deployment of display content. To define a new surface, toolkit users select the ‘Draw Surface’ button and then, using the mouse, ‘draw’ a display over the desired location in the live video feed (Figure 79) in approximately the desired location, size, and shape. This approach was chosen over a coordinate entry approach as it is quick to use (a matter of seconds) and exploits *toolkit users’* inherent visual understanding of the space.

The rotating callipers method [164] then snaps the ‘drawn’ area to a best-fit rectangle; providing bounding corners for the surface. Internally, the data generated

5.3 Implementation Challenges

provides each newly created `Surface` object with reference coordinates for each of its corners in the projected image (2D), the Kinect™ video feed (2D), and the Kinect™ point cloud (3D), thus removing the need for toolkit users to manually calibrate each surface (Requirement 6: Minimise Calibration).

As not all potential surfaces are planar and orthogonal to the projector, it is possible to adjust each surface's projection and sensing coordinates by dragging the corners of the surface with the mouse on the live video feed (using Click+Drag, and Shift+Click+Drag respectively). This feature can be useful in cases where the default interface calibration (created in Step 2) does not provide enough accuracy, the sensor has been subject to slight drift, or the *toolkit user* wishes to separate the projection and interaction sensing areas. Surfaces with non-rectangular geometries are not supported.

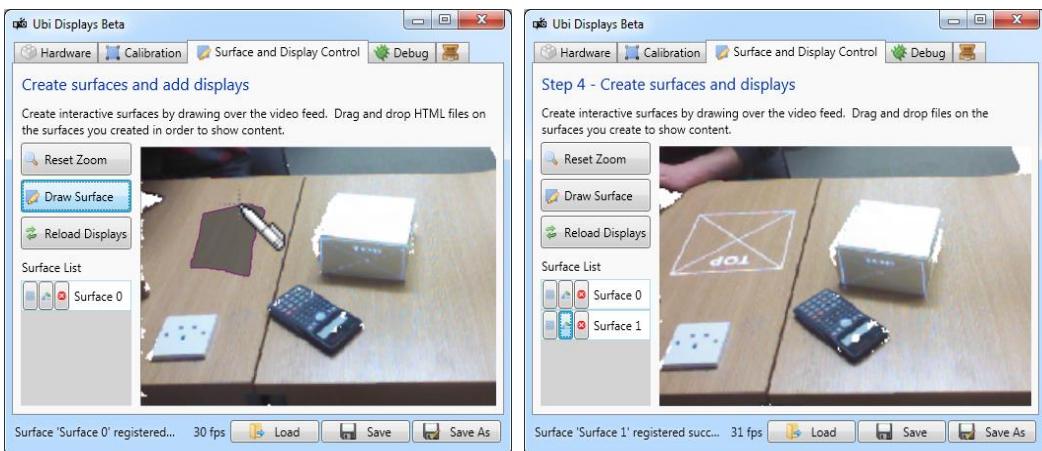


Figure 79: Toolkit User Interface surface creation and content deployment screen. Left shows a toolkit user 'drawing' a surface on the video. Right: The surface as projected.

To deploy content, the *toolkit user* drags-and-drops a file from the file system explorer directly onto the target surface in the video. When hovering over the video and dragging content, all the possible target surfaces are highlighted using a transparent green. This indicates that surfaces are there, even if no content is present. Any supported content (i.e. a .HTML, .JPG, .PNG files, etc) is automatically loaded and displayed. An error is written to the debug log if the content type is not supported.

5.3 Implementation Challenges

It is also possible to deploy content using a text-string URL in the drag-drop manner or via text box entry. This file or URL string acts as a ‘load instruction’, which the *deployment model* uses to create a WebContent IDisplay instance.

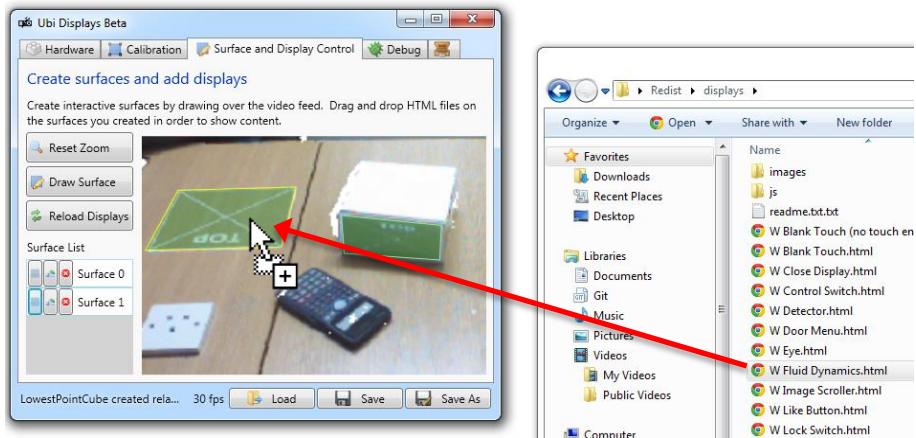


Figure 80: Toolkit User Interface display content deployment. A toolkit user drags a .HTML file onto the desired surface.

As with the calibration interface (Step 2), the video feed can be panned and zoomed using the right mouse button. A list of surfaces is provided to the left of the image which allows them to be manually configured and removed. Accelerator buttons (show debug image, rotate, and delete—left to right) are provided next to the name of each. Double clicking on the surface name (i.e. “Surface 0”) allows the *toolkit user* to enter more advanced properties such as custom names, render resolutions, and so forth.

5.3.2 Multi Touch Support

The multi-touch technique is the most complex interaction method offered natively by the toolkit. Unlike other implementations it cannot rely on the placement of the camera lens for an optimal perspective. The list below highlights three main implementation features that distinguish it from existing methods:

5.3 Implementation Challenges

1. Point cloud based touch detection rather than histogram based (Section 2.4.1) sensor perspective (supports touch detection from arbitrary sensor angles—Requirement 20: Robust Hardware Placements).
2. Multiple instances of the algorithm can run on the same data simultaneously (supports multiple simultaneous users—Requirement 10: Multiple Displays); meaning multiple content items can be concurrently used by multiple people from a single projector and sensor.
3. Scripted implementation as it does not generate platform specific multi-touch events (supports Requirement 18: Decoupled from Platform).

As described in Section 5.2.4.2, multi-touch interaction is implemented in JavaScript and operates by injecting touch events (W3C specification [120]) 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, and configure it with the JavaScript code listed in Figure 81.

```
1 <html>
2   <head>
3     <!-- Give the content a black background. -->
4     <style type="text/css">
5       html, body { background-color:black; }
6     </style>
7     <!-- Import the multi-touch module. -->
8     <script src="js/ubidisplays-multitouch-0.8.js" type="text/javascript"></script>
9   </head>
10  <body>
11    <script>
12      $(document).ready(function() {
13        // Configure and start the multi-touch algorithm.
14        var multitouch = new KinectTouch({
15          debug : true,           // Turn on debug points.
16          trails : true,          // Turn on finger trails.
17          point_limit : 200,       // The number of points allowed to process.
18          surface_zoffset : 0.015, // The offset from the surface to capture (meters).
19          height : 0.01,           // The height from the surface to capture (meters).
20          sendemptyframes : true, // Send the empty frames or not?.
21        });
22      });
23    </script>
24  </body>
25 </html>
```

Figure 81: Sample code for a content item with a simple black background that demonstrates how to add multi-touch support to a content item.

5.3 Implementation Challenges

Touch events are detected and then injected into content by following the process described below:

1. ***JS - Issue Request***: Request that the toolkit Content API provide point-cloud data from a desired area of 3D space (relative to a known interaction surface).
2. ***C# - Spatial Query***: On receipt of each new frame of data from the depth camera, the `ISpatialQuery` culls points outside the requested region, transforms them into the coordinate space of the surface (simplifying later calculations), and then dispatches these points back into the browser as arguments passed to a JavaScript function.
3. ***JS - Clustering***: The point-cloud data is clustered a using kd-tree [165] enhanced DB Scan algorithm [166] which groups points based on neighbour density. Features for each cluster are also computed, including: number of points, size, density, aspect ratio, and mean centres.
4. ***JS - Spatio-Temporal Grouping***: Using the computed features, these clusters are matched against clusters detected in the previous frame. New clusters are given a unique identifier and old clusters (which have not appeared for a number of successive frames) are removed. Clusters which do not match the shape-profile for a finger (based on target physical sizes) are rejected.
5. ***JS - Touch Injection***: Using the computed cluster features, touch events compatible with the W3C specification [120] are injected into the DOM.

In summary, this process involves: checking for the presence of physical objects slightly elevated above a surface plane, clustering, filtering, and grouping them based on the previous frame, and then injecting them into the DOM. The detection process differs from histogram-based algorithms used to optically detect multi-touch (including Wilson [37], Dippon et al. [85], and Klompmaker et al. [84]) as it operates on point-cloud data rather than an optically contiguous image frame.

5.3 Implementation Challenges

Advantages of this approach include: (1) a greater robustness to different sensor positions and orientations (Requirement 20: Robust Hardware Placements), (2) it is considerably easier to integrate data from multiple sensors by merging point clouds, (3) it can operate on comparatively small amounts of data (in comparison to histogram based methods), and (4) does not require a model of the background scene, which makes it more suitable to scenarios with unexpected physical changes or ‘imaginary’ surfaces not attached to physical objects.

Drawbacks of this point-cloud approach include: (1) that it is harder to extract information derived from the perspective of the sensor (i.e. the curve of a finger), (2) that it is slower to process in JavaScript than C#⁵⁵, and (3) that it can be computationally expensive in high-density point-clouds due to the computational complexity of the DB Scan algorithm.

This method is the subject of evaluation in the next chapter as it differs from other methods in the literature and providing walk-up-and-use interaction modalities (Requirement 15: Walk Up and Use) suitable for use public scenarios (Requirement 21: Public Deployments) are important requirements to be met if the toolkit is to be considered suitable for adoption. It is also important to evaluate the use of JavaScript as a way of implementing interaction sensing techniques which are decoupled from the underlying platform (Requirement 18: Decoupled from Platform).

5.3.3 Presence Detection Support

Presence detection is another interaction modality natively supported by the toolkit. Although it features less information bandwidth than multi-touch, its strengths include that it is simple to implement and versatile. It can be used to create a number of different effects [109]. For example:

⁵⁵ JavaScript vs C# execution speed comparison for binary trees:
<http://benchmarksgame.alioth.debian.org/u32/performance.php?test=binarytrees>

5.3 Implementation Challenges

- Detecting when a user touches anywhere on a surface to create a primitive switch. This is useful for smaller surfaces without the overhead of running a full multi-touch detector.
- Detecting when any item is placed on top of a surface. This can be used to activate graphical or audio notifications, or to trigger functionality in another display using the `Authority.call` API call.
- Determining the approximate size and shape of the physical objects placed over particular regions of a display. For instance, detecting legs over areas of a map floor display.
- Detecting the presence of a hand hovering nearby a surface to create a basic gesture detector which searches for the presence of an object above, atop, behind, or to the side of a surface.

There are two layers through which users can use the presence detection processing: *direct* and *wrapped*. The *direct* method (Figure 82) allows content to directly receive point-cloud frames from the toolkit Content API. This gives the content more control over how the data is processed at the cost of requiring more programming skills (i.e. to develop a multi-touch or specific gesture sensor).

```
1 // Request the point cloud data in a cube above the surface (z-sorted).
2 Authority.request("KinectLowestPointCube", {
3   relativeto : Surface.Name,           // The surface we want the cube relative too.
4   surface_zoffset : 0.02,             // The bottom of the cube off the surface (in meters).
5   height:0.10,                      // The height from the surface+offset (in meters).
6   callback : "handle_LowestPoints", // The function we want to call back with point data.
7   point_limit : 50,                 // The max number of points to accept.
8   sendemptyframes : false,          // Do we want callbacks when we have empty frames?
9 });
10
11 /**
12 * @brief Called by the toolkit Content API with point cloud data.
13 * @param pointList A list of points in the format: [[x,y,z],[x,y,z],...]
14 */
15 function handle_LowestPoints(pointList) {
16   // If we have more than 40 points.
17   if (pointList.length > 40) {
18     // ... do something ...
19   }
20   // We have less than 40 points.
21   else {
22     // ... do something ...
23   }
24 }
```

Figure 82: JavaScript code demonstrating the ‘direct’ method of presence detection which accesses the raw point-cloud data above a surface.

5.3 Implementation Challenges

The *wrapped* method (Figure 83) provides a simplified way of handling presence detection events by invoking a specific function once specific conditions have been met.

```
1 // Create a new presence detector. Args: name, height, surface offset.  
2 var pd = new PresenceDetector("Video", 0.1, 0.02);  
3  
4 // Called when *any object* either enters or leaves the surface.  
5 pd.onStateChange = function(bState) {  
6     // Do something here...  
7 }
```

Figure 83: JavaScript code demonstrating the ‘wrapped’ method for achieving presence detection.

By including presence detection as well as touch interaction, the toolkit offers a broader vocabulary of interaction capabilities out of the box. Experimentation with these may lead to a greater understanding of the trade-offs between different modalities in different contexts.

5.3.4 Content Threading Model

To provide graceful performance degradation (Requirement 4: Graceful Error Handling and Degradation) for content items with potentially highly varied performance profiles, the toolkit implementation exploits the multi-core architecture of the target hardware profile (Section 5.2.2) to strive for independent content item performance that scales to multiple simultaneous items and users.

To achieve this, items of *display content* (and subsequently any interaction modules that use) are executed in separate processes and synchronised only when necessary. This prevents the performance of one content item from impacting another and maximises the performance of multi-display content and interaction on commodity multi-core processors. Synchronisation between the content items is provided by an ordered asynchronous message queue. This has the advantage of speed, but makes it harder to design systems of stateful distributed content. Examples of this synchronisation include `Authority.call` for inter-display communication (Requirement 16: Inter-display Communication) and other Content API requests.

5.3 Implementation Challenges

5.3.5 Content API Features

This section describes the stock features of the toolkit Content API View (introduced in Section 5.2.3.2). This provides content with an ability to query their physical surroundings, which in turn allows multiple content items (Requirement 10) to implement responsive graphics (Requirement 11), interaction modalities (Requirement 12) regardless of the display size (Requirement 7). It also enables displays to appear, disappear, and move around their environment (Requirement 11) and react to changes in display size in real time (Requirement 5, Requirement 8).

Table 12 documents all the API requests as provided by the stock toolkit. All the items listed with the '*C# API*' label are accessed through the `Authority.request` mechanism described in Section 5.2.3.2 (Content API View). All of them items listed with the '*JS API*' label are accessed as standard JavaScript functions unless otherwise specified.

Table 12: List of functions provided with the stock toolkit Content API.

Function Name		Documentation
swapdisplay	C# API	<p>Swap this display with one on another surface. Content on the target surface will be moved to the calling surface.</p> <ul style="list-style-type: none">- <code>target</code>: The name of the Surface which this display content should jump too.
movedisplay	C# API	<p>Move the calling display to another surface. This will fail if content is already present on the target surface.</p> <ul style="list-style-type: none">- <code>target</code>: The name of the Surface which this display content should jump too.- <code>force_reload</code>: Should the display content be re-loaded during the move, or remain active.
swaptargetdisplay	C# API	<p>Swap a display on a target surface with another target surface.</p> <ul style="list-style-type: none">- <code>target1</code>: The name of surface which contains the display to be moved.- <code>target2</code>: The name of the surface which will receive the display.
movetargetdisplay	C# API	<p>Move a target display from one surface to another.</p> <ul style="list-style-type: none">- <code>source</code>: The name of surface which contains the display to be moved.- <code>dest</code>: The name of the surface which will receive the display.- <code>override</code>: Should content on the destination surface be closed. True or False.- <code>force_reload</code>: Should the display content be re-loaded during the move, or remain active.

5.3 Implementation Challenges

closedisplay	C# API	Close the calling display. This will delete the display from the deployment model and free up all its resources.
closetargetdisplay	C# API	Close the display on a given surface. <ul style="list-style-type: none"> - target: The name of the Surface which is currently showing the display to be closed.
surfacelist	C# API	Return a list of surface names active in the current deployment model back to the calling display. <ul style="list-style-type: none"> - callback: The JS function to be called back with the results.
surfaceinfo	C# API	Return information about a surface. <ul style="list-style-type: none"> - surfaces: An array of string surface names to get the information for. E.g. [“Surface 0”]. - callback: The JS function to be called back with the results. Returns a dictionary of results with surface names as keys. Missing names are not returned.
playsound	C# API	Play a sound at a specified file. This is deprecated - use HTML5 audio tags where possible. <ul style="list-style-type: none"> - file: The path to the sound file to play.
opendisplay	C# API	Open a display on another surface. <ul style="list-style-type: none"> - target: The name of the surface which the content should be opened on. - load: The ‘load instruction’ (i.e. URL) which should be opened on the new surface. - override: A Boolean which determines if any existing content on the target surface should be closed.
kinectlowestpointcube	C# API	Create a lowest point cube and attach it as a display resource. <ul style="list-style-type: none"> - relativeto: The name of the surface to detect points relative too (i.e. above the current surface). - callback: The function in the display content which will be called with the results. It will accept data in the format: [[x,y,z],...] - surface_zoffset: The offset from the surface to start detecting points. - height: The height at which to stop detecting points (+the offset). - point_limit: The maximum number of points to send in any one frame. - sendemptyframes: Should empty data frames be sent. - sendemptysuccessiveframes: Should empty data frames AFTER the first empty frame be sent.
Convert_P2M	JS API	Converts x pixels to meters for the display that calls it <ul style="list-style-type: none"> - value: The number of pixels to convert. E.g. 100 or 2.23 - axis: The dimension to convert using. “w” for width. “h” for height. - return: The resultant number of meters for ‘value’ pixels.

5.4 Chapter Summary

Convert_M2P	JS API	<p>Converts x meters to pixels for the display that calls it</p> <ul style="list-style-type: none"> - value: The number of meters to convert. E.g. 0.2 or 2 - axis: The dimension to convert using. "w" for width. "h" for height. - return: The resultant number of pixels for 'value' pixels.
PresenceDetector	JS API	<p>A presence detector function that will check for the presence of an object in a particular area above (or below) a display surface.</p> <ul style="list-style-type: none"> - sName: A unique name for this detector. e.g. "low", "middle" or "high", or "switch". - fHeight: The height of the area in meters above the display. In meters. - fOffset: The offset from the surface to start detecting at. In meters. - iCountLimit: The number of points required for a solid detection. <p>Usage:</p> <pre>var pd = new PresenceDetector("area", 0.1, 0.02); pd.onFound = function() { console.log("present"); } pd.onLost = function() { console.log("not present"); } pd.onStateChange = function(bState) { console.log(bState); }</pre>
KinectTouch	JS API	<p>A class which adds multi-touch to a page. Uses the same control arguments as 'kinectlowestpointcube' with the following additional parameters:</p> <ul style="list-style-type: none"> - debug: Should the touch detector render each touch point using a coloured circle. True for yes. False for no. - trails: Should the touch detector render each point in the point cloud detected above the surface. True for yes. False for no. <p>Usage:</p> <pre>var kt = new KinectTouch({ debug : true, height : 0.02 });</pre>

Aspects of this functionality that interact with the deployment model are implemented in the toolkit application. However, aspects which are content helpers (e.g. graphical conversion between pixels and physical units) are implemented in JavaScript. In both cases it is possible to extend the API. However, only toolkit developers who are able or willing to edit and re-compile the toolkit are able to add new 'C# API' methods by creating a new class which implements the `IRequest` interface (Section 5.2.3.2). Non-toolkit developers are able to write additional common modules or use 3rd party web libraries.

5.4 Chapter Summary

This chapter presents the software architecture and implementation of a toolkit that supports the rapid development of interactive projected displays. To use the

5.4 Chapter Summary

toolkit, three hardware components are required: a depth sensor, a projector, and a mid-range desktop PC. The toolkit software is divided into two main components:

- *Toolkit Application Software*: Interface for configuring and managing display deployments. Implemented using the WPF Framework, the Microsoft Kinect SDK, and the C# programming language. It can be extended by toolkit developers.
- *Toolkit Content*: The software developed by the toolkit users or content creators using web standards (circa 2012). Multiple items of content can be loaded and displayed by the toolkit application software simultaneously. Content is able to communicate with the *deployment model* directly via a *Content API*.

The main novel features of this architecture include: a decoupling of content from underlying application and operating platform, a named areas surface addressing scheme, a visual wizard based interface for creating and managing deployments, and a set of interaction modalities (including a novel multi-touch algorithm) implemented in JavaScript. Although there are multiple software architectures which could have met the requirements outlined in Chapter 4, the approach taken in this chapter focuses on simplifying the deployment process for toolkit users in ways which supports user innovation with interactive projected displays, for instance, using a visual interface to allow toolkit users to define surfaces by directly drawing them onto a live video stream.

In order to evaluate the toolkit implementation and validate its suitability for release, it is assessed in Chapter 6 through a series of controlled experiments and user studies. The lessons learned from these experiments are folded back into the toolkit in order to refine its design before release and adoption evaluation in Chapter 7.

Chapter 6. Toolkit Evaluation

6.1 Overview

The previous chapter described a software architecture and implementation of the requirements set out in Chapter 4. This chapter evaluates that implementation to determine the technical limits of the toolkit, in addition to studying the effectiveness of *toolkit users* using it to create applied interactive projected displays. This process validates that the toolkit is able to operate as intended and develops insights into the applications development process. The structure of this chapter is shown in Figure 84.

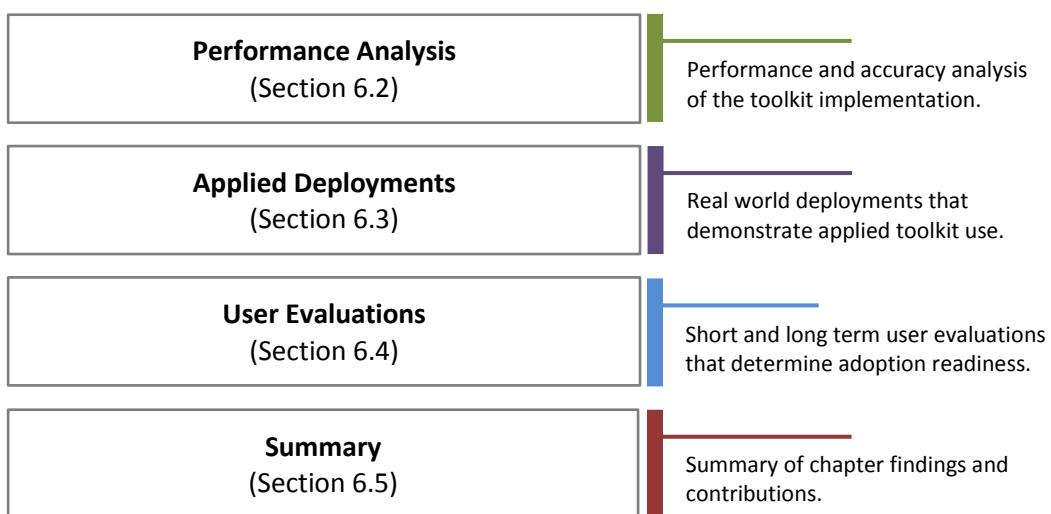


Figure 84: Structure of the toolkit evaluation chapter.

The evaluation is divided into three sections. Each section evaluates a different aspect of the implementation:

1. **Performance Analysis:** Quantitative evaluation of the accuracy and performance of the toolkit and compares it to other systems. This focuses on the touch interaction modality and toolkit operating

6.2 Performance Analysis

performance as these are likely to discourage adoption if not implemented to a high standard.

2. **Applied Deployments:** Evaluates feature completeness through a series of display deployments. These demonstrate the toolkit features working correctly in applied scenarios. Outputs are captured in a video that also helps to promote the toolkit.
3. **User Evaluations:** This furthers understanding of interactive projected display applications development and helps build confidence in the readiness for adoption. This is achieved through *short term* user studies of toolkit use, and deployment in a *long term* 3rd party project.

A strength of this approach is that it uses different methodologies to evaluate the toolkit from different perspectives. These findings inform academic understanding whilst generating confidence that it is ready for real adoption and public use. Usability issues identified in this chapter are addressed before release. As this evaluation does not study community adoption, Chapter 7 is dedicated to reporting and reflecting the use and application of the toolkit in the wild.

6.2 Performance Analysis

This section focuses on evaluating the accuracy of the touch interaction and content performance. These are important to evaluate as poor implementations could restrict the utility of the toolkit and thus adoption. Furthermore, the toolkit needs to be robust to various physical hardware configurations, including different sensor and projector positions and angles.

All analysis and evaluation conducted in this section used an Intel Core i5 2500K (3.30Ghz) PC with 4GB of RAM running the Windows 7 (64 Bit) operating system, a Microsoft Kinect™ for Windows, and a top-mounted short throw projector (InFocus IN1503) with a native resolution of 1280x800 pixels. This hardware was chosen as it is within the hardware parameters described in Section 5.2.2.

6.2 Performance Analysis

6.2.1 Touch Accuracy

The research question for this section is: how does touch accuracy vary with different angles and distances between the depth camera and interaction surface? The touch detection method used in the toolkit (described in Section 5.3.2) was evaluated and findings are compared with a capacitive touch screen to place them in context.

6.2.1.1 Methodology

To profile the accuracy of the multi-touch interaction algorithm a total of 30 accuracy samples were obtained over a range of angles and distances within the operating range of the Microsoft Kinect™. To measure ‘accuracy error’, the distance between the on-screen target (a 1cm² circle) and the computed touch position was recorded. To reliably vary angle and distance the Kinect™ was fixed to a pivoting boom. This boom was attached to a table as shown in Figure 85.

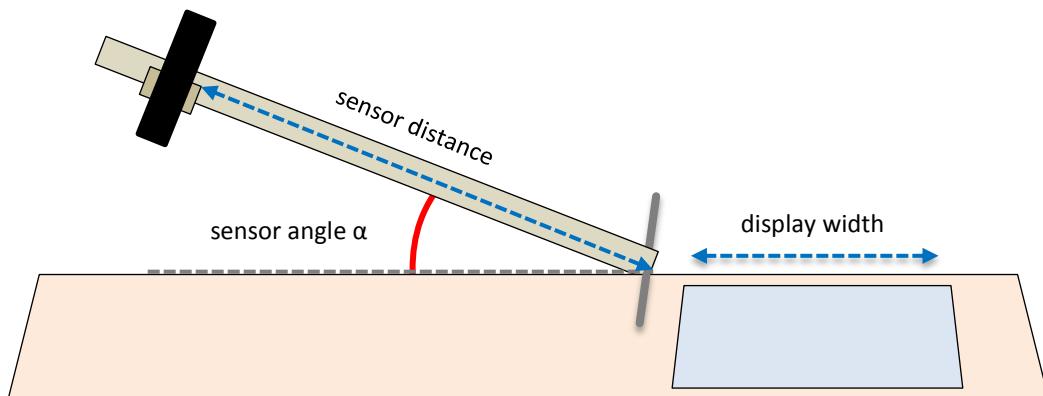


Figure 85: Hardware configuration used for accuracy profiling.

To obtain an accuracy sample, a single researcher (the author of this thesis) touched a projected target 100 times. Following each touch, the target would disappear and then re-appear 500ms later in a different randomised position. The use of randomising positioning as opposed to a repeating grid minimises sampling error resulting from sensor artefacts [37]. A drawback of measuring accuracy error this way is that it encounters variance due to user error. However, this is reduced

6.2 Performance Analysis

through (1) a practice period to counterbalance treatments, and (2), increased statistical power through a high number of touched targets per accuracy sample.

6.2.1.2 Distance and Angle Implications for Accuracy

Figure 86 visualises the relationship between sensor distance, sensor angle, and touch accuracy. The graph shows that accuracy is a function of sensor distance and is largely independent from sensor angle. Up to approximately 1.3m the multi-touch algorithm is able to operate over most angles and distances with an accuracy error of $\approx 5\pm 2\text{mm}$.

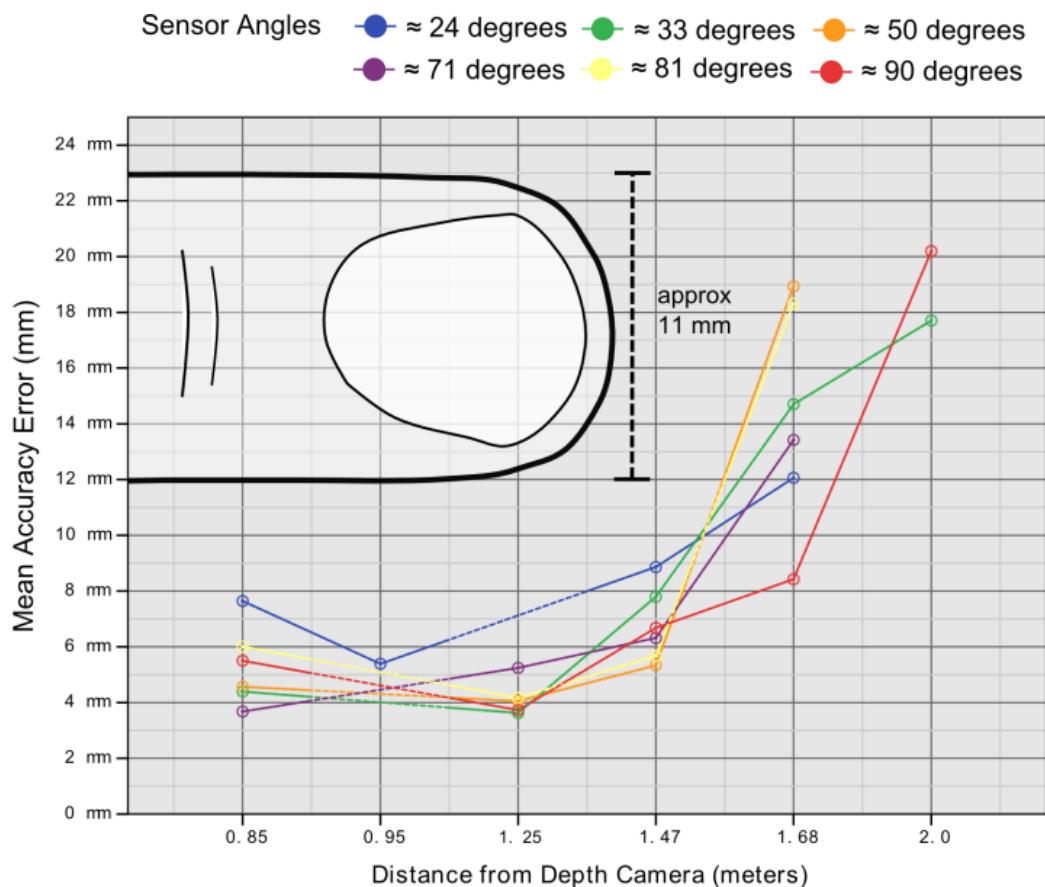


Figure 86: Graph to show the mean touch accuracy error (y-axis) separated by angle (colours) and distance (x-axis) between depth camera and interaction surface. Dashed lines show interpolated measurements.

Performance degrades faster as distance increases past 1.4m: the error can be sometimes as much as 2cm. This is approximately double the width of a typical adult

6.2 Performance Analysis

finger [167]). As the variance reported in Figure 87 suggests, this makes it very difficult to use multi-touch interaction for precise operation under circumstances where the sensor is further away.

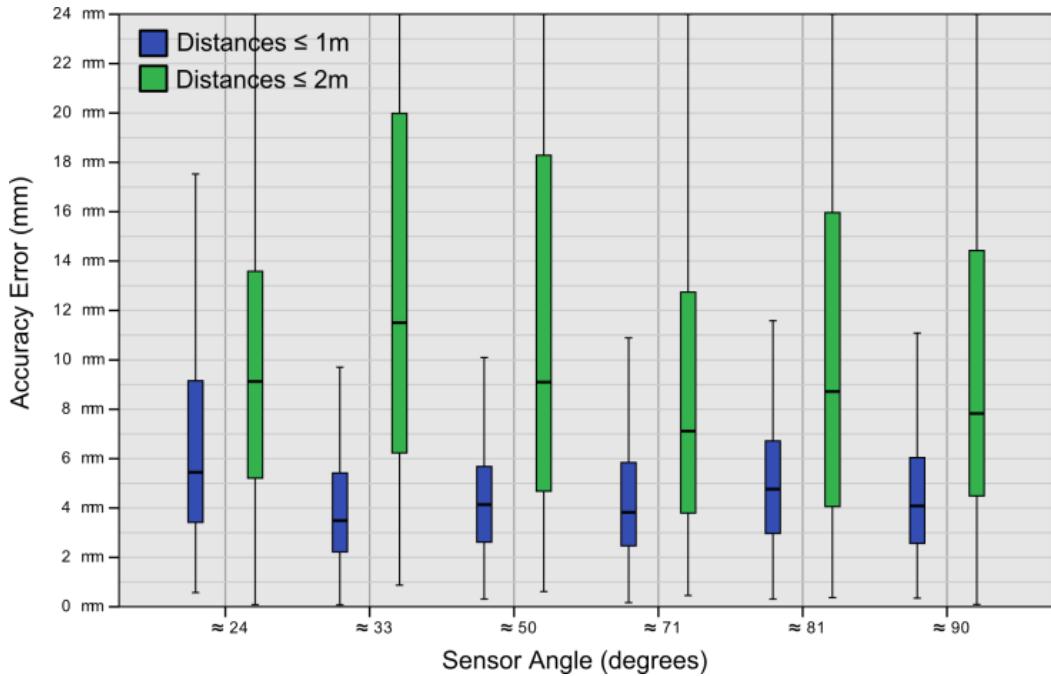


Figure 87: Shows the variance in ‘accuracy error’ by angle, grouped by exclusive sensor-to-interaction surface distances of $\leq 1\text{m}$ and $\leq 2\text{m}$.

While touch accuracy does not degrade with angle, calibrating the system and drawing displays at particularly acute angles ($\lesssim 21$ degrees) can be problematic on a video feed. At these angles, a surface 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 surfaces is easier.

6.2.1.3 Sensor Resolution Implications for Touch Detection

Although accuracy can be improved simply by moving the sensor closer to the target surface, examining why accuracy degrades reveals findings with more general implications for the use of depth cameras as touch sensors.

The Microsoft Kinect™ uses structured light algorithms to recognise a projected IR dot pattern in an image [37]. These values are then adjusted by the perspective

6.2 Performance Analysis

matrix of sensor to yield a 3D point cloud; each point representing a pixel. The further away from the centre of the projection, the sparser the point cloud becomes, thus objects closer to the sensor have more detail (Figure 88).

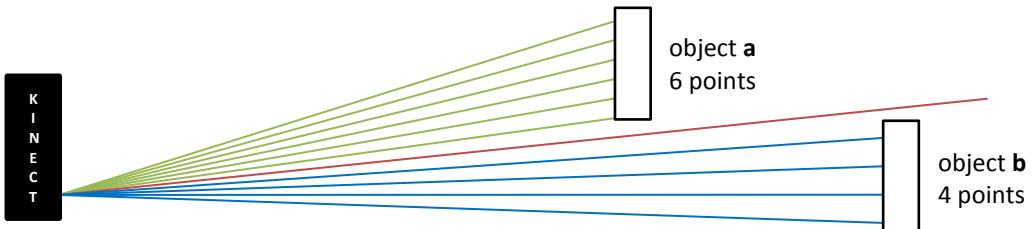


Figure 88: Illustrating why resolution decreases with distance from the sensor. Note 'A' has 6 intersections and 'B' has 4.

To examine the effects of decreasing resolution on the multi-touch detection algorithm, the toolkit was used to create a large interactive surface (90x26cm). This size was chosen to cover the distances where accuracy degraded most quickly (based on the findings in Figure 86). The raw point-cloud data used to form coherent touch points was measured for over 1,000 touch events in randomised positions along this surface. For each touch the mean number of data-points and standard deviation was computed.

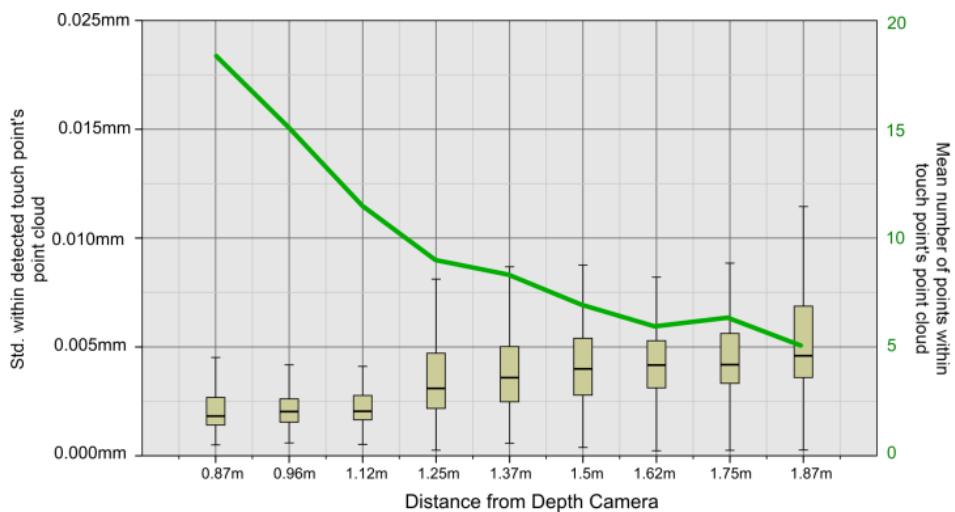


Figure 89: Showing standard deviation within a touch's point cloud increasing over distance (box plots), while the number of points in the same cloud decreases (line)

6.2 Performance Analysis

The results in Figure 89 show how deteriorating point cloud resolution impacts the algorithms ability to form a coherent touch point. The box plots (left axis) show that the mean standard deviation within each point cloud (representing a finger) increases with distance to the sensor. That is to say, the further away a touch is from the sensor, the more spread out its point cloud becomes. The line-graph (right axis) shows the average number of points used to identify a touch decreasing as distance from the sensor increases. The combination of a larger and less-dense touch point makes it considerably harder to assemble a coherent touch point.

The implications for deployments 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 89) 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. However, a person configuring the space can optimise its placement if they know where most interaction will take place. To help *toolkit users* achieve optimum accuracy for their deployments it may be prudent to offer a sensor placement efficiency measure for each surface.

6.2.1.4 Comparison to Capacitive Touch Screen

To compare the accuracy error of the toolkit to that of a capacitive touch screen, the same procedure used in Section 6.2.1.1 was used to generate an accuracy sample for a capacitive touch screen⁵⁶. The results in Figure 90 show that the capacitive surface exhibited less accuracy error ($\mu=1.0\text{mm}$, $\sigma=0.7\text{mm}$) than the depth camera ($\mu=4.5\text{mm}$, $\sigma=2.7\text{mm}$). In the context of a typical finger a variance of $\approx 5\text{mm}$ is acceptable for coarse general purpose interaction. Cross referencing this with qualitative findings (see later in Section 6.4.1), all participants indicated that the accuracy and speed of the multi-touch system was good enough to support their application.

⁵⁶ HP TouchSmartTM2: en.wikipedia.org/wiki/HP_TouchSmart#TouchSmart_tm2

6.2 Performance Analysis

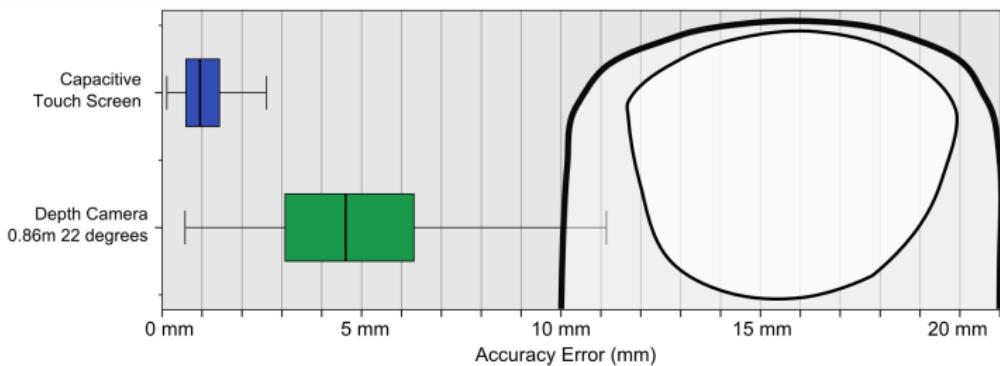


Figure 90: A comparison of capacitive and optical (depth camera) touch accuracy

6.2.2 Content Performance Profile

Performance was examined in three different ways and its response recorded. These were: (1) performance with an increasing number of touch points, (2) performance with an increasing number of displays rendering video, and (3) performance of two content items whilst one is under load. These experiments were chosen to evaluate the different parts of the architecture essential for smooth user interaction⁵⁷.

6.2.2.1 Touch Detection Performance

To measure FPS performance and algorithm execution time with increasing numbers of touch points, a total of 4,650 FPS samples were taken as the number of touch points varied. The variance in the number of samples per touch point count was ($\mu=354$, $\sigma=81$). To consistently vary the number of touch points, a number of marker-pen tops were used as analogues for human fingers and placed on top of the interactive surface being tested.

As shown in Figure 91, the toolkit offered acceptable performance under the measured conditions. When two fingers are present, a single frame of the touch detection takes approximately 5ms to fully complete processing. This increases to 25ms with 10 or more fingers. The increase is linear and plateaus after 11 touch

⁵⁷ These are content logic (CPU intensive), toolkit content host (multi-process intensive), and content animation (rendering intensive).

6.2 Performance Analysis

points as the number of point-cloud data points being sent to the JavaScript content reaches a maximum (see `point_limit` field in Section 5.3.2, Figure 81). It should be noted that this performance could be improved by implementing the multi-touch algorithm in the toolkit (.NET) rather than JavaScript hosted by the content. This would minimise data transfer (between the toolkit and JavaScript content) and perform the data processing using specialised language features, but would sacrifice high level control over the data processing.

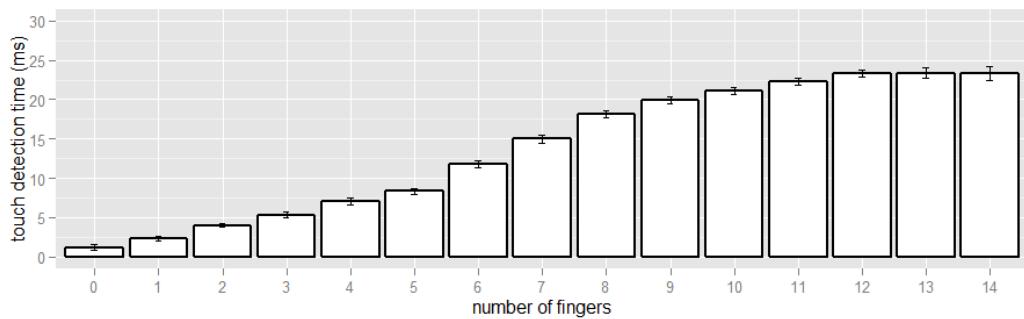


Figure 91: Time taken to perform touch detection (ms per frame) increases with the number of fingers.

6.2.2.2 Graceful Degradation

To measure system performance under increasing general load the number of individual displays rendering video content was increased whilst logging the FPS. The logging took place for a period of 5 minutes, during which time, 7 additional displays were added. Each added display was given a letter that corresponds to a series in Figure 92. A total of 2,541 samples were recorded.

The results show that, as expected, adding more displays increased the overall processing demand on the PC and the corresponding FPS reduction was spread across all the content displays. Figure 92 shows that the performance of individual content items degraded gracefully in unison. This shows that items of content are treated equally (i.e. the load did not fall on any one content item) and confirms that the multi-process architecture works correctly.

6.2 Performance Analysis

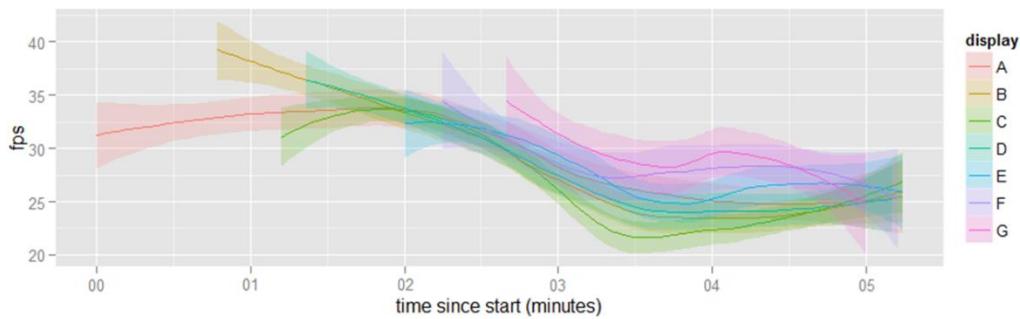


Figure 92: Display content performance degrades gracefully in unison as more system load is applied.

6.2.2.3 Content Performance Independence

To demonstrate performance independence of the content items, two items of identical display content were deployed: *normal* and *touches*. This demonstrated that additional CPU load applied to one content item does not transfer to others. Both items rendered the same video, but the *touches* content was made more processor intensive by performing touch detection (10 fingers). The results (Figure 93) show that the frame rate of the *normal* display did not vary with the number of touches applied to the *touches* display. This shows that content item performance is independent.

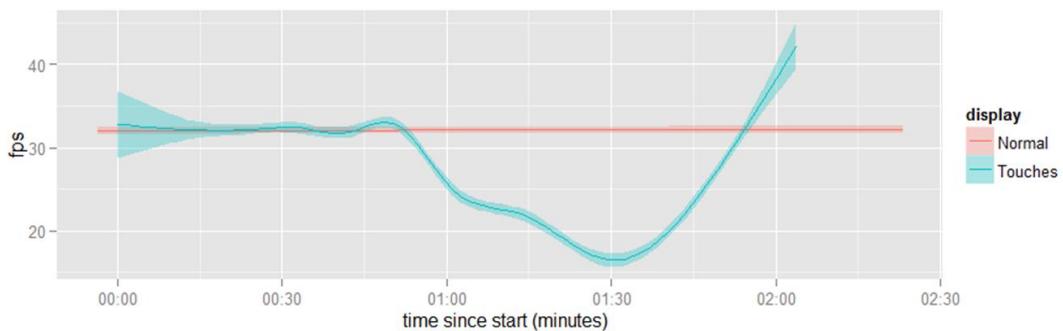


Figure 93: Two items of content demonstrating performance independence. “Normal” (red) remains at a consistent frame rate as processor load is applied to “Touches” (blue) causing its frame rate to vary.

6.2.3 Summary

The multi-touch algorithm is able to offer interaction over a range of angles and distances—and thus fulfilling (Requirement 20: Robust Hardware Placements) and

6.3 Applied Deployments

identifying operating limits. These are largely invariant of sensor angle and define the distances at which the algorithm is no longer able to function effectively with the Microsoft Kinect™. Following Dippon et al. [85] we recommend that developers avoid creating targets smaller than a finger. To assist with this, the toolkit Content API can access the surface dimensions so developers can convert between pixels and meters (Section 5.3.5). These findings also demonstrate that it is possible to achieve effective interaction without the individual calibration of each surface (Requirement 6: Minimise Calibration and Requirement 5: Rapid Configuration).

Content performance profiles confirm that the toolkit degrades gracefully and the performance of processor intensive content has minimal impact on other concurrently executing content (Requirement 4: Graceful Error Handling and Degradation). Both the touch accuracy and content performance profiles can be shared with a community of *toolkit users* to help them plan and troubleshoot their deployments.

6.3 Applied Deployments

This section presents eight applied display deployments that demonstrate the toolkit features work together correctly. The deployments show how the toolkit requirements map to applied interactive projected displays across different scenarios and contexts (i.e. domestic, office, commercial, etc).

Table 13 lists and describes the eight display deployments. Each deployment took an expert toolkit user (the author of the thesis) less than one hour to code and deploy. The deployments ran for approximately one hour each, during which time interactions were filmed. These were compiled into a video used to promote the toolkit and seed the community with ideas: <http://youtu.be/dflN07MoAUY>. Since its release on YouTube (Dec 2012) it has accumulated over 43,000+ views and been featured in the UK mainstream press. All but two requirements are covered, with the exception of (Requirement 1: Lower Skill Barriers) and (Requirement 3: Simple Abstractions) that specifically focus on support for non-expert users.

6.3 Applied Deployments

The hardware used was an i5 Laptop, a Dell M110⁵⁸ portable projector, and a Microsoft Kinect™ for Windows. In almost all cases the Kinect™ and projector were mounted in a convenient location using a Joby GorillaPod⁵⁹ by attaching them to nearby furniture.

Table 13: Selection of display deployments demonstrating the toolkit feature set.

Name and Picture	Description
Bed Display 	Intended to assist people in their morning routine, the bed display features a multi-touch menu where the user can select: lighting controls, headlines, or off. When the lighting mode is displayed (pictured) each wooden slat in the bed acts as a light switch. Touching a slat causes it to glow and sends a message to a web-socket connected micro-controller. This switches a relay to turn the light on or off. When the headline mode is selected, the headlines of the day are projected along the top of the bed (pictured). When not in use, the controls disappear. This was created using three content items deployed across five surfaces (slat1, slat2, slat3, top, left_side). This demonstrates interoperation with external systems (<i>Requirement 22</i>), projection mapping on non-standard shapes (<i>Requirement 9</i> , <i>Requirement 7</i>), multiple displays (<i>Requirement 10</i>), and inter-display communication (<i>Requirement 13</i> , <i>Requirement 16</i>).

⁵⁸ Dell M110 Portable Projector: <http://www.dell.com/ed/business/p/dell-m110/pd>

⁵⁹ Joby GorillaPod Tripod: <http://joby.com/gorillapod>

6.3 Applied Deployments

Water Usage Display 	<p>Intended to communicate real-time water usage to users during their ablutions. Uses easy-to-relate-to visualisation rather than abstract numerical quantities.</p> <p>Lacking a water sensor, this was the only display to use wizard-of-oz-techniques. When the wall is touched, the display appears and water fills the jugs at a constant rate, estimated based on water flow.</p> <p>This was created using a single content item and animations used the jQuery library.</p> <p>Given this display was deployed in a bathroom, it was important to place the projection hardware in a safe location. Rapid configuration (<i>Requirement 5</i>) and projection mapping (<i>Requirement 9</i>) were important in achieving this.</p>
Floor Display 	<p>Intended to create an engaging floor display based on Michael Jackson's Billie Jean music video.</p> <p>The design adopts the existing architectural aesthetics by mapping glowing squares to floor tiles. When users steps on a floor tile, it glows. When they move off it, the glow disappears.</p> <p>To create the system, the same item of content (floorswitch.html) was deployed on 8 different surfaces mapped to floor tiles.</p> <p>The large number of display created for this made not calibrating each tile individually useful (<i>Requirement 6</i>, <i>Requirement 5</i>). This was deployed in a public place (<i>Requirement 21</i>) as a literal walk-up-and-use display (<i>Requirement 15</i>) that needed to support multiple concurrent users (<i>Requirement 14</i>).</p>

6.3 Applied Deployments

Door Display 	<p>Intended to ‘invite’ users to interact with a door display when the occupant of a room is out. This display appears when a user approaches. This indicates that the display is for the user to interact with. It shows the name of the room’s occupant and a map to their location (Google Latitude). To create this system, a single item of content (door.html) used a presence trigger and a multi-touch detector.</p> <p>This display integrated two interaction modalities based on the state of the user (<i>Requirement 12, Requirement 11, Requirement 18</i>) and made use of external web standards and 3rd party systems (<i>Requirement 17, Requirement 22</i>). This was all managed using programmable content (<i>Requirement 13</i>). Being able to project on the door without impacting the surrounding space was achieved by placing the hardware on the floor away from traffic in the corridor (<i>Requirement 20, Requirement 9</i>).</p>
Restaurant Menu 	<p>Intended to add novelty to a restaurant dining scenario⁶⁰ by placing menu ‘specials’ on an interactive menu.</p> <p>This display is placed by the side of a table and users interact by touching the display and sliding the images left and right.</p> <p>It was created using a single item of content (menu.html) and used a 3rd party library iScroll⁶¹ to provide smooth scrolling (<i>Requirement 22</i>).</p> <p>The public nature of the deployment meant that it had to be set up discretely and quickly to avoid negatively impacting the on-going business (<i>Requirement 2, Requirement 5</i>).</p>

⁶⁰ Thanks to Wibbly Wobbly Burger Bar at Lancaster University.

⁶¹ iScroll library: <http://cubiq.org/iscroll-4>

6.3 Applied Deployments

Pervasive Advertising 	<p>Intended to demonstrate how advertising material could appear ‘everywhere’, including places where it would be difficult for standard display form-factors to reach. Although this use-case has a simplistic design, it demonstrates a range of display form factors. To create this display, the (menu.html) content item was re-used.</p> <p>This was deployed discretely in a public place (<i>Requirement 21, Requirement 20</i>) and used projection mapping to create the irregular display shape (<i>Requirement 7, Requirement 9</i>). Rich media standards are important for advertising scenarios that attract attention (<i>Requirement 17</i>) [114].</p>
Peripheral Office Screen 	<p>Intended to give desktop PC users a specialist auxiliary display that can be used for video communication and news reading.</p> <p>A projection is mapped to a piece of wood (240x190mm) placed adjacent to a monitor. Placing objects on post-it-notes controls the content (or lack of content) on the display.</p> <p>To create this system, 5 items of content were created: ustream.html, news.html, tog_mute.html, tog_ustream.html, and tog_news.html. Ustream⁶² was used for web-based video streaming (<i>Requirement 22</i>). This was processor intensive. To ensure low performance delay on the interaction independent content performance was important (<i>Requirement 4</i>).</p> <p>This deployment makes use of tangible/object based interaction rather than touch—making use of the decoupling between interaction modality and platform (<i>Requirement 18</i>). This used multiple displays (<i>Requirement 10</i>), inter-display communication (<i>Requirement 16</i>), projection mapping for the non-planar projection onto the wood (<i>Requirement 9</i>), and different display sizes to indicate a difference between input and outputs (<i>Requirement 7</i>).</p>

⁶² UStream online video streaming web service: <http://www.ustream.tv/>

6.3 Applied Deployments

<p>Cookery Assistant</p>	<p>Intended to assist students with the cooking process. Two surfaces are created: one on the counter-top and another on the wall behind the oven hobs.</p> <p>When a user places an object (e.g. a packet of pasta) on the counter, the counter top display suggests recipes based on a pre-set list. This could be enhanced with object detection. When one is selected, the chosen recipe jumps from the counter-top surface to the surface on the wall; showing a video of the necessary cookery steps.</p> <p>This deployment used multiple displays (<i>Requirement 10</i>) projected onto two different surface planes (<i>Requirement 9</i>). Programmable content (<i>Requirement 13, Requirement 16</i>) enabled an animated fade out (<i>Requirement 17</i>) before the content swapped surface (<i>Requirement 16, Requirement 8</i>).</p>
--------------------------	--

The deployments demonstrate that the toolkit can operate effectively to produce a variety of interactive projected displays. The deployments identified a number of bugs with the toolkit Content API, and as these were relatively minor issues (i.e. bad parameter naming), they were fixed in place. A high level of re-use was possible with more ‘abstract’ display content such as presence detectors and video players. For instance, a floor switch was easy to convert into an object detector for a kitchen countertop. Standardised naming conventions for functions helped improve the speed of developing multi-display applications.

Reflecting on the process of developing and conducting these deployments, robust hardware placement (*Requirement 20: Robust Hardware Placements*) and projection mapping (*Requirement 9: Projection Mapping*) were particularly useful features. Previously, installing a projector might have required a special rig or mount point. However, with the toolkit it was possible to place the hardware in a safe and convenient location, and still create displays in lots of different places very quickly and with minimal disruption. This benefit was particularly noticeable in the

6.4 User Evaluations

restaurant and foyer environments as business was able to continue normally while the deployment was set up and conducted.

6.4 User Evaluations

This section evaluates the suitability of the toolkit to be adopted through a two *toolkit user* studies. These are:

1. ***Short Term***: 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 developed an application of their own design.
2. ***Long Term***: The second study took place over *four months* and was designed to assess the toolkit in terms of feedback on how well the toolkit helped the project staff achieve their requirements.

The findings confirm users are able to operate the toolkit effectively, identify areas for improvement, and highlight considerations for applications development with interactive projected displays.

6.4.1 Short Term Evaluation

The research question for the short-term lab study asked: can users with web programming experience operate the toolkit given minimal training? This was chosen to provide insight into the levels of effort required to use the toolkit, and identify the parts of the applications development process where that effort exists. As with the other evaluations, it helps to build confidence in the readiness for adoption.

6.4 User Evaluations

6.4.1.1 *Methodology*

This study is divided into two parts: (1) a prescriptive task that familiarises them with the toolkit, and (2) an open-ended task to develop an interactive projected display of their choice.

In **Part I** participants followed a simple tutorial and created two example display applications. The tutorial guides them through using the main features of the toolkit, whilst the example display applications familiarise them with the HTML, CSS, and JavaScript required to develop toolkit content.

In **Part II** participants were given freedom to design and program their own (relatively complex) display deployment. If they could not decide on a creative design, they had the option to choose one from a list of three suggestions: (1) A mechanism for transferring content between a display and a mobile device. (2) An interactive video viewer with separate control panel. (3) A ‘shape mixer’ whereby a user selects a colour on one display and a shape on another so that the combined shape and colour was shown on another display. These cover a range of the toolkits capabilities and ensure a build complexity that is non-trivial for the available allotted time.

Combined, Parts I and II lasted on average two hours per-participant. As participants progressed through the study, they were asked to fill out a questionnaire to capture levels of understanding on Likert scales. These questions assessed their understanding of the steps they were following and included space for them to suggest improvements.

6.4.1.2 *Participants*

Eight participants undertook the study. Of these, 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™. Although eight participants is a relatively small sample size, the study was exploratory and designed to reveal insights into the effort required to use the

6.4 User Evaluations

toolkit, rather than to make assertions about interactive projected displays in general. Experience with web programming was part of the participant selection process so that they might fully exercise the toolkit's ability to create content.

6.4.1.3 Study Part I – Tutorial and Sample Applications

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 that 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 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.

6.4 User Evaluations

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 (P2) 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?

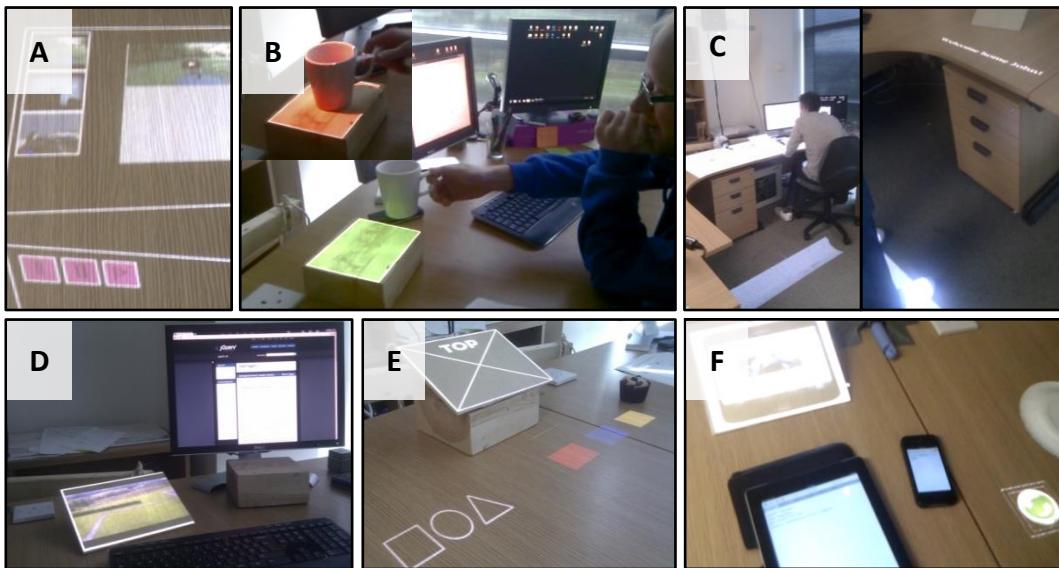
6.4.1.4 Study Part II - Involved Development

In the open ended development task, half of the participants opted to design their own display rather than take one of the three choices. Their project names and a selection of photographs featuring the systems they created are provided in Figure 94.

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.

6.4 User Evaluations

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.



List of participant projects developed:

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 94: Projects developed by participants during Part II of the short term user study.

Overall Perceptions

Almost all participants reported that they enjoyed using the toolkit and expressed 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, it is not to suggest that the toolkit’s design was the source of this enjoyment, but rather the creativity and novel

6.4 User Evaluations

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.

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 a description language which combines the desirable features of both could be used to help 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

6.4 User Evaluations

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.

Developing for the Physical World

All but two participants heavily tested and debugged their systems in the physical world (i.e. deployed on a surface) rather than in an on-screen browser. In that sense, the ability to interactively place content onto target areas helped to emphasise the relationship with the physical environment. Without this relationship, participants would have been forced to imagine how their application would behave before deployment. Although far from a complete solution, the toolkit and the abstractions offered may be a considered what Abowd referred to as a *programming environment for programming physical environments* [16].

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’

6.4 User Evaluations

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 posing 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.

Suitability for Rapid Prototyping

In terms of the toolkit’s ability to facilitate rapid prototyping, integrating open source libraries and samples was easily achieved. P4 used an online webcam streaming service to construct a desk security camera for her peripheral vision. 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.

6.4.2 Long Term Evaluation

The research question for the long-term user evaluation asked: is the toolkit suitable for adoption and continued use in a long term project? Long term use was investigated in conjunction with an external application driven research project. The findings report (1) the extent to which the toolkit satisfied the project requirements and (2) feedback from the toolkit users covering four months of use.

6.4.2.1 *Setting*

The toolkit was given to an application driven research project investigating how novel and engaging displays can improve feedback quality about public spaces. They used the toolkit to construct a large ($\sim 2\text{m}^2$) interactive floor display (Figure

6.4 User Evaluations

95) capable of recognising the specific areas people walk over (visual representations of buildings on a 2D map). This triggers the display of related content on separate co-located display.



Figure 95: A prototype of the deployment used in the long term project.

6.4.2.2 Findings

The investigator responsible for the project highlighted that the easy setup and deployment process let them progress with their interests rather than focusing on the technical aspects of interaction sensing. Further, 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 that 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

6.4 User Evaluations

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.

6.4.3 Discussion

This section reflects on how the user evaluations informed improvements to toolkit features, how the toolkit implementation supports *toolkit users* in the creation and deployment of content, and discusses findings that improve our understanding of applications development for interactive projected displays.

6.4.3.1 Toolkit User Abstractions

To help simplify the process of development and deployment it was important the abstractions (Surface and Display, Section 5.2.1) were easy to understand and work with. The responses showed that all the participants understood the abstractions and were able to use them effectively. However, due to the skill range of participants (i.e. most had web programming experience) it is difficult to infer if these transfer to non-programmers without further study (see Chapter 7).

Exposing the Surface and Display abstractions 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 opened up many possibilities for content automation methods and creative applications—as demonstrated by the participants. However, it also exposed flaws which need to be addressed with moderation policies, such as the P2’s ‘content homing device’.

6.4 User Evaluations

6.4.3.2 Located Code

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. It could be argued that a mental model and interface design that associates logic with a physical space is advantageous, as it reminds users that digital logic is present in physical locations. 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. A computer game 'level editor'-style approach 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.

6.4.3.3 Touch Accuracy

To help address lower accuracy at larger sensor distances, it is recommended that developers increase the size of interactive controls to suit the accuracy profile provided in the system evaluation section. However, not all participant-created deployments lent themselves to a 'push-button' or touch-screen based interaction design. For instance, the glowing wooden slats on the bed display (Section 6.2) required less interaction precision than touch event detection, and so could be deployed further away from the depth camera. The availability of simple interaction techniques like presence detection may encourage developers to experiment with designs that leverage physical shapes and aesthetics already present in the space.

6.5 Chapter Summary

6.4.3.4 Limitations of Web Standards

Most of the participants agreed that the use of web standards were not limiting in terms of what could be created. However, given the relatively small sample size and the exploratory nature of the study (i.e. the users are not aware of anything different) it is difficult to draw conclusions from these findings. However, it is encouraging that the participants were able to combine their ideas with more advanced web technologies in order to make them interoperate with other systems. Further evaluation is required in order to be able to assess if web standards are enough or more features are required.

6.4.3.5 User Interface Adjustments

We asked participants from both user evaluations what the three most important aspects they would change or improve were. Discounting minor user interface tweaks, the most common responses were: more documentation about the range of toolkit API features, a detachable debug log, and an HTML element inspector/debugger. Furthermore, following P3's experience deploying surfaces over two planes, the floor and table we would recommend an additional advanced display management mode to give *toolkit users* more precise control over how they positioned surfaces. This would make it easier to align data from multiple depth cameras and projectors.

6.5 Chapter Summary

This chapter evaluates the toolkit implementation in three ways. It analyses performance and accuracy to determine an effective operating range, demonstrates that toolkit features work together effectively through a series of 8 deployments, and studies *toolkit users* creating applications that use interactive projected displays.

6.5 Chapter Summary

The findings show the toolkit is able to operate under a range of hardware placement conditions and identifies the ways hardware limitations impact on the interaction modalities that can be used. It compares the performance of the multi-touch interaction algorithm to a typical capacitive touch screen; concluding that while the toolkit is less accurate sensing than dedicated hardware it is accurate enough for the concepts developed by participants of the user evaluation.

Eight sample display applications were created and deployed that demonstrate use in domestic, office, and commercial contexts. These show how a range of toolkit features map back to the requirements and validate that these features work together correctly. However, as these were created by an expert toolkit user they do not build confidence that the toolkit is suitable for adoption.

To address this, user evaluations support the position that *toolkit users* are able to operate the toolkit effectively. The user evaluations also serve to identify areas for improvement (i.e. UI design) and highlight considerations for applications development with interactive projected displays (i.e. developing and debugging in physical spaces). Factors such as familiarity with web standards and programmable content are important to make the most of its capabilities. It is also important to properly communicate the capabilities to would-be *toolkit users* so they are able to take full advantage of the features it provides.

In summary, the toolkit achieves its goal of being a simple to use method of rapidly creating interactive projected displays. This is evidenced by all the *toolkit users* reporting that they were able to create a diverse range of applications very similar to how they were envisioned. This signals approval that the toolkit is ready to be publically released following minor usability corrections and amendments. In terms of limitations, evaluation with *toolkit users* is exploratory and reliant on observational methodology. Studying toolkit use first-hand in this way is inherently time-consuming and thus imposes practical limits on the sample size. With that in mind, care must be taken to generalise the findings. Additionally, while this approach generates insight into the value of requirements defined in Chapter 4, it does not reflect on if these were correct requirements. The next chapter studies a years' worth toolkit adoption and usage in real applications around the globe.

Chapter 7. Toolkit Adoption

7.1 Overview

The previous chapter presented an evaluation of the toolkit and discussed how its features can support *toolkit users*. However, it remains to be seen if the toolkit can support the application scenarios of *toolkit users* in the real-world. If so, what can be learned from these application scenarios that feeds back into the general academic knowledge and design of interactive projected displays? To answer these questions, this chapter provides an analysis of over one year of toolkit usage data following the toolkit release. The structure of this chapter is shown in Figure 96.

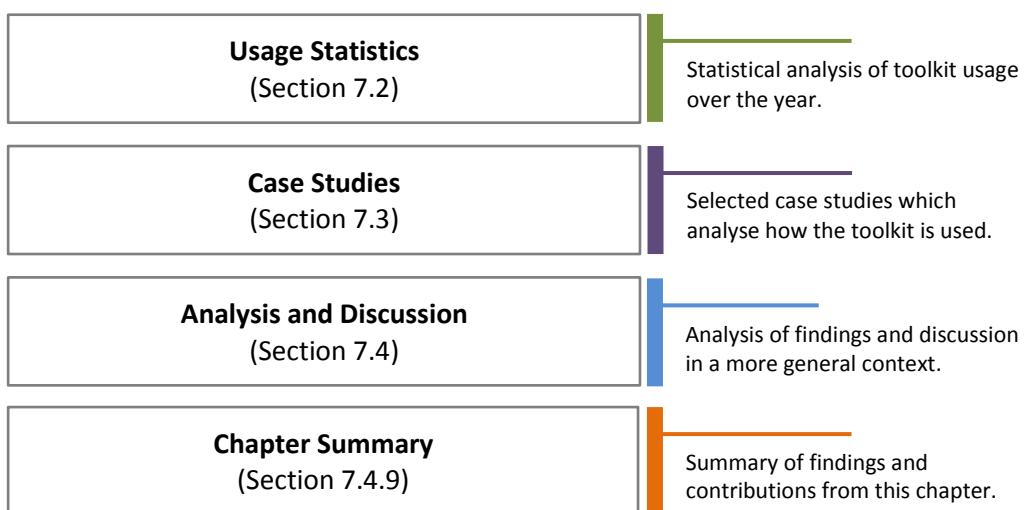


Figure 96: Structure of toolkit usage chapter.

Section 7.2 provides summary statistics that describe the quantitative impact of the toolkit. Section 7.3 follows these with a set of case-studies from selected projects that provide additional depth. All of the findings from the usage data and case studies are collectively discussed in Section 7.4. This focuses on supporting the different *toolkit user* groups and the effectiveness of different toolkit features in the context of different toolkit adopter groups.

7.2 Usage Statistics

This section provides a quantitative summary of the toolkit's usage over the period of a year since its public release. The findings presented in this section demonstrate the extent of adoption and impact. It considers download figures, usage statistics, error reports, and correspondence volume. Analysis of these findings identifies areas of strength, weakness, and where *toolkit user* expectations do not match the toolkit design.

7.2.1 Data Collection

The toolkit was released to the public on **6th December 2012** alongside the presentation of the paper "*Toolkit Support for Interactive Projected Displays*" at the MUM2012 conference held in Ulm, Germany. The data collection ran from this date up until the **28th December 2013**.

To study the toolkit adoption, eight indicators are measured:

1. *Downloads*: The total number of toolkit installer downloads.
2. *Website Hits*: The total number of website hits on the Google Code project website.
3. *YouTube Views*: The total number of times the promotional and tutorial videos were viewed.
4. *Toolkit Uses*: The total number of times the toolkit was executed on internet connected computers.
5. *Issues Reported*: The total number and type of the issues reported on the Google Code issue tracker.
6. *Forum Usage*: The total number of people and usage statistics of the community support forum.
7. *Personal Correspondence*: The volume and type of personal correspondence (support, advice, requests) that did not come through the public support forum or Google Code issue tracker.

7.2 Usage Statistics

8. *Research Form:* An online-web form (that *toolkit users* could optionally fill out) was built into the toolkit application and accessible through an icon in the computer interface.

These indicators provide different measures of *toolkit user* interaction with the toolkit. Naturally, there is likely to be a certain amount of error in each measure. For instance, the toolkit may be shared between toolkit users via USB stick rather than direct download.

A research form was included in the toolkit applications user interface along with a web-link to the toolkit support forum and showcase. As the research form was stored online, starting the toolkit application would access its URL which would cause the server to log basic usage information via a URL resolution service⁶³. An online community support forum and personal correspondence (i.e. email) provide another way to capture toolkit usage in more depth. All of the correspondence was coded and listed. In all instances, with the exception of those who gave explicit permission to be included in the thesis, *toolkit users* (and where appropriate, their projects) are presented anonymously.

7.2.2 Descriptive Summary

Table 14 presents a descriptive summary of the eight toolkit usage indicators. In total the toolkit was downloaded over 2,300 times. It has also been run over 21,000 times. The ratio between these figures is encouraging; suggesting that once downloaded it is used multiple times. This is discussed in more depth in Section 7.2.3. Of all the indicators, the research form provided the least insight, with only 6 forms being completed and submitted, 5 of which not contain data. This was perhaps due to the form lack of prominence in the user interface.

⁶³ Your Own URL Shortener: <http://yourls.org/>

7.2 Usage Statistics

Table 14: Summary of toolkit usage statistics between 6/12/12 and 28/12/13.

Downloads	Total Number of Toolkit Installer Downloads	
	2,381	
Website⁶⁴	Visits⁶⁵ 18,476	Unique Visitors 7,365
	Avg. Visit Duration 3m 37s	Bounce Rate 56.75%
YouTube	Promotional Video 36,744 (views) 181 like – 4 dislike	Tutorial Video 11,592 (views) 55 like – 0 dislike
Toolkit Uses	Total 21,942 records (99%) (excluding Lancaster and anonymous proxy)	
	Lancaster Only 203 records (0.92%)	Anonymous Proxy Only 17 records (0.08%)
Issues	Bugs and Defects 18 (12 invalid or duplicate)	Feature Requests 4 (0 invalid or duplicate)
Forum Usage beginning 30 th Jun 2013	Number of Posts 146	Number of Topics 27
	Number of Members 54	Posts per Day 0.80
Personal Correspondence	Email 70 people (details in Table 17)	
Research Form	Responses 6 total (1 valid, 5 invalid)	

⁶⁴ Figures are based on the Google Project page (code.google.com/p/ubidisplays) that links to the toolkit downloads and source code.

⁶⁵ Total number of visitors, excluding identifiable robots.

7.2 Usage Statistics

7.2.3 Usage Patterns

This section provides a deeper analysis of usage patterns and toolkit user habits, beginning with the analysis of toolkit usage data provided by the URL shortening service. The usage figures for Lancaster University (i.e. those by this author, accounting for 0.92% of recorded data) are not included. It focuses on addressing the following questions:

1. *Rate*: How often is the toolkit being used?
2. *Distribution*: How many people are using the toolkit?
3. *Period*: How often does an average user use the toolkit?
4. *Frequency*: How long does the average user use the toolkit for?

7.2.3.1 *Usage Rate*

To determine how often the toolkit is used, Figure 97 plots the daily usage data between the public release and last day of data capture as a time-series bar chart. It shows consistent toolkit use throughout the year. A linear correlation test shows that the number of daily uses tends to increase steadily with time, rather than decrease after an initial peak: $r(397) = .49, p < .001$. Usage rates do not increase monotonically, as per the ‘bulge’ between June 2013 and Dec 2013. The adoption and consistent usage over the period of one year evidences the claim that the toolkit is suitable and achieves its goal of supporting user innovation with interactive projected displays.

7.2 Usage Statistics

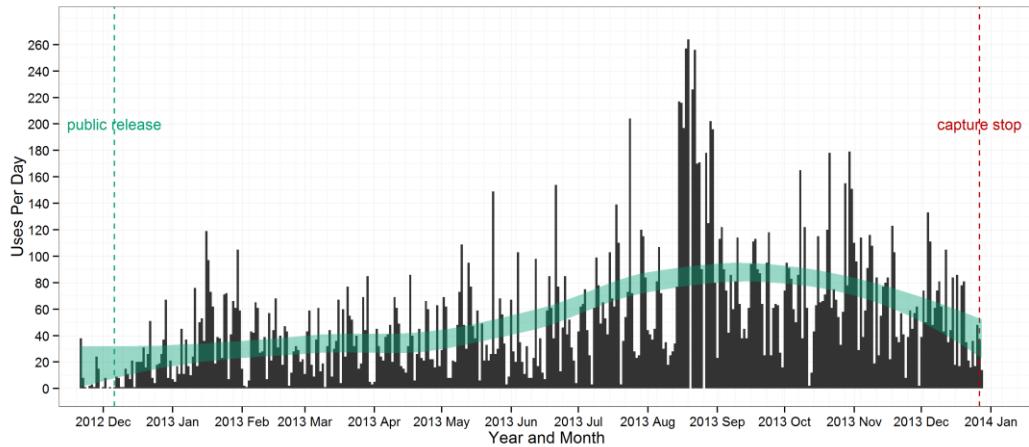


Figure 97: Daily toolkit usage data. Beginning 6th Dec 2012, ending 28th Dec 2013. A local polynomial regression model (loess smoothing) fit to the data is overlaid in green to illustrate usage trends.

7.2.3.2 Toolkit User Distribution

To determine an approximate figure for how many people are using the toolkit, the usage statistics are aggregated by IP address. This process makes the assumption that each IP address approximately represents an individual *toolkit user*. Although this method is not infallible (i.e. the toolkit may be used offline, particularly in longer term projects) it is considered suitable for studying trends and drawing conclusions on a summary basis. Ethically, it is important that this analysis does not describe behaviours in a way that threatens the anonymity of *toolkit users*. Subsequently, only summary data is presented and any GIS lookup accuracy is reduced⁶⁶ to the nearest city to account for time zones in toolkit usage analysis.

Beginning with summary statistics, Table 15 shows that there are an *estimated 2,119 individual toolkit users*. These users ran the toolkit 21,942 times in total ($M=10.35$, $SD=33.63$), with (73%) running it between 1 and 10 times and the remaining (27%) running it more than 11 times. To illustrate the geographic distribution, Table 15 lists the top countries and renders the IP address locations on a map of the world accurate to the nearest city. This shows worldwide adoption with

⁶⁶ GeoIP Lite Cities Database is 99.5% on a country level and 79% on a city level.
http://www.maxmind.com/en/geolite_city_accuracy

7.2 Usage Statistics

activity concentrating around major cities, as might be expected. There is a high concentration of adoption in and around Europe and North America, but also penetration into developing countries (i.e. BRIC).

Table 15: Summary statistics of users broken down by country. Image shows geographic localisation of toolkit user numbers overlaid onto a Google map. This is available as an interactive web application.

Summary Data			
Total Users		2,119 Users (excluding Lancs Uni and anonymous proxy IPs)	
Total Hits		21,942 Hits	
Country Specific Data			
1	US (United States) : 386 users, 4303 hits	6	CN (China) : 82 users, 410 hits
2	DE (Germany) : 179 users, 1950 hits	7	IN (India) : 81 users, 693 hits
3	GB (United Kingdom) : 131 users, 1487 hits	8	IT (Italy) : 63 users, 527 hits
4	MX (Mexico) : 94 users, 714 hits	9	FR (France) : 50 users, 928 hits
5	BR (Brazil) : 90 users, 754 hits	N	Other Countries : 963 users, 10176 hits

1-10 uses (1554) 11-25 uses (386) 26-50 uses (136) 51-75 uses (24) 76-100 uses (6) 101-150 uses (8) 151+ Uses (5)

7.2.3.3 Period and Frequency of Use

To understand how *often* and for *how long* the toolkit is used by individual users, the mean time between each use (*recurrent period*, hours) and the time between the first and last use (*total period*, days) was computed for each *toolkit user*.

7.2 Usage Statistics

Analysis revealed a correlation between the *total* and *recurrent periods* $r(2117) = .51$, $p < .001$ which suggests that people who use the toolkit for a short amount of time do so quickly (i.e. experimentation all on the same day), whereas people who use the toolkit for longer do so on a regular basis (i.e. daily). To examine this in more detail, the *total period* was divided into five nominal factors representing the number of weeks the toolkit was used for.

Figure 98 and Figure 99 present this data as relative and absolute density plots⁶⁷. These show that people with shorter *total periods* (i.e. <1 week) almost always concentrate all their usage within a single day (note the sharp decrease in relative density in less than 8 hours). By contrast, those with longer *total periods* (i.e. 4 weeks+) generally have *recurrent periods* of over 24 hours or more. The middle ground (i.e. over a week, but under a month) tends to have a relatively even *recurrent period* distribution, with slight peaks around the 12, 24 and 60 hour marks. This suggests a project timespan of around 2-4 weeks with an initial period of experimental development (where the toolkit is used a lot on the same few days) which can then develop into a steady daily use.

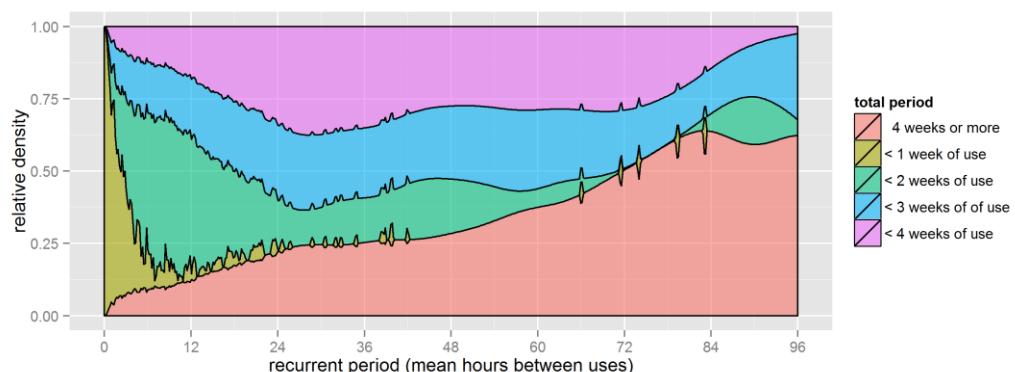


Figure 98: Relative density plot of recurrent period (hour) split by total usage period (weeks).

Cross-referencing these findings with the absolute density distribution plots in Figure 99 shows a very high volume of *toolkit users* in the <1 week of use category (88%). The remaining (12%) have a striking density increase around the 24 hour

⁶⁷ The two plots provide better visual comparison due to the high number of short-term users relative to long-term users.

7.2 Usage Statistics

mark and similar smaller bulges around 48 and 72 hours; suggesting a small number of regular daily users intermixed with a set of irregular yet repeated users.

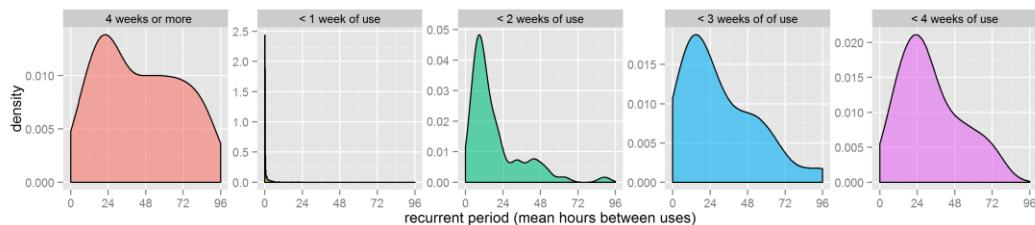


Figure 99: Density plots of recurrent period (hour) split by total usage period (weeks). Y axis is free to represent scale.

Logically these *recurrent periods* are likely to be shifted slightly lower due to the ‘experimentation’ phase of getting to grips with the toolkit and more intermittent longer term use. The idea of an experimentation phase is supported by a lower median *recurrent period* ($Mdn=1.2$ hours) than mean ($M=29$ hours, $SD=232$). Furthermore, a correlation between mean and standard deviation of *recurrent periods* $r(2117) = .49$, $p < .001$ suggests that regular toolkit use has a daily *recurrent period*. This can be seen as a visual trend in Figure 99, by noting how the peak recurrent periods get closer to 24 hours with each extra week (<2 weeks, <3 weeks, <4 weeks) as the impact of the ‘experimentation phase’ on the mean diminishes with additional data.

7.2.3.4 Toolkit User Classification

From these findings, three speculative toolkit user classes are identified and characterised:

1. *Curious*: These *toolkit users* only use the toolkit a small number of times (i.e. < 10) for a short amount of time (i.e. < 7 days). It is likely that they are experimenting with its capabilities and evaluating it to work out if it meets their needs or expectations. They may have a specific idea in mind already or simply have been interested by the publicity video.

7.2 Usage Statistics

2. *Regular*: These *toolkit users* operate the toolkit on a more regular basis (i.e. once every 1-4 days) over a greater period of time (i.e. 1-4 weeks). However, this period and the intensity of the usage can vary. These *toolkit users* are likely engaging in more purposeful usage, perhaps building a specific project or using it at an event.
3. *Daily*: These *toolkit users* operate the toolkit on a daily basis (i.e. approx. every 24 hours or less) and do so over longer periods of time (i.e. 1 month+). They are likely using the toolkit for a specific purpose, such as a permanent installation.

To understand how a '*curious*' *toolkit user* becomes a '*regular*' *toolkit user* the motivation for downloading the toolkit (i.e. expectations and project ideas) should be considered. However, capturing and assessing these is difficult given the low use of the research form. Furthermore, focusing on the '*curious*' users may actually yield less academically interesting findings if their expectations simply do not match the toolkit's purpose (i.e. an interactive whiteboard). With that in mind, the next two subsections focus on reported toolkit issues and individual projects in an effort to understand the expectations and if the toolkit was able to meet them.

7.2.4 Reported Issues and Feature Requests

This section analyses the defects and feature requests reported via the Google Code page. In total 6 unique defects were reported. A further 12 invalid⁶⁸ or duplicate issues were reported, and 4 feature requests were received. These are presented in Table 16.

The most severe accepted defect was *Issue 5* that prevented touch from working correctly on certain European system locales due to a number format localisation issue. Given the varied geographical adoption of the toolkit, this was resolved quickly. *Issues 1 and 2* remain open lacking the hardware necessary to

⁶⁸ The reported defect is not due to a problem with the toolkit. Common examples include missing dependencies (e.g. Kinect SDK) and not plugging the Kinect into the computer.

7.2 Usage Statistics

reproduce them. However, in both cases, workarounds are available. The most significant feature request was the ability to inject touch events automatically into 3rd party content. This feature was promptly added, and can be accessed via the ‘Advanced Surface Properties’ dialog. *Issue #8* (Linux port) requires more work, but is suggestive that the Windows platform is not necessarily preferred.

Table 16: Reported toolkit defects and feature requests.

ID	Type	Status	Summary
1	Defect	Open	Content font size varies with the system font size. On very high resolution displays (i.e. 3200x1600) this creates font rendering issues in the toolkit content. This can be worked around by reducing the system font size to 100%.
2	Defect	Open	Black screen running in Parallels⁶⁹ on OSX. Selecting a projector (Section 5.3.1.1) on a MacBook Pro Retina places the output window on the primary monitor rather than the secondary projector. This can be worked around by manually moving the window.
3	Defect	Fixed	Debugging link “Launch Chrome Inspector” does not work. Caused by an assumption about user install directories. This can be worked around by navigating a Google Chrome browser to http://localhost:9222 while the toolkit is running.
4	Defect	Fixed	Broken Showcase Link. The link to the community support can showcase linked to the wrong URL. This has since been fixed.
5	Defect	Fixed	Touch not working with European system locales. The number formatting (i.e. comma rather than period for decimal points) prevented touch from working. This was identified and quickly fixed within a few days of release.
6	Defect	Fixed	Toolkit not shutting down correctly. Occasionally the Kinect drivers do not release resources. This caused the toolkit to crash when restarted.
7	Feature Request	Open	Automatic Screen Extending. This would enable the toolkit to automatically configure the projector for screen extending.
8	Feature Request	Open	Linux Port. This would enable the toolkit to run on non-Microsoft Kinect hardware (i.e. Asus Xtion Pro Live) and potentially run on smaller embedded devices.
9	Feature Request	Done	Touch Injection on 3rd party websites. This would allow the toolkit to inject the touch interaction into 3 rd party websites (i.e. Bing Maps). This feature has since been provided.
10	Feature Request	Open	Whiteboard application sample display to be provided. This would help create simple installations (i.e. classroom) and demonstrate the touch interaction modality.

⁶⁹ Parallels is a desktop virtualisation application for OSX which allows Windows applications to be run on a Mac. <http://www.parallels.com/>

7.2 Usage Statistics

Overall, the low number of issues reported in Table 16 is encouraging. The absence of evidence to the contrary (in the face of extensive usage) suggests that the toolkit is stable. The number of duplicate or invalid issues suggests that—even with the readme, FAQ, and tutorial video—clear and concise quick start guides are a must.

7.2.5 Project Listings

The descriptive summary and analysis of usage patterns show significant adoption and regular use. To describe this usage in more detail (i.e. who are the main user groups? What kind of application scenarios are driving usage?) personal correspondence⁷⁰ from *toolkit users* was coded and analysed. The results are presented as a large descriptive table (Table 17) and a corresponding summary analysis (Section 7.2.5.3).

7.2.5.1 Data Capture

The high volume of toolkit related correspondence versus the negligible yield from the research form made it the obvious choice for revealing the most interesting findings. Analysis of unstructured correspondence (i.e. emails, etc.) required each item to be coded and analysed. The coding fields are listed below:

- *Application Scenario Summary*: A short sentence which characterises the application scenario, if applicable.
- *Specific / Exploratory*: Is the toolkit use exploratory (i.e. general interest) or with a specific intention (i.e. realisation of a pre-defined application scenario).
- *Background*: The major background of the *toolkit user*: **PER**sonal, **CS**-**A**cademic, **AC**ademic, or **C**OMmercial.

⁷⁰ That which did not come through the public support forum or Google Code page.

7.2 Usage Statistics

- *Contact Motivation:* What motivated the correspondence: ***ADvice, FAQ, COMmercial interest, DeskTOP applications support, or SHOWcase inclusion?***
- *Desktop Applications:* Did the *toolkit user* desire desktop application content support, in addition to web content?
- *Existing Code:* Did the application require integration with existing code or content?

To address ethical concerns with the analysis of personal correspondence all the data collected is anonymous and presented in summary. A small minority explicitly requested that their application scenarios or identity not be included. These cases do not form part of this analysis.

The major limitation of this sample is that it is self-selecting. Indeed, the impetus for correspondence (i.e. the reason for getting in contact: errors, advice, commercial propositions, etc.) has a different motivation to filling out a research form. This has the potential to bias the sample in unexpected ways. To factor this into the analysis, correspondence is also coded by ‘contact motivation’. This analysis must also necessarily trust that information provided is accurate, although this is a problem with any survey method.

7.2.5.2 Data Presentation

Table 17 contains a list of real application scenarios derived from an analysis of personal correspondence. All entries are presented anonymously. External links are either public domain or provided with explicit permission.

7.2 Usage Statistics

Table 17: List of application scenarios which use the toolkit. Entries are coded with abbreviations listed in Section 7.2.5.1. Notable cases highlighted in orange.

id	Application Scenario Summary	Specific / Exploratory	Background	Contact Motivation	Desktop Applications	Toolkit Extensions	Existing Code or Content
1	Foot and touch detection in a vocational training context.	E	CSA	ADV	N	N	
2	Exploring collaboration across larger vertical and horizontal surface displays. Also interested in frameless UI experiences.	E	CSA	ADV	Y	N	WPF Control
3	Real Estate Advertising	S	COM	FAQ	N	N	
4	Primary Education Support	E	COM	FAQ	N	N	
5	Formula 1 in Schools. AR presentation.	S	PER	ADV	N	N	
6	Trade show stand with web maps integration.	S	COM	ADV	Y	N	
7	Adding TUIO support (unable to disclose more).	S	CSA	FAQ	Y	Y	TUIO and WPF Controls
8	Transforming a wall into a synth-based musical instrument (video with permission) youtu.be/DEPzRFOHOY0	S	PER	ADV	N	N	Midi Synth
9	Home automation with interactive surfaces. Integration into existing platform.	S	COM	COM	N	N	Proprietary Platform
10	Interactive wine bars.	E	COM	FAQ	N	N	
11	Testing multi-touch environmental modelling software.	S	ACA	DTOP	Y	N	
12	Technology promotion (unable to disclose)	E	COM	COM	N	N	
13	Interactive whiteboard replacement.	S	PER	DTOP	Y	N	
14	Floor display for special needs children.	S	PER	ADV	N	N	
15	Technology promotion (unable to disclose)	S	COM	COM	N	N	
16	Object Detection to support worker assistance.	S	CSA	COM	N	Y	WPF Control
17	Experimentation with museum exhibits.	E	COM	FAQ	Y	N	Flash / AS3
18	Experimentation for personal projects	E	PER	FAQ	N	N	
19	Emergency response control centre design (unable to disclose more at this stage)	S	COM	FAQ	N	N	
20	Display pieces for architectural work	S	PER	DTOP	Y	N	Snowflake

7.2 Usage Statistics

21	Customer insight and innovation centre 'video wall'	S	COM	COM	N	N	
22	Converting existing TV into Smart TV	S	PER	DTOP	Y	N	
23	Whiteboard replacement	S	CSA	FAQ	N	N	
24	Interactive museum exhibits	E	COM	COM	N	N	
25	Interactive table for the home	S	PER	DTOP	Y	N	
26	Interactive 'Kanban board' to help organise office work.	S	COM	DTOP	Y	N	Kanban App
27	Eye doctor using displays to educate patients.	S	COM	ADV	N	N	
28	Evaluation for use in an interactive nursery with toys and object detection.	E	COM	COM	Y	N	Unity Game Engine
29	Conducting a study of innovation in education.	E	CSA	COM	N	N	
30	TouchPTV: Enhancing TV Experiences with Projection. Interactive room and projected video controls. Touch and foot interaction.	S	CSA	ADV	N	N	
31	Interactive displays at a car dealership.	S	COM	FAQ	N	N	
32	Technology exploration. Received grant to develop armrest media controller, automatic light when seated, contextual textbooks in class, therapy aid for walking, and children's floor games.	E	PER	SHOW	N	N	
33	Presentation system for customer facing meetings.	S	COM	COM	N	N	JS Application
34	Touch TV creation	S	PER	DTOP	Y	N	
35	Assisting with the manufacture and assembly of bathroom and kitchen taps.	E	COM	ADV	N	N	PDF Content
36	Integration into an internet of things platform.	S	CSA	COM	Y	N	Thing Broker
37	Evaluation of the toolkit (application not disclosed)	E	COM	COM	N	N	
38	Creating a low cost rear projection touch table	S	PER	DTOP	Y	N	
39	Two sided touch screen	S	CSA	COM	N	N	TUIO and WPF Control
40	Experimentation	E	PER	FAQ	Y	N	Flash / AS3
41	Classroom whiteboard replacement	S	PER	ADV	Y	N	
42	Creating a low cost interactive desk	S	COM	COM	N	N	
43	Evaluation and experimentation (application not disclosed)	E	COM	COM	N	N	
44	Classroom whiteboard replacement	S	PER	DTOP	Y	N	

7.2 Usage Statistics

45	Home automation	E	COM	COM	N	N	
46	Interactive floor	S	COM	COM	N	N	
47	Restaurant dining tables evaluation.	E	COM	COM	N	N	
48	Real estate virtual tours exploration	E	COM	DTOP	Y	N	
49	Interactive whiteboard replacement.	S	COM	COM	Y	N	
50	Interactive whiteboard replacement	S	COM	DTOP	Y	N	
51	Evaluating commercial potential (application not disclosed)	E	COM	COM	N	N	
52	Engaging students outside lecture halls	S	ACA	ADV	Y	N	
53	Experimentation with projection mapping	E	PER	ADV	N	N	Flash / AS3
54	Experimentation creating a desk display for children in schools.	S	ACA	DTOP	Y	N	
55	Building a homebrew interactive coffee table for his anniversary with his girlfriend.	S	PER	ADV	Y	N	
56	Developing more advanced gesture detection interaction modalities.	S	CSA	ADV	N	Y	
57	Tourism interactive installations.	E	COM	COM	N	N	
58	Exercise games for children with motor control problems.	S	ACA	ADV	N	N	
59	Fine Arts project. Creating a board which, when areas of it are touched, display imagery.	S	ACA	ADV	N	N	
60	Mini games to help visual and developmental skills in children and recovering patients.	S	COM	ADV	N	N	
61	A planning charrette for architects and building designers.	E	COM	COM	N	N	
62	Technology integration company (application not disclosed)	S	COM	DTOP	Y	N	
63	Visual Instrument made of projected wires that play different notes when held down.	S	PER	ADV	N	N	
64	Developing a food game as an interactive museum exhibit.	S	COM	ADV	N	N	Flash / AS3
65	Wedding photography showpiece.	S	PER	FAQ	N	N	
66	Squishable sidewalk spiders Halloween game. (public video) youtu.be/ZXmWdcBHx9g	S	PER	SHOW	N	N	JQuery Library jFormics
67	Interactive water projection as part of a technology exploration course.	E	ACA	ADV	N	N	
68	Integration with Resolume VJ software.	E	PER	SHOW	N	N	Resolume
69	Interactive whiteboard replacement.	S	PER	DTOP	Y	N	
70	Interactive dungeons and dragons table.	S	PER	DTOP	Y	N	D&D App

7.2 Usage Statistics

7.2.5.3 Findings

Table 17 presents 70 application scenarios alongside coded fields which characterise the *toolkit user*. Although a minority have a similar theme (i.e. whiteboard / table-top replacements) many are novel real-world application scenarios that use the toolkit. Education, novel advertising, displays in restaurants and bars, healthcare, technology research, and individuals experimenting by creating low-cost interactive coffee-tables all offer application scenarios.

The major user groups are: corporate (46%), personal (31%), computer science academic (14%), and non-computer science academic (9%). These groups are identify that interactive projected displays are of corporate interest, but also that the toolkit is being used in for academic application driven research (in both a CS and a non-CS context). This is a strong argument for asserting that the toolkit is archiving its overarching goal of supporting user innovation across different *toolkit user* communities.

Figure 100 shows *toolkit users* backgrounds and contact motivation. The personal *toolkit users* were the only ones to contact for showcase reasons, perhaps due to publication restrictions on corporates and academics.

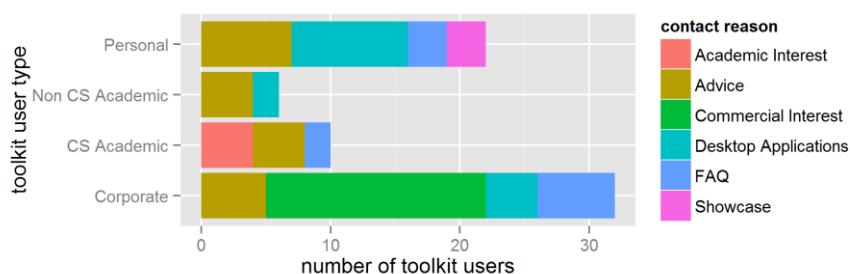


Figure 100: The proportion of toolkit user backgrounds separated by correspondence motivation.

Figure 101 shows *toolkit user* backgrounds with the nature of their toolkit use: specific or exploratory. A Pearson chi-square test showed this did not vary significantly across the backgrounds: $\chi^2(3, N = 70) = 4.47, p = .22$; all exhibiting generally more specific use (66%) than exploratory (34%). This ratio supports the argument that the toolkit is suitably supportive of application driven uses cases.

7.2 Usage Statistics

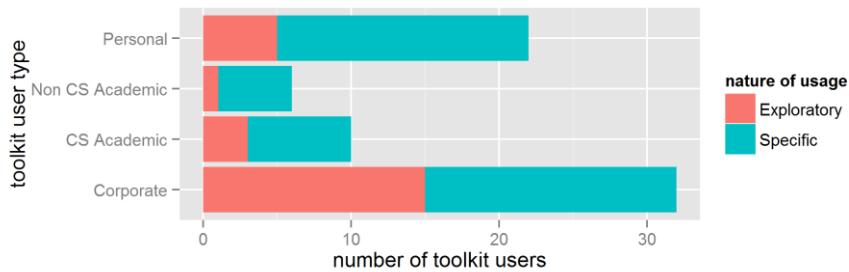


Figure 101: The proportion of toolkit user backgrounds separated by the nature of use.

A common theme was an interest in supporting desktop applications as well as programmable web content. Figure 102 plots this level of interest for all *toolkit user* backgrounds and the nature of their usage. The strongest interest was from personal *toolkit users* with a specific application scenario. Cross referencing these values with Table 17 reveals that many from this group were striving to create a low-cost interactive table or whiteboard solution. However, use by non-CS academics and existing corporate applications also offers a compelling case for desktop application support.

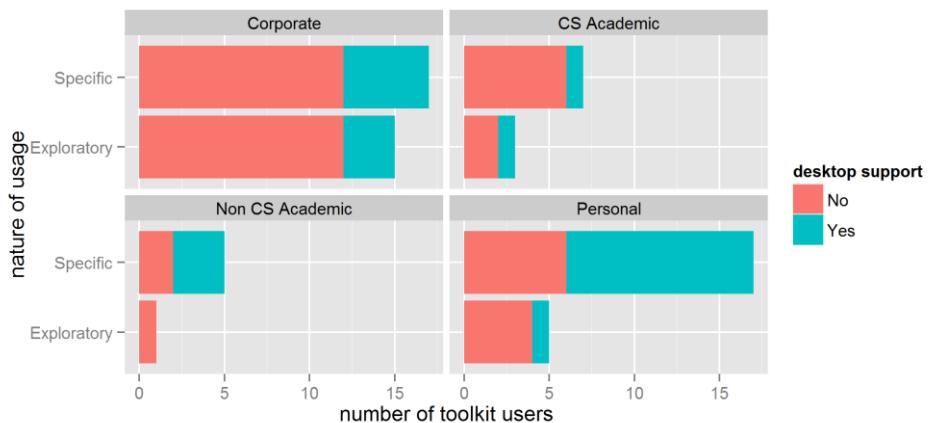


Figure 102: Level of interest in desktop application support, separated by nature of usage and background.

In conclusion, the analysis of toolkit usage reveals a high level of use that typically falls into a number of usage patterns. A further analysis of correspondence from toolkit users helps to clarify the backgrounds of toolkit users and identify

7.3 Case Studies

typical application scenarios. However, the nature of the correspondence analysis limits the depth to which these can be interrogated in detail. To address this issue, the next section presents a series of short case studies.

7.3 Case Studies

This section presents a set of short case studies to provide deeper insight into the application scenarios the toolkit supported. Six cases are selected in total—all taken from the personal correspondent section with permission. These examine the breadth of applications the toolkit created, assess its features, and indicate directions of further study and potential requirements refinement.

7.3.1 Patient Education Hub

Dr. Paras Mehta is an ophthalmologist (Consultant Eye Surgeon) based at the Baroda Eye Institute⁷¹ in Gujarat, India. The toolkit was used to create a patient education hub that informed people about the processes and technologies used by the institute. To quote Dr. Mehta: “*My idea was to project a background picture [that] contains a cross section of eye. Whenever, a user is touching any part of that eye section, its details should be highlighted in popup/hover/tooltip kind of text.*” This concept was ultimately extended to include a separate button menu display (*Requirement 10, Requirement 7, Requirement 16*), videos (*Requirement 17*), and two image cross-sections: a theatre and the eye (*Requirement 7, Requirement 15*).

Figure 103 shows the display following development and installation. It began with an initial development phase without full installation into the intended area. After Dr. Mehta confirmed the toolkit was suitable through initial testing and a first iteration of the software was developed, the hardware mountings were installed into the space.

⁷¹ <http://www.barodaeye.com>

7.3 Case Studies



Figure 103: Toolkit installed at the Baroda Eye Institute. Photographs courtesy of Dr. Paras Mehta.

Physically, the display covers a 34" wide by 43" high frosted glass panel. The Kinect is mounted to ceiling at a distance of 1.2 meters. As the projector is mounted behind the glass, the 'technology' appears to be hidden from the users. All the development was done in house without professional programming experience (*Requirement 1*)—adapted from the samples and documentation provided with the toolkit (*Requirement 19*). Advice was requested in order to get the toolkit to play AVI/MP4 videos. Although the `<video>` tag was used, a 'Missing Plugin' message was displayed. This was caused by a proprietary MP4 decoder not being installed. To resolve the problem, the video was converted to the open OGG format. The sample supplied with the toolkit relied on YouTube videos.

7.3.2 TouchPTV: Enhancing TV Experiences with Projection

This case study presents two projects designed to enhance television viewing experiences with ubiquitous projection to provide additional screens. These were

7.3 Case Studies

bachelors thesis undertaken by *Dennis Wolf* [168] and *Kathrin Osswald* [169] at the University of Ulm and are discussed here with permission. *Mr. Wolf* focused on the back-end implementation (i.e. interaction techniques, application presentation, client-server architecture) and *Ms. Osswald* focused on applications development and user evaluation of the enhanced TV experience.

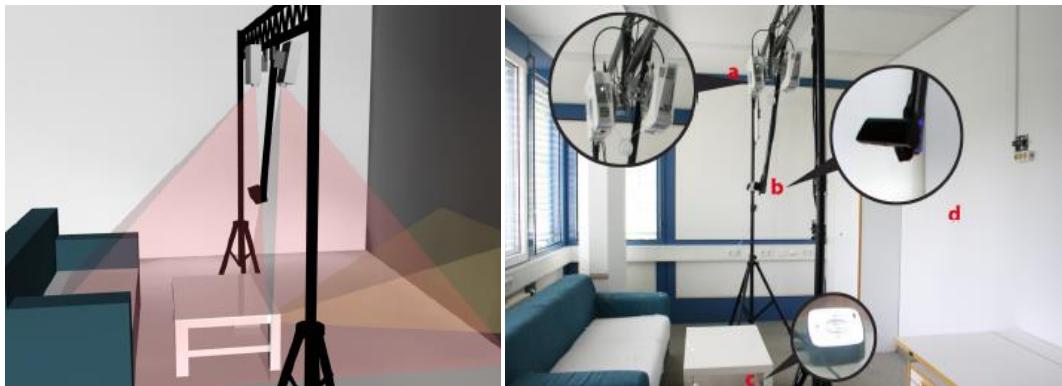


Figure 104: Augmented living room setup for TouchPTV. Photographs courtesy of Mr. Wolf and Ms. Osswald.

The projects are application driven computer science studies that take place in a controlled environment (Figure 104). The environment was designed to emulate a living room, featuring: a projected wall TV, a couch, a coffee-table, and a floor—all interactive (*Requirement 7*, *Requirement 9*, *Requirement 10*, and *Requirement 16*). It was intended to be suitable for two users (*Requirement 14*). In its default state, all the projections are turned off, leaving the user with a common living room scene. The user can activate the system and select necessary functions by touching the coffee-table. Following a review of available toolkits, they chose to adopt this toolkit as it supported three main project needs: maximising hardware resources to cover a large space (*Requirement 20*), support for more than predefined interface widgets (*Requirement 8*), and the rich features of HTML were needed for the desired applications (*Requirement 17*).

7.3 Case Studies

7.3.2.1 Development

As projected interfaces lacked the haptic feedback of traditional touch devices (i.e. vibration) Mr. Wolf and Ms. Osswald investigated ways to improve interaction with the projections (Figure 105, Requirement 18). They note that a projected button is activated by touch rather than by push and depression, and thus random swipes across the interface can unintentionally trigger targets. They present three approaches to minimise this effect: (1) button timeout (prevents repeated presses), (2) a long touch (ensures the function is intended), and (3) colour feedback.



Figure 105: Showing the ‘long touch’ interaction modality. Note how the arc draws around the finger as a method of visual and temporal selection feedback. Photographs courtesy of Mr. Wolf and Ms. Osswald.

To show relationships between different interfaces deployed in the space they projected lines between them. To achieve this, a save file was manually edited to include a ‘background’ surface that covered the entire projection field (Figure 106).

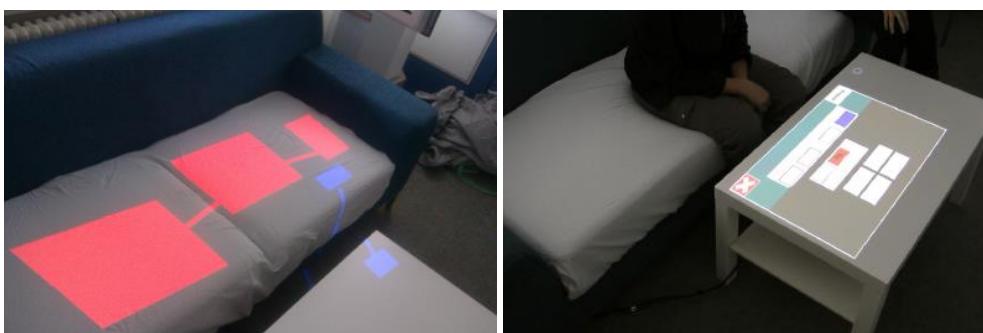


Figure 106: Left: Projected lines connecting individual surfaces. Right: The projected control menu coffee-table. Photographs courtesy of Mr. Wolf and Ms. Osswald.

7.3 Case Studies

7.3.2.2 Applications

A total of three applications were developed for this environment, accessible via an additional projected control menu. These were: (1) a documentary application which allows users to reveal more contextual information on the coffee-table without interrupting the narrative of the documentary, (2) a multi-player quiz application which allows users to play along with quiz shows, and (3) an interactive dance application which used the foot-detection capabilities. These are shown in Figure 107 and described in more detail in their reports [168] and [169].



Figure 107: Three applications developed for TouchPTV. Top-left: quiz application, Top-right: documentary application, and Bottom: dance application. Photographs courtesy of Mr. Wolf and Ms. Osswald.

Applications (1) and (2) could be controlled from the couch, whilst application (3) was controlled through foot movements (*Requirement 12, Requirement 18*). The use of the affordances of the furniture and space in the design (i.e. hiding answers on the other side of your person) and the content (i.e. time-coding relevant information into the documentary) that can be accessed without disturbing others are

compelling demonstrations of how physical spaces can be converted into computer interfaces (*Requirement 11*).

The majority of their analysis focuses on the use of projection to enhance the TV viewing experience. As this is application specific, it is not reported here. However, elements of it could be argued to generalise into useful information for those developing interactive projected displays. This project offers evidence that the toolkit achieves part of its primary goal by demonstrating that it can be used by others to generate new knowledge about applied interactive projected displays.

7.3.3 Object Detection

This case study describes how object detection support was added to the toolkit (*Requirement 22*). Markus Funk (PhD Student, University of Stuttgart) visited Lancaster University to combine the toolkit with his C++ implementation of the BRISK feature recognition algorithm [170] that used OpenCV. This gave display content the ability to determine which objects (if any) are placed in an area visible by a separate web camera. This was motivated by the application scenario of assisting workers with general learning disabilities. Elements of this work have since been published in the CHI extended abstracts [171].

The integration process was relatively simple given programming experience. To add new definitions to the object detection recogniser, images of the object to be recognised are added to a target folder on the local file system. The existing C++ binary was adapted to produce textual output which listed the filenames of the recognised objects. To allow display content to execute and access the output of this binary, the native toolkit *Content API* was extended by adding a new request handler: ‘StartProcess’ (an `IRequest`). This enabled display content to invoke external processes and interact with them by streaming the `stdin`, `stdout` and `stderr` pipes into JavaScript (*Requirement 13*).

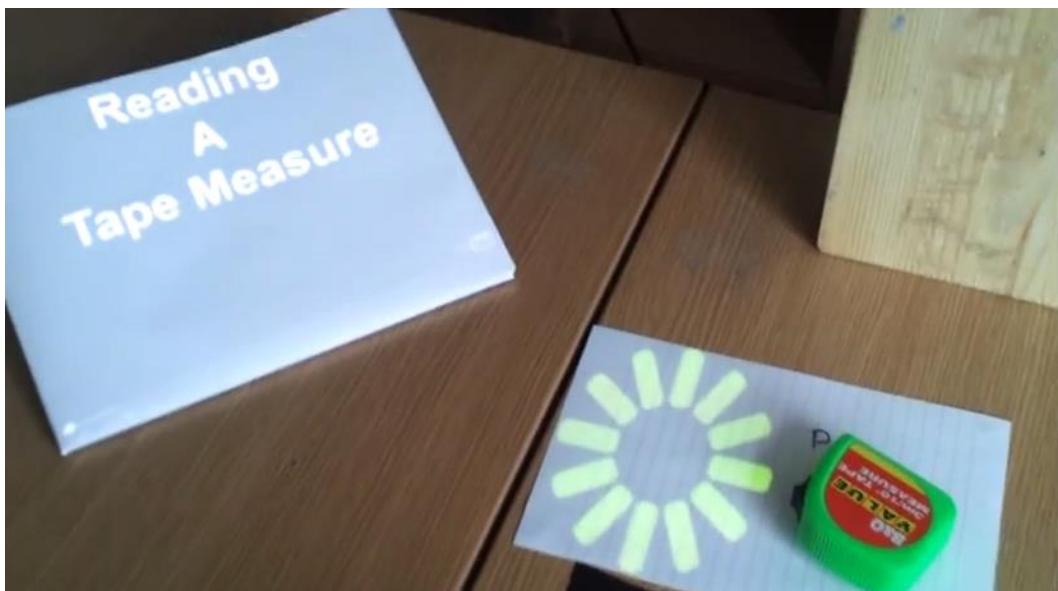


Figure 108: Demonstration of object detection integration with the toolkit.

The result was successful and demonstrated through a mock-up display (Figure 108). The stock `PresenceDetector` module was used to determine when to invoke the object detection process. If a known object was detected, a related video then opens on a nearby display (*Requirement 16*). Although this extension could be useful in other contexts for other users—particularly the home automation application scenarios in Section 7.2.5.2—it was not included in the public releases as allowing display content to invoke external processes is a major security risk. It has since been added as an option that users must explicitly activate, but alludes to additional security-based requirements as toolkits become more established.

7.3.4 Interactive Milk (Hyper Island)

Hyper Island is a private educational institution that specialises in real-world industry training using digital technology. A team of three students on the digital media creative course elected to use the toolkit in order to develop an interactive liquid touch screen (*Requirement 1*, *Requirement 2*, *Requirement 3*). The final project was demonstrated (alongside others) at the Media Evolution City in Malmö, Sweden to approximately 400-500 industry professionals (*Requirement 21*).

7.3 Case Studies

They use the AquaTop [172] system as inspiration, deciding to project onto milk to create a better projection surface than clear water. They initially found that the touch calibration was offset (likely due to the Kinect sensing the container through the water) but that this could be corrected by calibrating against an initial plane (*Requirement 18, Requirement 5*). They got in contact via the support forum, asking how to get an existing web app to work with touch detection⁷² (*Requirement 19*). The web app supported touch events so it was possible to simply click to enable the experimental touch injection mode. However, for the exhibition itself they elected to use one of the existing sample displays provided with the toolkit. The physical hardware and the final results are shown in Figure 109.



Figure 109: Interactive milk at Hyper Island. Left: Prototype setup. Right: Deployed at the Malmö event. Photographs courtesy of Hyper Island.

This case study demonstrates a number of the toolkit's strengths: (1) those with little to no-programming experience were able to achieve their goals, (2) that the toolkit is able to support projection onto exotic materials such as liquids, and (3)

⁷² <http://weavesilk.com/> This would allow users to draw patterns in the milk.

7.3 Case Studies

that the toolkit is suitable for a rapid prototyping context (see the mounts in Figure 109) and public use. With that said, the toolkit could have been improved if it supported a wider range of existing content. Exploration of interaction with water—a non-technological material—was the focus of this project, not the development of content. This challenges the assumption of content-driven applications that guided the section of research probes and requirements.

7.3.5 Thing Broker and Really Easy Displays

Mr. André Bueno (UFSCar, Brazil) and *Mr. Roberto Calderon* (UBC, Canada) are Ph.D students studying the human perception of ubiquitous technology in interactive environments. An on-going project integrates Thing Broker⁷³ [173], the Really Easy Displays framework⁷⁴ [174], and the toolkit described in this thesis.

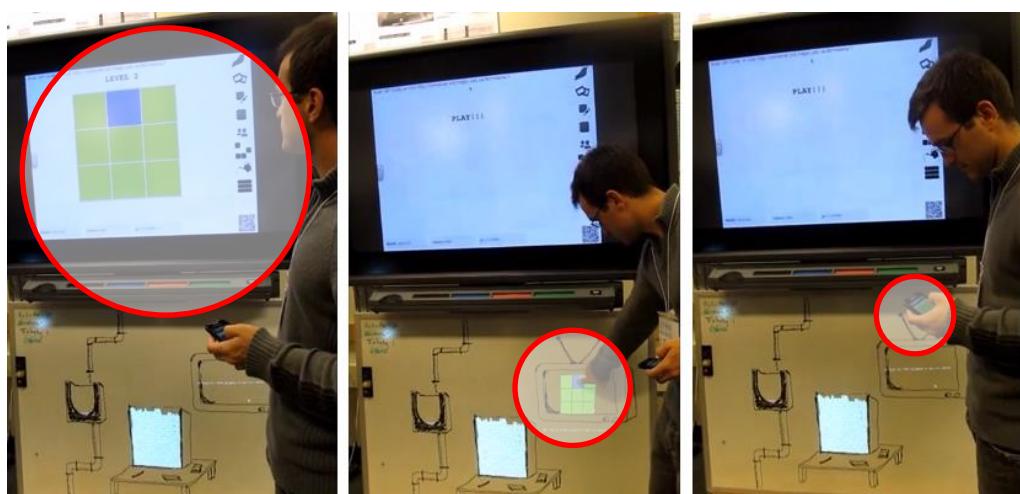


Figure 110: Integration ThingBroker and the RED framework. Red overlays show big screen, projected display, and mobile device. Photographs taken from: <http://youtu.be/4oLqq4qiiCY>

They are building applications that can be controlled across different kinds of devices, including small and big screens (*Requirement 7*). This will see interactive projected displays applied to cross-device ubiquitous interaction using existing

⁷³ Thing Broker is a RESTful interface for easy internet of things applications development: <http://www.magic.ubc.ca/wiki/pmwiki.php/ThingBroker/ThingBroker>

⁷⁴ Really Easy Displays Framework is a web framework that allows users to develop and deploy multi-display applications easily: <http://red.icd.magic.ubc.ca/>

7.3 Case Studies

frameworks (*Requirement 22*). Two simple games have been developed thus far: a simple ‘hit the racoon’ game: (<http://youtu.be/xMiXqHE-qt0>, not pictured), and a ‘memorise’ game that is played across a large screen, a projected screen, and a mobile device (<http://youtu.be/4oLqq4qiiCY>, Figure 110).



Figure 111: Watering the garden. Photograph courtesy of Mr. André Bueno.

An additional project called the “*Watering the garden*” was published at DIS2014 [175]. This featured a plastic box containing real grass and a small pipe attached to a pump controlled by an Arduino⁷⁵ microcontroller. They drew a bucket on the whiteboard with marker pen and it filled with virtual projected water (Figure 111). When a user touched the virtual water, the bucket emptied and real water ran into the garden. This project demonstrates how frameless projection can animate physical objects that are expected to be static to produce a creative user experience. This invites reflection on the value of projection (with its high price point) relative

⁷⁵ Arduino: <http://arduino.cc/>

7.3 Case Studies

to the potential of screen-based technologies to generate similar findings. Toolkit features including: projection mapping (*Requirement 9*), rapid experimentation (*Requirement 5*) with variable display sizes (*Requirement 7*), and interoperation with other frameworks (*Requirement 22*) made it easier to explore different configurations and designs.

7.3.6 Two Sided Transparent Touch Screen

Mr. Christopher Bull is a Ph.D student (Lancaster University, UK) researching software engineering studios and education. A two-sided transparent touch screen was prototyped collaboratively with the author of this thesis. The designs followed a discussion of the use of whiteboards in the studio environment and how they inherently divided the space and affected collaboration. The project constraints were that it must be completed in a short amount of time (1 day max, *Requirement 5*) and use a minimum of materials and existing equipment. The outputs were used to motivate further work. The finished system (Figure 112) was made using a 1x0.6m sheet of wood, a sample of transparent diffuser film, and two small sheets of glass cut by a local supplier.

Although the prototype was smaller than would-be required for deployment in the software studio, the system provided different novel interaction techniques that helped two people to simultaneously share a visual computer interface (*Requirement 14*). This identified a number of HCI challenges for transparent displays, such as: poor contrast, disturbed motion behind the display, and the correct orientation for different types of content. The authors experimented by mirroring text, flipping content globally or locally, and with simultaneous interaction on both sides.

7.3 Case Studies

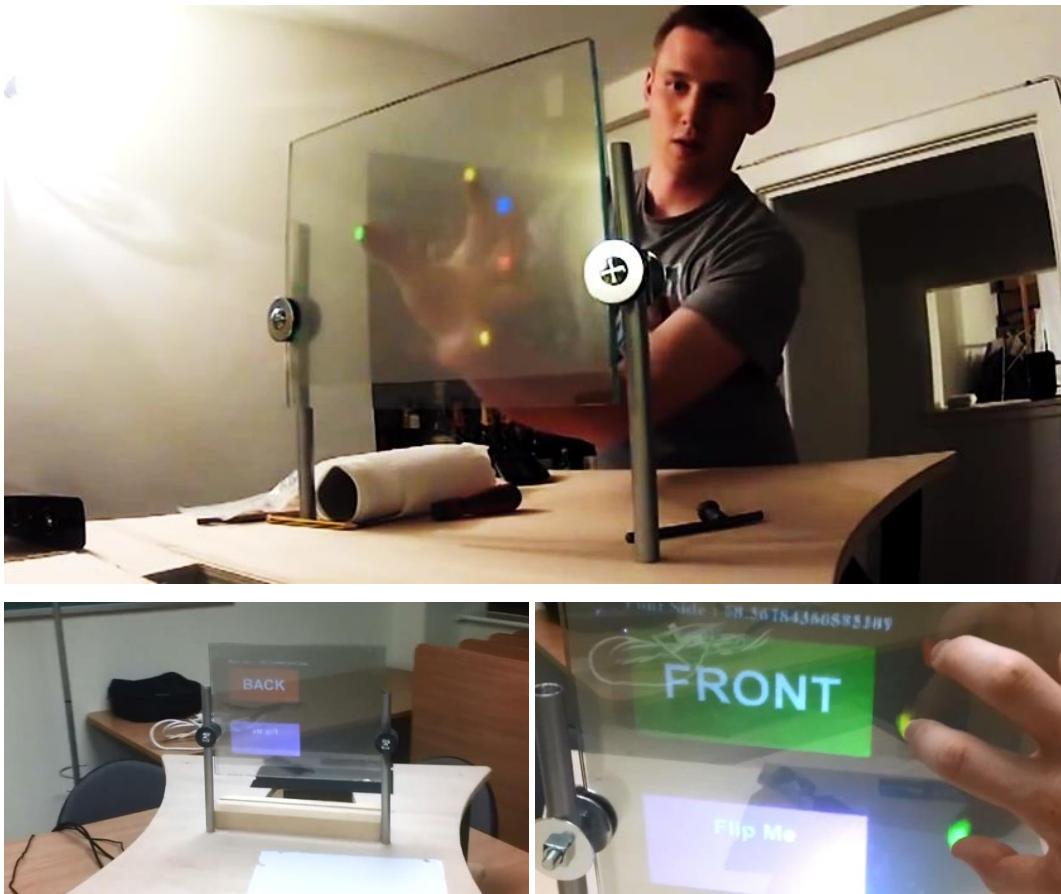


Figure 112: The two sided transparent touch screen. Left: far view from the back. Right: close view from the front. Top: screenshot from: <http://youtu.be/BOTbbx95Qp0>

To detect which side of the screen the user was interacting on, the multi-touch detection settings (Section 5.3.2) used a negative `surface_offset` value (i.e. behind the glass) and a larger touch volume (i.e. to include both sides). To remove point cloud noise created by the glass, a line of depth-test code was added which rejected points in the centre of the sensing area. The remaining points describe a user's fingers. An additional variable 'side' was added to the generated touch events which classified the side of the touch point based on its depth (*Requirement 18*). The ability to modify the touch detection script to add edge cases was very useful, as it meant that changes could be prototyped immediately without recompilation. It also meant that the two-sided touch screen code was compatible with existing display content without requiring significant modification.

7.4 Discussion

It was also possible to define ‘imaginary’ toolkit surfaces either side and above the glass (by placing an object there, defining the surface, and then taking the object away again). This enabled crude yet effective gesture detector capable of determining when people are passing objects around or above the screen; capturing more of the interactions in a collaborative workspace. The toolkits use of variable display sizes, programmable content, and multiple interaction modalities meant that the interactions for this specific device could quickly adapt to its (prototype, and thus changing) physical form. However, the toolkit constraints meant that only one projector could be used at once---making it difficult to project onto both the glass (from below) and onto the top of the wood (to form an interactive work surface). Although it would have been possible to run two instances of the toolkit simultaneously and use a framework such as RED [174] manage inter-deployment communication, a better approach would be support multiple depths sensors and projectors within the same toolkit instance. This would enable increased free-form design choices when designing new device form-factors and make it possible to cover larger physical areas. However, this is not without increased technical challenges and computing requirements.

7.4 Discussion

This section discusses the findings of the usage statistics and case studies. It reflects on the strengths and weaknesses of different toolkit features in the context of different *toolkit user* groups and the limitations of the analysis.

7.4.1 Ease of Setup

A major strength of the toolkit is that it is generally simple and fast to set up. This is demonstrated in the previous chapter and clearly echoed in personal correspondence. This makes it highly suitable for rapid prototyping as it is possible to quickly create, delete and manipulate displays with little to no technical

7.4 Discussion

knowledge. These features are available across all the identified user groups, although significant features (particularly, those relating to content development) are restricted to programmers with HTML and JavaScript experience.

Although this restriction is recognised in the requirements, better support can be provided to non-programmers through the addition of desktop application support. This could easily be achieved using a specialist `IDisplay` implementation (Section 5.2.3.1) which assumes the use of the touch interaction modality. In the same vein, better support could be provided to WPF application developers by extracting the touch interaction modality from the toolkit and making it compatible with the Windows touch stack. This would follow an approach very similar to the prototype WiiTUIO toolkit (Section 3.3.5.3).

7.4.2 Web Content

Working with web content and web applications has advantages and disadvantages. For instance, although it restricts the advanced *toolkit user* community to those with HTML and JavaScript skills, it also provides a consistent standard that is relatively easy to learn, has lots of community support, and is widely adopted in other systems.

No instances of erroneous behaviour were reported as a result of the toolkit's implementation of web standards. However, a minority lacked newer features (e.g. hardware accelerated WebGL and the flexbox model). Application development was aided because individual parts of the display content could be edited, deployed, and reloaded without restarting the toolkit application or recompiling code. It was also possible to easily modify existing common modules using the duck-typing features of JavaScript. In the case of the two-sided transparent touch screen this was particularly important, as tweaking the touch detection algorithm would have been difficult to do theoretically.

In terms of limitations, CS Academics found the use of web content restricting, and preferred developing code in native languages. Although in some cases this was down to familiarity, other reasons include that they had existing resources (i.e. a WPF control) or JavaScript was not fast enough for their application scenario. In

7.4 Discussion

terms of user innovation, the first barrier to adoption is the perceived difficulty. To quote correspondence:

CS Academic Toolkit User A: “[It is] restricting with apps with hardware attached, like Arduino. It can be done with doing a web-socket wrapper and exposing it in Javascript, but direct access to serial port is always better.”

CS Academic Toolkit User B: “[We] worry about being too limited by the use of JavaScript, because it even costs much time to implement “basic features”. Is it possible to directly provide touch events in C# without having to use JavaScript?

An unexpected finding is that the ‘content’ of the display was not always an important part of the application scenario design goals. The Interactive Milk case study (Section 7.3.4) is an example of where the physical properties of the interaction drive the application (i.e. as an art piece) rather than the content itself. Reflecting on the toolkit design process, the research probes made the assumption that the content will be the focus of the application. Although this assumption was not particularly harmful, it is likely that the students of Hyper Island would have been more satisfied with the toolkit if it had better support for existing applications.

7.4.3 Monitoring and Debugging

Monitoring the display content via the Google Chrome Inspector was particularly helpful as a scripting and debugging interface. One *toolkit user* noted that having scripting access to all display content running from one central interface (i.e. the ability to type commands directly into the JavaScript console for each item of display content) was integral to his application and aesthetic design processes. However, this is not necessarily the case for interactive projected displays more generally. For instance, distributed applications scenarios, such as the integration of the toolkit with the Thing Broker [173] and the RED Framework [174] (Section 7.3.5), debugging application scenarios in physical spaces needs to take into account a larger software ecosystem. Further research still needs to be done on how this could be achieved. One possibility is proposed in Future Work, Chapter 8.

7.4 Discussion

Similar to those discussed in the previous chapter, new challenges of programming physical spaces were also encountered by some of the toolkit users. For instance, working with multiple people requires toolkit users to maintain control over a larger physical space. This makes development difficult as sufficient numbers of test-users are not always available or able to re-create erroneous conditions. One *toolkit user* suggested simulating a Kinect to address the problem:

CS Academic Toolkit User A: *[The toolkit] lacks a way to simulate a Kinect and the visuals (i.e. people would often work on their couch and would prevent them from testing the projections).*

7.4.4 Maximising Existing Resources

Of the corresponding *toolkit users* (11%) felt their application scenarios would have been easier to create given better support for existing code and content resources. Generally, this referred to content in languages and formats that was not directly supported by the toolkit (i.e. Unity Game Engine, Specialist WPF Controls, 3rd party native applications). Despite these limitations, a number of toolkit users were able to successfully use features such as HTML5 websockets to communicate with external resources (i.e. the musical instrument: #8, Section 7.2.5.2).

A popular application scenario was to use the toolkit to create cheap interactive tables. Typically, these requests are motivated by one of four reasons: (1) they have a pre-existing native application they want to use, (2) they lack experience programming JavaScript, (3) they want to use a full Windows desktop environment, or (4) they cannot afford the cost of a large damage-resistant multi-touch table display.

7.4.5 Interaction Modalities

Toolkit users report that low point-cloud resolution beyond 1.4m made it difficult to create and manage larger setups that use touch detection (such as TouchPTV, Section 7.3.2, which cover large sections of a room). However, presence detection and foot interaction is generally more accurate over longer distances and

7.4 Discussion

provided a nice alternative to touch. Being able to trivially switch interaction modalities was also useful in the prototyping stage.

The effectiveness of an interaction modality can depend heavily on the context that it is used in. Encouragingly, *toolkit users* have already begun to determine measurements of what is acceptable in their own application domains [169]. This indicates that *toolkit users* are producing findings which can feed back into the more general interactive projected displays domain.

Lacking the native ability to combine multiple sensors prevents the toolkit being used to create physically large touch-enabled devices and application scenarios. Furthermore, recalibration is required from time-to-time if the Kinect is knocked or moved. These make it harder to manage longer-term installations. One work-around was to modify the `surface_zoffset` value. However, this process was too technical for many not familiar with JavaScript. To address this, an adaptive surface modelling system could be developed which would do this automatically. Another solution (suggested by a *toolkit user*) would be to apply an (optional) per-surface 8-point calibration.

7.4.6 Separation of Interaction Modality and Platform

The separation of interaction modality and underlying platform was a complex, but ultimately positive design decision. The application scenarios use a broad range of interaction modalities (touch, foot, presence, and even extensions to the touch algorithm to provide two sided touch and object detection). This feature was a particularly important factor in the versatility of the toolkit. However, as expected, its reliance on JavaScript also limited its performance. At least one external project has already begun separating the multi-touch detection from the toolkit⁷⁶.

7.4.7 Extensibility

In terms of the *Content API* extensibility the `Authority` class and handlers were seen as easy to modify and create, so that new features can be added. While they are

⁷⁶ These are the initial findings of an unpublished paper.

7.4 Discussion

limited to the local machine and difficult to apply to distributed systems, many other projects (i.e. the RED Framework [174]) focus on overcoming these issues from an application developer perspective.

To help make use of existing native applications, the `StartProcess` extension (Section 7.3.3) has a lot of potential. The streaming of `stdin/stdout` pipes is also an elegant solution of how to allow web content to control this process. However, the security risk makes it unsuitable for public release. Based on the analysis of the different types of toolkit users (Section 7.2.5.3) it would be prudent to include a ‘security options’ requirement for future revisions of the toolkit.

Reflecting on the nature of interoperation, the toolkit design limits interoperation with external systems by requiring them to obey the constraints of the toolkit (i.e. exist either as content or toolkit extensions). Another approach would be to integrate elements of the toolkit into other systems (i.e. separate the touch detection or projection mapping).

7.4.8 Performance

The toolkit system requirements (i.e. a mid-range i5 processor) were too high for it to be adopted by some user groups. There is a considerable variation of computer hardware of different ages, so reducing the system requirements is an important objective for future work. One corresponding toolkit user commented that for it to be commercially viable in their context, it would need to run on a netbook.

A very high proportion of the application scenarios involved using the touch interaction modality. Investing time in a more highly refined C# implementation is a valuable exercise to promote continued adoption.

The trade-off between performance and support for diverse application scenarios is a complex design choice. Although easy transfer to production [32] is a motivating factor, the toolkits design prioritised the range of diverse application scenarios. Given the motivation of the toolkit is to explore a range of new application scenarios, this decision would be repeated.

7.4 Discussion

7.4.9 Analysis Limitations

The analysis of adoption is inherently limited by an inability to characterise the data that is not captured. For instance, usage data from laptops deploying the toolkit on-location without an internet connection is lost. Subsequently, the analysis reflects a minimum level of use not a complete measure. Beyond simplistic indicators such as the number of downloads, it was difficult to extract insight from raw usage ‘counts’. However, aggregation by IP address and subsequent analysis of computed total and recurrent usage periods proved valuable in characterising different groups of *toolkit user*. Furthermore, it was much less likely to be misleading because it is examining a series of personal trends, rather than treating the entire user-group as one large trend⁷⁷. *Total* and *recurrent periods* are a better measure of *toolkit user* habits than raw usage counts, yet more work is required to be able to distinguish the experimentation phase from more regular use. With that mind, it may be possible to perform run-time analysis of the recurrent period in order to estimate if a person is likely to no longer be using the toolkit.

The usage statistics provide a strong quantitative overview of the what, where, and when. This relatively basic and non-invasive metric can reveal a lot of information. The manual analysis of personal correspondence and reporting through case-studies was an effective method of investigating the why and how of toolkit use. Most correspondents were of a pleasant disposition and were happy to share their impressions. While personal correspondence is generally of a high quality and accurate, there are a number of drawbacks. Firstly, many are not fluent English speakers. Given the worldwide adoption, translation is important. Secondly, a minority were not willing to share ideas. Thirdly, many did not document their work with pictures and video, so were unable to provide them when asked in order to featuring in this analysis. Providing an additional tool which helps people document their work as part of the main toolkit may help to capture information that is both accurate and valuable to the analysis.

⁷⁷ The latter approach is effective at extracting seasonal or globally common factors. For instance, a primary analysis of usage records and time of day (adjusted for time zone) accurately corresponded to typical daylight hours.

7.5 Chapter Summary

The community forum provided a useful way of showing off work, offering advice, and community support. The workload for managing the personal correspondence can get quite high (as many people require advice or examples) and so the community forum was a useful way of getting *toolkit users* to help one and other. The low level of research form use was disappointing. This was intended to capture demographic and project statistics, as well as the impressions of a range of toolkit users. Having an optional form is not effective. It would be worth exploring a non-optional feedback system integrated into the toolkit interface. Although it would have been possible to incorporate logging software, invasive data collection (i.e. Kinect video) was avoided due to the complex ethical and legal concerns of capturing live video from private spaces, even if explicit permission were granted. Furthermore, having a non-optional research form would interfere with the users work. It was for this reason that a modification was not made mid-release.

As only a single toolkit was observed and analysed, conclusions cannot be drawn regarding whether or not the quantitative data and usage statistics might apply to other projected displays toolkits as these have different designs, goals, and implementations. Further, the observation and community interactions were carried out by the developer of the toolkit which may introduce a degree of bias. This was minimised in the usage statistics analysis by using a consistent pro-forma (i.e. url hits) and constant reflection in the case studies and personal correspondence analysis to help reduce observer subjectivity.

7.5 Chapter Summary

The findings of this chapter are evidence that the toolkit supports user innovation with interactive projected displays. This evidence takes the form of a diverse set of application scenarios, use by a range of different *toolkit user* groups, case studies that generate applied academic knowledge, and usage statistics that describe adoption. These assert that the requirements, design features, and

7.5 Chapter Summary

implementation are effective whilst also identifying areas for improvement. These aspects are reflected upon in Chapter 8.

As of the 28th December 2013 the toolkit had been downloaded over 2,300 times and executed over 21,000 times by approximately 2,100 unique *toolkit users*. This adoption was distributed worldwide and typically fell into one of four high level categories (personal, corporate, academic, or computer science academic) and exhibited one of three usage patterns (curious, regular, or daily).

Analysis of 70 real-world application scenarios (captured through considered analysis of individual correspondence) provided a deeper look at common themes. A series of six case studies also provide spotlight detail. The proportion of exploratory and specific application scenarios was relatively consistent across all the types of user groups. However, the lack of HTML/JavaScript skills required to develop new content excluded a large number of would-be users. This was particularly noticeable amongst personal users with a specific application scenario in mind. However, many of the corresponding corporate and cs-academic users who were developing new systems (i.e a museum exhibit which would have previously used Flash/AS3) were willing to engage with other interaction modalities and new design ideas (i.e. the floor displays and enhancing the gesture detection capabilities). In terms of *toolkit user* expectations, the high interest in support for desktop applications indicates that familiarity with existing systems and existing applications is important to support the adopting demographic.

The discussion reflects on the adoption to identify the strengths and weaknesses of different toolkit features. It notes that while the motivation for adopting web standards was valid and demonstrably useful in a number of cases, interest from a wider user group than anticipated (i.e. non-programmers and corporate users) revealed a missed opportunity to support existing native desktop application content. However, there is a risk that adding this support would turn the toolkit into a complex interactive whiteboard application—obscuring many of the advanced features such as the Content API, multi-display design, responsive physical design and interchangeable interaction modalities. This might discourage the design of new content specifically for physical spaces. With that being said, Hyper Island's

7.5 Chapter Summary

interactive milk case study (Section 7.3.4) weakens the argument that content design is always an important and necessary part of application scenarios. The ability to create new and strange experiences that illustrate design possibilities may be just as important to the improvement of the technology as designing valuable ‘killer’ applications.

Pervasive interactive projected displays are still an emerging technology. As such, technological requirements, design challenges, and social impact are still being explored. Although a different toolkit may have yielded different results, as a case study, the adoption yields insights into the value of features that can inform future works. This chapter demonstrates that the toolkit as-is supported this process by engaging a diverse community of *toolkit users* and offering a design and feature set that enabled a range of different applied interactive projected displays.

Chapter 8. Conclusions

8.1 Thesis Summary

The question asked by this thesis is: *how can a toolkit effectively facilitate user innovation with interactive projected displays?* The answer was developed through three different research objectives: *exploration* of the toolkit scope through application driven research probes, *development* of a valuable toolkit design, and an *evaluation* to demonstrate effectiveness and learn from adoption. Figure 113 visualises this as a divergent and convergent process across the thesis chapters.

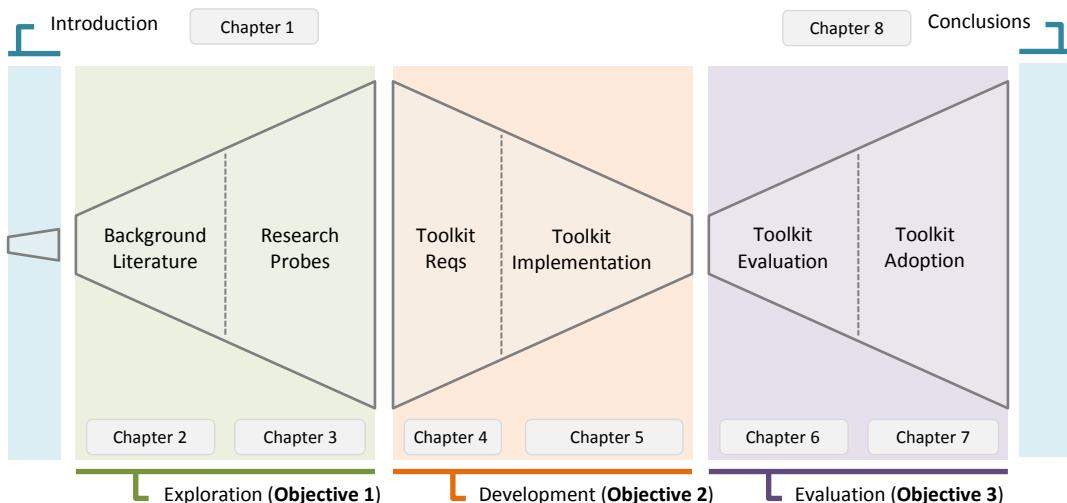


Figure 113: Overview of the thesis design process methodology. Each stage is mapped to thesis objectives and chapters.

The *exploration* objective serves to build a foundation of knowledge and experience constructing interactive projected displays. Chapter 2 examined existing literature and identified a set of implementation technologies, challenges, important characteristics, and existing toolkits. Chapter 3 expanded on this knowledge through two application-driven research probes. These yielded: (1) a deeper understanding of important display characteristics in application domains, (2) experience building

8.2 Contributions

and deploying interactive projected displays, and (3) research findings in targeted application domains enabled by the introduction of interactive projected displays.

The *development* objective converged on a set of toolkit requirements and built these into a working toolkit. Chapter 4 presented 22 requirements (alongside associated rationale) based on the findings of the previous two chapters. To help ensure that the requirements facilitate user innovation, they were framed around von Hippel's criteria for *toolkits that effectively support user innovation* [32]. Chapter 5 transformed the toolkit requirements into a single cohesive toolkit design. This was accompanied by a discussion of software architecture, implementation challenges, and contributions in the form of solutions to how they are addressed. Throughout these chapters, toolkit features can be traced back to the requirements and rationale that led to their inclusion.

Chapter 6 and Chapter 7 evaluated the toolkit through (1) technical experiments, (2) proof-of-concept deployments, (3) controlled user studies, (4) a long term case study, and (5) an in-depth analysis of adoption by the public user community. This included an analysis of 70 real-world application scenarios (captured through considered analysis of individual correspondence) provided insights into common application scenarios, themes, and important features. A series of six case studies also provided spotlight detail on specific projects. Combined, these chapters presented evidence that the toolkit effectively facilitated user innovation by engaging a diverse community of *toolkit users* and offering a feature set and interface that enabled them to develop a wide range of application scenarios.

8.2 Contributions

This thesis makes technical, conceptual, and applied contributions to the domain of interactive projected displays. Major thesis contributions are categorised below around the three research objectives:

8.2 Contributions

Exploration of interactive projected displays in application driven research in order to identify and converge on an appropriate scope and feature set:

- C1. A literature search captured influential work, projection and interaction technologies, content development, and existing toolkits (Chapter 2).
- C2. Two research probes that explored applied interactive projected displays (Chapter 3). These yielded insights into the practical challenges of developing applied interactive projected displays and concurrently made research contributions into each probes' application domain. Specifically:
 - a. The concept, design, implementation, and evaluation of an interactive projected display that can augment the co-located collaborative software engineering process for small teams (Section 3.3).
 - b. The implementation and longitudinal investigation of an interactive projected office desk (Section 3.4). Findings address long-term usability, interface considerations, as well as projected personalisation and decorations.

Development of a toolkit that simplifies and expedites the process of creating interactive projected displays:

- C3. A set of toolkit requirements structured around von Hippel's criteria for toolkits that support user innovation [32] (Chapter 4).
- C4. A software architecture and toolkit implementation that supports the identified requirements and integrates them into a cohesive design sensitive to the needs of the target user community (Chapter 5, Section 5.2).
- C5. The concepts and novel implementation of features including: physical responsive design, platform-agnostic interaction modalities, and a point-cloud based multi-touch detection algorithm to enable a wider range of hardware placements (Section 5.3).
- C6. Online support and discussion forums for a community of over 2,000 users that have downloaded and used the toolkit.

8.3 Discussion

Evaluation of the toolkit in terms of technical viability, suitability for adoption, valuable features, and analysis of in-the-wild adoption:

- C7. A technical assessment of the toolkit implementation and a profile of touch accuracy and performance (Section 6.2).
- C8. Applied deployments to ensure operational correctness (Section 6.3) and a user-study that address the toolkit's suitability for adoption and ability to support diverse application scenarios (Section 6.4).
- C9. An analysis of the volume and diversity of user innovation through quantitative longitudinal analysis and qualitative case-studies (Chapter 7). The quantity and quality of adoption and application scenarios explored by the toolkit users evidences the claim that the toolkit design effectively facilitated user innovation.

These contributions address the core research question "*how can a toolkit effectively facilitate user innovation with interactive projected displays?*" by dividing the work into exploratory, constructive, and evaluation stages. The design process followed and the methods used culminate in the production and adoption of a toolkit that facilitates user innovation. This generates practical value to the user community through adoption, and academic value in terms of design and evaluation grounded in real ecologically valid toolkit usage.

8.3 Discussion

The methodology that resulted in the thesis contributions was successful in that it produced a toolkit that was adopted and used. However, there are limitations that are discussed in this section.

8.3 Discussion

8.3.1 Effective Facilitation of User Innovation

This thesis is based on the assumption that facilitating user innovation is a valuable pursuit within the interactive projected displays domain (Section 1.2). As the findings of Chapter 7 demonstrate, facilitating user innovation has enabled a significant range of toolkit users to develop a diverse range of interactive projected displays for different purposes. However, it is difficult to objectively measure *how effectively* a toolkit facilitates user innovation in the short term. Buxton [176] argues that all modern technologies stand on decades of prior research, development, and experimentation—and that the bulk of innovation is low-amplitude and takes place over a long period of time. With that in mind many applications developed using toolkits are of incremental value or failures from which we can learn. The impact on the thesis research question is that evaluating a toolkit based on the volume of positive or negative outcomes, or the strength of successes of failures, does not necessarily reflect its ability to facilitate the overall innovation process [29]. It is rather through (a) simplifying and expediting the process of working with interactive projected displays, and (b) enabling a more diverse user-community to explore a wider set of application scenarios, that the thesis facilitates user-led idea refinement and augmentation that eventually result in ‘innovations’. The toolkit is left available to the community with the hope that it will pave the way for a new generation of interactive projected displays.

8.3.2 Analysing Adoption

To this point, the thesis has identified a set of features that are valuable to toolkits that facilitate user innovation with interactive projected displays. The adoption analysis characterises adopters and how indicates how they are supported by attributes and features of the toolkit. This helps to inform future work and new directions for the toolkit (i.e. considering popular themes in addition to areas that lacked support). Capturing information about adopter application scenarios was challenging because external successes and failures are not necessarily reported [16]. However, the digital support channels that were made available alongside the

8.3 Discussion

toolkit, such as internet forums, issue reports, feature requests via email, worked well but necessitated operating active and friendly support channels to cultivate insight. The workload required is a serious consideration for any similar toolkits, if a suitable self-sustaining support community cannot be built.

8.3.3 Single Data Point

A weakness of a toolkit-driven methodology is that it only generates a single data point that represents the toolkit implementation and its specific combination of features. To reduce the risk of this not being adopted (and thus not being able to report themes from diverse adoption) the methodology grounded the toolkit in experience gained from the research probes and structured requirements around von Hippel's criteria for toolkits that support user innovation [32]. Furthermore, the *exploration* objective specifically drew on multiple disciplines (i.e. ethnography, statistical analysis) to integrate different perspectives into the design. Although reasoning about different combinations of features is still fundamentally a speculative task, this approach increases the likelihood of generating a design with academic and practical value.

8.3.4 Alternative Methodologies

There are other methodologies, or variations on the chosen method, capable of exploring how toolkits can facilitate user innovation with interactive projected displays. For instance, studying the use of existing tools and developing a theoretical framework that assists with design goals, or decreasing the depth of research probes would allow time for a greater number. However, the chosen method is grounded in the facilitation of the user innovation process through practical contributions and an ecologically valid context to inform the design of future interactive projected display applications. The probes were of a similar complexity level to a number of the applied interactive projected displays described in Chapter 7. It is possible that this method generalises to other emerging domains (i.e. wearable computing). It is particularly well suited to research areas where complex implementation and

integration challenges prevent adoption by a wider audience, and is less suited to focused investigation of specific interaction issues.

8.4 Reflection

The process of building the toolkit, application driven interactive projections, and observing others do the same has led to a number of informal observations and insights that fall outside the rest of the analysis.

8.4.1 Limitations of Projection

Projection differs from other display technologies (CRT, LCD, TFT) because the presentation lacks physical constraints such as a fixed size, shape, or underlying material. Although these are powerful attributes, the cost of the projection, sensing, and computing hardware mean that applications need to add enough value to justify adoption in a commercial context. Although it is expected that exploration of these attributes will lead to new valuable application scenarios (e.g. Section 3.3), the limitations of projection hardware will continue to reduce the range of application scenarios that can be explored.

Poor contrast in high ambient light environments make deployment outdoors or in public spaces impractical. Lens focus can reduce the quality of content when projecting at an angle. Lower resolution than competing technologies also limits the range of application scenario to those that do not include small text. Furthermore, occlusion results in hidden visual content and reduces already sparse sensing data. This highlights the importance of considering projector and sensor placements that minimise occlusion for typical interaction positions. The toolkit addresses some of these issues by enabling the hardware placements that maximise the ability of the projection and sensing hardware separately (i.e. moving the projector closer to the surface whilst keeping the sensor at an occlusion minimising angle). However, this also increases the deployment complexity.

8.4 Reflection

8.4.2 Designing for Physical Spaces

The adoption analysis revealed that supporting existing desktop applications is important for certain toolkit user groups. However, good design practices for interactive projected display applications are not the same as those for desktop applications. Naïve support of desktop applications will not encourage people to think about the design of their applications, but will endorse evaluation of the new technology in terms of yesterday's tasks. While applications of interactive projected displays will certainly continue to be influenced by preceding technology, the best way for toolkits to facilitate this transition in design practice remains an open question.

The fast pace and rapidly changing requirements of the software industry mean that software designers typically design for the short-term (i.e. websites, apps). This is in contrast to the lifespan of physical devices such as chairs, desks, ovens, and beds that can be considerably longer. Thus building one into the other has the potential mismatch between the projected content and the object itself. This may mean that projection is better suited to being a programmable infrastructure or abstract overlay, than part of a fixed device.

When designing high-end consumer products like smart phones, attention to detail and use of high quality materials plays an important role. The same is true for household furniture such as desks and kitchen counter tops. Integrating interactive surfaces into these objects usually requires surface instrumentation (i.e. embedding a tablet PC). Projection is a very useful design tool here as it allows designers to choose from a broader array of materials. For instance, when parts of a desk are not in use, they are simply a wooden desk rather than an inert backlit touchscreen.

Respecting the aesthetics of existing physical objects (i.e. mapping projection to the borders of a floor tile, or wooden slats on a bed that control a light) can produce a more compelling user experience than the same functionality provided by a projected touch-sensitive button grid. When interacting with projected displays, the symbolic abstraction of the button does not appear to be as compelling as other design opportunities, such as actions triggered by picking up an object or touching a

8.4 Reflection

specific area of space. This is perhaps due to the lack of tactile feedback or audio provided by the surface.

8.4.3 Rapid Prototyping

Being able to draw out new display surfaces and reconfigure them without restarting the application greatly accelerated the rapid-prototyping process. This was particularly valuable when adapting applications to new environments (i.e. a brief deployment or demonstration in a shop or car showroom) or in projects that explore novel computing form factors (i.e. the two-sided transparent touch screen in Section 7.3.6). Similarly, drawing and dragging display surfaces on a live video of the physical space was an effective way for people to quickly describe the desired interactive surface geometries. However, different sensor rotations could result in counter-intuitive correspondences with cursor movement, and acute sensor-to-surface angles made drawing displays harder because fewer pixels correspond to larger movements. One way to avoid this in future revisions would be to provide a rotatable 3D model of the video environment. However, this comes at the cost of increased interface complexity.

When constructing interactive projected display applications, it was common to re-use display content that performed a basic function. These content items acted as pre-configured sensors or triggers that could push to, or be accessed by other display content based on its location. For example, `Presence Switch.html` would react to physical items placed upon it and call a `handleObjectPresent(surface, arguments)` function on a target surface. Swapping out the display content on the target surface with another content item that implemented the same function allowed the ad-hoc assembly of multi-display applications. Another dimension to this reusability is that the way a content item is deployed (i.e. size of deployment surface) can completely change the user experience, even though the application content is identical.

8.4.4 Tools and Extensibility

Toolkits serve as building blocks with which users can construct something useful or interesting, sometimes even surprising to the toolkit creators. While tools are empowering in this sense, design assumptions and simplifying abstractions inherently shape the things that are built with them. With early stage technologies, investing too much in one particular tool can create an intellectual gravity well where users will resist radical change because they have invested a lot in reaching some level of maturity or stability. While researchers are naturally mindful about such things, it is perhaps less clear where to draw the line between extending an existing toolkit and developing a new one entirely.

Much of what motivated people to trial the toolkit and ultimately adopt the technology for an application scenario was its ability to interoperate with existing systems that were of importance to them. At the moment the toolkit focuses on supporting other systems being integrated into it (i.e. web standards, web sockets, new interaction modalities, etc.), but another approach would be to design the toolkit so that elements could be integrated into other systems (i.e. touch detection on arbitrary surfaces). This would result in a more modular, but perhaps more complex toolkit design.

8.5 Future Work

Support deployments with multiple sensors and projectors: By increasing the number of sensors and projectors in a space it is possible to increase the visual coverage and overall tracking quality of interactions, including interactions in the spaces between surfaces. However, achieving this at scale remains an open challenge with many intersecting technical (i.e. how to achieve the required processing requirements) and social issues (i.e. how to ensure privacy is not compromised or that the most appropriate display surface for the content is selected). Since the submission of this thesis, projects such as Microsoft's

8.6 Conclusion

SurroundWeb [177] and BBC's Unconventional Screens [178] project are beginning to explore these issues in more depth.

Reduce hardware requirements: As it stands the processing and memory requirements of the toolkit are too large to easily support operation on embedded devices such as the Raspberry Pi⁷⁸. Further work would investigate methods and algorithms that improve the performance of high accuracy passive interaction sensing on low-cost and low-power computing devices. This would enable a new wave of ubiquitous interactive projected displays as it would reduce a major cost in the deployment process. Subsequent systems would draw on the findings of Chapter 7 in order to design effective remote management systems and identify target stakeholders.

Towards programmable physical spaces: In the computer games domain, scripting complex interactions for multiple users (characters) in rich virtual spaces is a well understood problem with mature and accessible development tools. In contrast, programming for physical spaces is still a challenging task [16] with many unresolved issues: limited a-priori knowledge of available devices, unattended operation, functional heterogeneity, ad-hoc architectures and increased volatility [179]. This thesis provides the Surface and Display abstractions as basic building blocks. Future work could consider additional layers of abstraction (such as those used in computer games) to simplify the development of programs for spaces such as the home, office, and factory.

8.6 Conclusion

Today, the primary use of projection technology is to create large flat displays used in entertainment, education, and digital signage. In research labs, interactive projected displays reach far beyond creating simple large flat interactive surfaces in order to produce and study entirely new computing experiences. However, few of

⁷⁸ Raspberry Pi computer: <http://www.raspberrypi.org/>

8.6 Conclusion

these have had commercial success or adoption beyond the labs that initially created them due to significant implementation and deployment complexities. By simplifying and expediting the process of building and deploying interactive projected displays, a larger and more diverse user group is able to build applications and engage with interactive projected display technology. As interactive projected displays are refined and the ideas they represent gain traction the research challenge changes from one of '*how to build*' to be one of '*what to build*'. If future physical environments can treat what we do as input, the design challenge lies in envisioning valuable and comfortable outputs. The complexity of this challenge is that uncovering value in such a large potential design space requires a multi-faceted and multi-disciplinary approach [16] [30]. Facilitating the process of discovering these outputs (the process of user innovation) is the main contribution of this thesis.

As a result of this thesis, over *two thousand* people have been able to experiment with or apply interactive projected displays to their own application scenarios. These people have diverse backgrounds, skill levels, and motivations. Applications can now be created and deployed by novices in hours rather than days. Widespread toolkit adoption beyond the computer-science academic community will continue to stimulate an exciting new set of interactive projected display applications that combine the functionality of computing with physical spaces.

References

- 1 Weiser, M. The Computer for the 21st Century. *Scientific American Special Issue on Communications, Computers, and Networks* (September 1991).
- 2 Ishii, H. and Ullmer, B. Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems* (Atlanta, Georgia, US 1997), ACM, 234-241.
- 3 Raskar, R., Welch, G., Cutts, M., Lake, A., Stesin, L., and Fuchs, H. The Office of the Future : A Unified Approach to Image-Based Modeling and Spatially Immersive Displays. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (Orlando, Florida, USA 1998), ACM.
- 4 Pinhanez, C. The Everywhere Displays Projector: A Device to Create Ubiquitous Graphical Interfaces. In *Ubicomp '01* (2001), ACM, 315-331.
- 5 Underkoffler, J., Ullmer, B., and Ishii, H. Emancipated pixels: real-world graphics in the luminous room. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA 1999), ACM Press/Addison-Wesley Publishing Co., 385--392.
- 6 Pinhanez, C., Kjeldsen, R., Levas, A., Pingali, G., Podlaseck, M., and Sukaviriya, N. Applications of Steerable Projector-Camera Systems. In *Proceedings of the IEEE International Workshop on Projector-Camera Systems at ICCV* (Nice, France 2003), IEEE Computer Society Press.
- 7 Molyneaux, D. and Kortuem, G. Ubiquitous Displays in Dynamic Environments: Issues and Opportunities. In *Workshop on Ubiquitous Display Environments (Ubicomp'04)* (2004).
- 8 Wilson, A. and Benko, H. Combining Multiple Depth Cameras and Projectors for Interactions On, Above, and Between Surfaces. In *Proceedings ACM Symposium on User Interface Software and Technology (UIST)* (New York 2010), ACM, 273-282.
- 9 Rekimoto, J. and Saitoh, M. Augmented surfaces: a spatially continuous work space for hybrid computing environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit* (Pittsburgh, Pennsylvania, United States 1999), 378--385.
- 10 Wellner, P. The DigitalDesk calculator: tangible manipulation on a desk top display. In *User Interface Software and Technology - UIST* (1991), 27-33.
- 11 Raskar, R., Welch, G., Low, K., and Bandyopadhyay, D. Shader Lamps. In *12th Eurographics Workshop on Rendering* (London, UK 2001).
- 12 Pinhanez, C. and Podlaseck, M. To Frame or Not to Frame: The Role and Design of Frameless Displays in Ubiquitous Applications. In *In Proceedings of Ubiquitous Computing (Ubicomp '05)* (Tokyo, Japan 2005), Springer-Verlag, 340-357.
- 13 Cotting, D. and Gross, M. Interactive environment-aware display bubbles. In *Proceedings of the 19th annual ACM symposium on User interface software and*

References

- technology* (Montreux, Switzerland 2006), ACM, 245-254.
- 14 Bogers, M., Afuah, A., and Bastian, B. Users as Innovators: A Review, Critique, and Future Research Directions. *Journal of Management*, 36, 4 (July 2010), 857-875.
 - 15 von Hippel, E. Lead Users: A Source of Novel Product Concepts. *Management science*, 32, 7 (July 1986), 791-805.
 - 16 Abowd, G. What next, Ubicomp? Celebrating an intellectual disappearing act. In *UbiComp'2012* (2012).
 - 17 Davies, N., Landay, J., Hudson, S., and Schmidt, A. Rapid Prototyping for Ubiquitous Computing. *IEEE Pervasive Computing*, 4, 4 (Oct-Dec 2005), 15-17.
 - 18 Davies, N. and Gellersen, H.W. Beyond prototypes: challenges in deploying ubiquitous systems. *Pervasive Computing, IEEE*, 1, 1 (Jan-March 2002), 26-35.
 - 19 Li, Y. *Rapid Prototyping of Ubiquitous Computing Applications: Tools & Frameworks*. University of Washington, Google Campus, Mountain View, USA., 2008.
 - 20 Zaharakis, I. and Komninos, A. Ubiquitous Computing - A Multidisciplinary Endeavour. *Latin America Transactions, IEEE*, 10, 3 (April 2012), 1850-1852.
 - 21 Hodges, S., Villar, N., Scott, J., and Schmidt, A. A New Era for Ubicomp Development. In *IEEE Pervasive Computing* (2012), 5-9.
 - 22 Lee, J. Hacking the Nintendo Wii Remote. In *IEEE Pervasive Computing* (2008), 39-45.
 - 23 Noble, J. *Programming Interactivity: A Designer's Guide to Processing, Arduino, and openFrameworks*. O'Reilly Media, 2009.
 - 24 Borkowski, S. Steerable Interfaces for Interactive Environment. *Institut National Polytechnique de Grenoble (INPG)* (2006).
 - 25 Pinhanez, C. Using a steerable projector and a camera to transform surfaces. In *CHI '01 extended abstracts on Human factors* (2001), ACM, 369-370.
 - 26 Kjeldsen, R., Levas, A., and Pinhanez, C. Dynamically Reconfigurable Vision-Based User Interfaces. In *Machine Vision and Applications* (2004), Springer-Verlag, 6-12.
 - 27 Xiao, R., Harrison, C., and Hudson, S. WorldKit: Rapid and Easy Creation of Ad-hoc Interactive Applications on Everyday Surfaces. In *The 31st Annual SIGCHI Conference on Human Factors in Computing Systems* (Paris, France 2013), ACM, 879-888.
 - 28 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.
 - 29 Rogers, E. *Diffusion of innovations (5th edition)*. Free Press., New York, USA, 2003.
 - 30 von Hippel, E. Democratizing innovation: The evolving phenomenon of user innovation. *Journal für Betriebswirtschaft*, 55, 1 (March 2005), 63-78.
 - 31 Tuomi, I. *Networks of Innovation: Change and Meaning in the Age of the Internet*. Oxford University Press, 2006.

References

- 32 von Hippel, E. Perspective: User Toolkits for Innovation. *Journal of Product Innovation Management*, 18, 4 (July 2001), 247-257.
- 33 THE DESIGN COUNCIL. *Eleven lessons: managing design in eleven global brands*. Design Council, London, UK, 2007.
- 34 Best, K. *Design Management: Managing Design Strategy, Process and Implementation*. AVA Publishing, 2006.
- 35 Harrison, C., Benko, H., and Wilson, A. OmniTouch: wearable multitouch interaction everywhere. In *UIST'2011* (2011), 441-450.
- 36 Lee, J., Dietz, P., Maynes-Aminzade, D., Raskar, R., and Hudson, S. Automatic projector calibration with embedded light sensors. In *Proceedings of the 17th annual ACM symposium on User interface software and technology* (Santa Fe, NM, USA 2004), ACM, 123-126.
- 37 Wilson, A. Using a Depth Camera as a Touch Sensor. In *ACM International Conference on Interactive Tabletops and Surfaces* (Saarbrücken, Germany 2010), ACM.
- 38 Raskar, R., Brown, M., Yang, R. et al. Multi-projector displays using camera-based registration. In *Proceedings of the IEEE Visualization* (San Francisco, CA, USA 1999), 161-168.
- 39 Bimber, O. and Raskar, R. *Spatial augmented reality: Merging real and virtual worlds*. A. K. Peters, 2005. Available Online: <http://www.uni-weimar.de/medien/ar/SpatialAR>.
- 40 Letessier, J. and Bérard, F. Visual tracking of bare fingers for interactive surfaces. In *Proceedings of the 17th annual ACM symposium on User interface software and technology* (Santa Fe, NM, USA 2004), ACM, 119-122.
- 41 O'Hara, K. *Public and situated displays: Social and interactional aspects of shared display technologies*. Springer, 2003.
- 42 Newman, J., Bornik, A., Pustka, D., Echtler, F., Huber, M., Schmalstieg, D., and Klinker, G. Tracking for Distributed Mixed Reality Environments. In *VR 2007 Workshop on Trends and Issues in Tracking for Virtual Environments* (Charlotte, NC, USA 2007), IEEE.
- 43 Hausler, H. *Media Façades: History, Technology and Media Content*. AVEdition, 2009.
- 44 Dalsgaard, P. and Halskov, K. Designing urban media façades: cases and challenges. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA 2010), ACM, 2277-2286.
- 45 Wiethoff, A. and Gehring, S. Designing Interaction with Media Façades: A Case Study. In *Proceedings of the Designing Interactive Systems Conference* (Newcastle, UK 2012), ACM, 308-317.
- 46 Storz, O., Friday, A., Davies, N., Finney, J., Sas, C., and Sheridan, S. Public Ubiquitous Computing Systems: Lessons from the e-Campus Display Deployments. *IEEE Pervasive Computing*, 5, 3, 40-47.
- 47 Willis, K., Poupyrev, I., Hudson, S., and Mahler, M. SideBySide: ad-hoc multi-user

References

- interaction with handheld projectors. In *UIST'2011* (2011), 431-440.
- 48 Mistry, P., Maes, P., and Chang, L. WUW - wear Ur world: a wearable gestural interface. In *CHI'09* (2009), 4111-4116.
- 49 Weiser, M. *Ubiquitous Computing Homepage*. PARC, 1996.
- 50 Wigdor, D., Penn, G., Ryall, K., Esenther, A., and Shen, C. Living with a Tabletop: Analysis and Observations of Long Term Office Use of a Multi-Touch Table. In *Workshop on Horizontal Interactive Human-Computer Systems* (2007), 60-67.
- 51 Morris, M., Brush, A.J.B., and Meyers, B. A field study of knowledge workers' use of interactive horizontal displays. In *Workshop on Horizontal Interactive Human-Computer Systems - TABLETOP* (2008), 105-112.
- 52 Hardy, J. Experiences: A Year in the Life of an Interactive Desk. In *In Proceedings of DIS 2012* (2012), ACM, 679-688.
- 53 Hornecker, E. "I don't understand it either, but it is cool" - visitor interactions with a multi-touch table in a museum. In *3rd IEEE International Workshop on Horizontal Interactive Human Computer Systems* (Amsterdam 2008), 113-120.
- 54 Åkerman, P., Puikkonen, A., Huuskonen, P., Virolainen, A., and Häkkilä, J. Sketching with strangers: in the wild study of ad hoc social communication by drawing. In *Proceedings of the 12th ACM international conference on Ubiquitous computing* (Copenhagen, Denmark 2010), 193-202.
- 55 Kruger, R., Carpendale, S., Scott, S., and Greenberg, S. How people use orientation on tables: comprehension, coordination and communication. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work* (Sanibel Island, Florida, USA 2003), 369-378.
- 56 Ryall, K., Forlines, C., Shen, C., and Ringel-Morris, M. Exploring the Effects of Group Size and Table Size on Interactions with Tabletop Shared-Display Groupware. In *Proceedings of CSCW* (2004), 284-293.
- 57 Norman, D. *The Psychology of Everyday Things*. Basic Books, 1988.
- 58 Welch, G., Fuchs, H., Raskar, R., Towles, H., and Brown, M.S. Projected imagery in your "office of the future". *Computer Graphics and Applications, IEEE* , 20, 4 (July-Aug 2000).
- 59 Underkoffler, J. and Ishii, H. Urp: a luminous-tangible workbench for urban planning and design. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit* (Pittsburgh, Pennsylvania, United States 1999), ACM, 386--393.
- 60 Sukaviriya, N., Podlaseck, M., Kjeldsen, R., Levas, A., Pingali, G., and Pinhanez, C. Augmenting a Retail Environment Using Steerable Interactive Displays. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems* (Fort Lauderdale, Florida, USA 2003), ACM, 978-979.
- 61 Sukaviriya, Noi , Podlaseck, Mark , Kjeldsen, Rick, Levas, Anthony , Pingali, Gopal , and Pinhanez, Claudio. Embedding Interactions in a Retail Store Environment: The Design and Lessons Learned. In *Ninth IFIP International Conference on Human-Computer Interaction* (Zurich, Switzerland. 2003).

References

- 62 Lai, J., Levas, A., Chou, P., Pinhanez, C., and Viveros, M. BlueSpace: Personalizing Workspace through Awareness and Adaptability. *International Journal of Human Computer Studies*, 57, 5 (2002).
- 63 Hornbeck, L. Digital Light Processing and MEMS: Timely Convergence for a Bright Future. In *Proceedings SPIE* (1995).
- 64 Dachselt, R., Häkkilä, J., Jones, M., Löchtefeld, M., Ml., Rohs., and Rukzio, E. Pico projectors: firefly or bright future? *ACM Interactions*, 19, 2 (March + April 2012), 24-29.
- 65 Raskar, R., Welch, G., and Fuchs, H. Spatially Augmented Reality. In *First International Workshop on Augmented Reality* (San Francisco CA 1998), IEEE.
- 66 Molyneaux, D. *Smart Object, not Smart Environment: Cooperative Augmentation of Smart Objects Using Projector-Camera Systems*. Lancaster University, Lancaster, UK, 2008.
- 67 Wren, C. *Perspective Transform Estimation*. MIT Media Lab, 1998.
- 68 Sukthankar, R., Stockton, R., and Mullin, M. Smarter Presentations: Exploiting Homography in Camera-Projector Systems. In *International Conference on Computer Vision* (Vancouver, Canada 2001), IEEE, 247-253.
- 69 Lee, J., Hudson, S., and Tse, E. Foldable Interactive Displays. In *Proceedings of the 21st annual ACM symposium on User interface software and technology* (Monterey, CA, USA 2008), ACM.
- 70 Park, H., Lee, M., Kim, S., and Park, J. Surface-Independent Direct-Projected Augmented Reality. In *Proceedings of the 7th Asian conference on Computer Vision* (2006), 892-901.
- 71 Park, H., Lee, M., Seo, B., and Park, J. Undistorted Projection onto Dynamic Surface. In *Proceedings of the First Pacific Rim conference on Advances in Image and Video Technology* (2006), ACM, 582-590.
- 72 *Stationary Observation System for High-speed Flying Objects*. University of Tokyo, Tokyo, Japan, 2013.
- 73 Summet, J., Flagg, M., Cham, T., Rehg, J.M., and Sukthankar, R. Shadow Elimination and Blinding Light Suppression for Interactive Projected Displays. *IEEE Transactions on Visualization and Computer Graphics* (May-June 2007), 508-517.
- 74 Raskar, R., van Baar, J., and Chai, J. A Low-Cost Projector Mosaic with Fast Registration. In *Asian Conference on Computer Vision* (2002).
- 75 Ashdown, M. *Personal projected displays*. University of Cambridge, 2004.
- 76 Tuddenham, P. and Robinson, P. Rapid prototyping of high-resolution and mixed-presence tabletop applications. In *Proceedings of the IEEE TableTop* (Newport, Rhode Island, USA 2007), IEEE.
- 77 Stone, M. Color and brightness appearance issues in tiled displays. *Computer Graphics and Applications*, 21, 5 (2001), 58-66.
- 78 Majumder, A. and Stevens, R. Lam: Luminance attenuation map for photometric uniformity in projection based displays. In *Proceedings of the ACM virtual reality*

References

- and software technology* (2002), ACM.
- 79 Bimber, O., Emmerling, A., and Klemmer, T. Embedded entertainment with smart projectors. *IEEE Computer*, 38, 1 (Jan 2005).
- 80 Grundhöfer, A. and Bimber, O. Real-Time Adaptive Radiometric Compensation. *IEEE Transactions on Visualization & Computer Graphics*, 14, 1 (Jan-Feb 2008), 97-108.
- 81 Kjeldsen, R., Pinhanez, C., Pingali, G., Hartman, J., Levas, T., and Podlaseck, M. Interacting with Steerable Projected Displays. In *Proceedings of the 5th International Conference on Automatic Face and Gesture Recognition* (2002).
- 82 Marshall, J., Pridmore, T., Pound, M., Benford, S., and Koleva, B. Pressing the Flesh: Sensing Multiple Touch and Finger Pressure on Arbitrary Surfaces. In *Proceedings of Pervasive* (2008), 38-55.
- 83 Hilliges, O., Izadi, S., Wilson, A., Hodges, S., Garcia-Mendoza, A., and Butz, A. Interactions in the air: adding further depth to interactive tabletops. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology* (Victoria, BC, Canada 2009), ACM, 139-148.
- 84 Klompmaker, F., Nebe, K., and Fast, A. dSensingNI: a framework for advanced tangible interaction using a depth camera. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction* (Kingston, Ontario, Canada 2012), 217-224.
- 85 Dippon, A. and Klinker, G. KinectTouch: Accuracy Test for a Very Low-Cost 2.5D Multitouch Tracking System. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* (Kobe, Japan 2011), ACm.
- 86 Echtler, F. and Klinker, G. A multitouch software architecture. In *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges* (2008), ACM, 463-466.
- 87 Fails, J. and Olsen, D. Light Widgets: Interacting in Every-day Spaces. In *Proceedings of the 7th international conference on Intelligent user interfaces* (2002), ACM, 63-69.
- 88 Potter, R., Weldon, L., and Schneiderman, B. Improving the accuracy of touch screens: an experimental evaluation of three strategies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Washington, D.C., USA 1988), ACm, 27-32.
- 89 Albinsson, P. and Zhai, S. High precision touch screen interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Ft. Lauderdale, Florida, USA 2003), AC, 105-112.
- 90 Benko, H., Wilson, A., and Baudisch, P. Precise selection techniques for multi-touch screens. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada 2006), ACM, 1263-1272.
- 91 Ng, A., Lepinski, J., Wigdor, D., and Sanders, S. Designing for low-latency direct-touch input. In *Proceedings of the 25th annual ACM symposium on User interface software and technology* (Cambridge, Massachusetts, USA 2012), ACM, 453-464.
- 92 Lee, S. and Zhai, S. The performance of touch screen soft buttons. In *Proceedings*

References

- of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA 2009), ACM, 309-318.
- 93 Shotton, J., Fitzgibbon, A., Cook, M. et al. Real-Time Human Pose Recognition in Parts from a Single Depth Image. In *Computer Vision and Pattern Recognition* (2011), IEEE.
 - 94 Bolt, R. Put-that-there: Voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques* (Seattle, Washington, USA 1980), ACM, 262-270.
 - 95 Krueger, M., Gionfriddo, T., and Hinrichsen, K. VIDEOPLACE—an artificial reality. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Francisco, California, USA 1985), ACM, 35-40.
 - 96 Hardy, J., Rukzio, E., and Davies, N. Real World Responses to Gesture Based Public Displays. In *Proc' MUM 2011* (2011).
 - 97 Lowe, D. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60, 2 (2004), 91-110.
 - 98 Bay, H., Tuytelaars, T., and Van Gool, L. Surf: Speeded up robust features. In *European Conference on Computer Vision* (2006), IEEE.
 - 99 Huber, J., Steimle, J., Liao, C., Liu, Q., and Mühlhäuser, M. LightBeam: nomadic pico projector interaction with real world objects. In *CHI 2012 Extended Abstracts on Human Factors in Computing Systems* (Austin, TX, USA 2012), ACM, 2513-2518.
 - 100 Ziola, R., Grampurohit, S., Landes, N., Fogarty, J., and Harrison, B. Examining interaction with general-purpose object recognition in LEGO OASIS. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on* (Pittsburgh, PA 2011), IEEE, 65 - 68.
 - 101 Ziola, R., Grampurohit, S., Landes, N., Fogarty, J., and Harrison, B. OASIS: Creating Smart Objects with Dynamic Digital Behaviors. In *Interacting with Smart Objects* (Palo Alto, USA 2011).
 - 102 Patten, J. and Ishii, H. Mechanical constraints as computational constraints in tabletop tangible interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA 2007), ACM, 809-818.
 - 103 Boring, S., Baur, D., Butz, A., Gustafson, S., and Baudisch, P. Touch projector: mobile interaction through video. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA 2010), ACM, 2287-2296.
 - 104 Schmidt, D., Chehimi, F., Rukzio, E., and Gellersen, H. PhoneTouch: A technique for direct phone interaction on surfaces. In *Proceedings of UIST 2010* (2010), ACM, 13-16.
 - 105 Davies, N., Friday, A., Newman, P., Rutledge, S., and Storz, O. Using bluetooth device names to support interaction in smart environments. In *Proc' MobiSys'09* (2009), 151-164.
 - 106 Vogel, D. and Balakrishnan, R. Interactive public ambient displays: transitioning from implicit to explicit, public to personal, interaction with multiple users. In *Proc' UIST 2004* (2004), ACM, 137 - 146.

References

- 107 Müller, J., Walter, R., Bailly, G., Nischt, M., and Alt, F. Looking Glass: A Field Study on Noticing Interactivity of a Shop Window. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA 2012), ACM.
- 108 Want, R., Hopper, A., Falcão, V., and Gibbons, J. The active badge location system. *ACM Trans. Inf. Syst.*, 10, 1 (1992), 91-102.
- 109 Greenberg, S., Marquardt, N., Ballendat, T., Diaz-Marino, R., and Wang, M. Proxemic interactions: the new ubicomp? *Interactions*, 18, 1 (January and February 2011), 42-50.
- 110 Burrell Saward, T. and Williams, D. *Display Cabinet*. 2011.
- 111 Pinhanez, C. Creating Ubiquitous Interactive Games Using Everywhere Displays Projectors. In *Proc. of the International Workshop on Entertainment Computing (IWEC'02)* (Makuhari, Japan 2002).
- 112 Heidrich, F., Golod, I., Russell, P., and Ziefle, M. Device-free interaction in smart domestic environments. In *Proceedings of the 4th Augmented Human International Conference* (Stuttgart, Germany 2013), ACM, 65-68.
- 113 Müller, Jörg, Wilmsmann, Dennis, Exeler, Juliane, Buzeck, Markus, Schmidt, Albrecht, Jay, Tim, and Krüger, Antonio. Display Blindness: The Effect of Expectations on Attention towards Digital Signage. In Tokuda, Hideyuki et al., eds., *Pervasive Computing*. Springer Berlin / Heidelberg, 2009.
- 114 Huang, E.M., Koster, A., and Borchers, J. Overcoming Assumptions and Uncovering Practices: When Does the Public Really Look at Public Displays? *Pervasive Computing*, 5013/2008 (2008), 228-243.
- 115 Xiao, X. and Ishii, H. *Perpetual Canon*. Tangible Media Group at MIT Media Lab, 2013.
- 116 Khalilbeigi, M., Lissermann, R., Kleine, W., and Steimle, J. FoldMe: interacting with double-sided foldable displays. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction* (Kingston, Ontario, Canada 2012), ACM, 33-40.
- 117 Frain, B. *Responsive Web Design with HTML5 and CSS3*. Packt Publishing, 2012.
- 118 Molyneaux, D., Izadi, S., Kim, D., and al., et. Interactive Environment-Aware Handheld Projectors. In *Pervasive'2012* (2012).
- 119 Weigel, M., Boring, S., Steimel, J., Marquardt, N., Greenberg, S., and Tang, A. *ProjectorKit: Easing the Development of Interactive Applications for Mobile Projectors*. University of Calgary, Alberta, Canada, 2013.
- 120 Schepers, D., Moon, S., and Brubeck, M. *Touch Events Specification*. 2011.
- 121 Kaltenbrunner, M., Bovermann, T., Bencina, R., Costanza, E. TUO - A Protocol for Table-Top Tangible User Interfaces. In *GW'05* (2005).
- 122 ROCKWELL GROUP. *Space Brew Homepae*.
- 123 ROCKWELL GROUP. *Open TSPS (Toolkit for Sensing People in Spaces) Homepage*.
- 124 OPEN FRAMEWORKS. *Open Frameworks Toolkit*.
- 125 THE PROCESSING FOUNDATION. *Processing.js Homepage*.

References

- 126 CYCLING '74. *Max Homepage*.
- 127 Butterworth, T. and Marini, A. *Syphon Homepage*.
- 128 Bellucci, A., Malizia, A., and Aedo, I. TESIS: turn every surface into an interactive surface. In *Proceedings of the 2011 ACM International Conference on Interactive Tabletops and Surfaces* (2011), ACM.
- 129 Hardy, J., Bull, C., Kotonya, G., and Whittle, J. Digitally annexing desk space for software development: NIER track. In *Proceeding of the 33rd international conference on Software engineering* (2011), ACM, 812--815.
- 130 van Deursen, A. Mesbah, A., Cornelissen, B., Zaidman, A., Pinzger, M., and Guzzi, A. Adinda: a knowledgeable, browser-based IDE. In *ICSE '10 Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2* (2010), ACM, 203-2076.
- 131 Bragdon, A., Reiss, S. P., Zeleznik, R. et al. Code Bubbles: Rethinking the User Interface Paradigm of Integrated Development Environments. *Proceedings of the 32nd International Conference on Software Engineering* (2010), 445-464.
- 132 DeLine, R., Czerwinski, M., Meyers, B., Venolia, G., Drucker, S., and Robertson, G. Code Thumbnails: Using Spatial Memory to Navigate Source Code (2006), 11-18.
- 133 DeLine, R. and Rowan, K. Code Canvas: Zooming towards Better Development Environments. In *Proceedings of the International Conference on Software Engineering (New Ideas and Emerging Results)* (2010), ACM, 207-210.
- 134 Bragdon, A., Zeleznik, R., Reiss, S. P. et al. Code Bubbles: A Working Set-based Interface for Code Understanding and Maintenance. In *In CHI '10: Proceedings of the 28th international conference on Human factors in computing systems* (2010), 2503-2512.
- 135 Ko, A.J., Aung, H.H., and Myers, B.A. Eliciting Design Requirements for Maintenance-Oriented IDEs: A Detailed Study of Corrective and Perfective Maintenance Tasks. *Proc. International Conference on Software Engineering* (2005).
- 136 Star, S. and Grieseme, J. Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-1939. *Social Studies of Science*, 19, 3 (August 1989), 387-420.
- 137 Snygg, D., Combs, A. Individual behaviour: a new frame of reference for psychology. *Journal of Consulting Psychology*, 13, 4 (1949).
- 138 Teasley, S. D., Covi, L. A., Krishnan, M. S., and Olson, J. S. Rapid Software Development through Team Collocation. *IEEE Transactions on Software Engineering*, 28, 7 (July 2002), 671-683.
- 139 Need, D. *Mitigating Airspace Issues In WPF Applications*. Microsoft MSDN Blogs, 2013.
- 140 Chidamber, S.R and Kemerer, C.F. A metrics Suite for Object Orientated Design. *IEEE Transactions on Software Engineering*, 20, 6 (June 1994).
- 141 Wise, A. *Augmented Reality: Inspiring the Next Generation*. Manchester University, Manchester, 2010.

References

- 142 Hardy, J. Reflections: a year spent with an interactive desk. *ACM Interactions*, 19, 6 (Nov-Dec 2012), 56-61.
- 143 Lie, M. and Sørensen, K. *Making technology our own? Domesticating technology into everyday life*. Scandinavian University Press, Oslo, 1996.
- 144 Heikkinen, H.L.T., Huttunen, R., and Kakkori, L. 'And this story is true'... on the problem of narrative truth. In *A paper presented at the European Conference on Educational Research* (University of Edinburgh 2000).
- 145 Sandberg, J. How Do We Justify Knowledge Produced Within Interpretive Approaches? *Organizational Research Methods*, 8, 1 (January 2005), 41-68.
- 146 Wallace, J.R. and Scott, S.D. Contextual design considerations for co-located, collaborative tables. In *Workshop on Horizontal Interactive Human-Computer Systems - TABLETOP* (2008), 57-64.
- 147 Scott, S.D., Grant, K.D., and Mandryk, R.L. System Guidelines for Co-located, Collaborative Work on a Tabletop Display. In *European Conference on Computer Supported Cooperative Work - ECSCW* (2003), 159-178.
- 148 Grudin, J. Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. In *Computer Human Interaction - CHI* (2001), 458-465.
- 149 Benko, H., Morris, M., Brush, A.J. B., and Wilson, A. *Insights on interactive tabletops: A survey of Researchers and Developers*. Microsoft Research, 2009.
- 150 Wimmer, R., Hennecke, F., Schulz, F., Boring, S., Butz, A., and Hußmann, H. Curve: Revisiting the Digital Desk. In *NordiCHI* (2010), 561-570.
- 151 Kurniawan, S. and Zaphiris, P. Reading Online or on Paper: Which is Faster? In *Abridged Proceedings of the 9th International Conference on Human Computer Interaction* (2001).
- 152 Nielsen, J. Electronic Books – A Bad Idea. ([Online] <http://www.useit.com/alertbox/980726.html> 1998).
- 153 Schmidt, D. Know Thy Toucher. In *In CHI 2009 Workshop: Multitouch and Surface Computing* (Boston, USA 2009).
- 154 Ringel, M. When One Isn't Enough: An Analysis of Virtual Desktop Usage Strategies and Their Implications for Design. In *Proceedings of CHI'03 Extended Abstracts* (2003), 762-763.
- 155 Marcus, A. Graphic Design for User Interfaces. In *SIGGRAPH 93 tutorial notes* (1993).
- 156 Kirsh, D. and & Maglio, P. On Distinguishing Epistemic from Pragmatic Action. *Cognitive Science: A Multidisciplinary Journal*, 18, 4 (1994), 513-549.
- 157 Nakatani, L. and Rohrlich, J. Soft Machines: A Philosophy of User-Computer Interface Design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, Massachusetts, USA 1983), ACM, 19-23.
- 158 Lui, W., Stappers, P.J., Pasman, G., and van der Helm, A. Demonstrating Generation Y Interactions through Interactive Prototyping. In *UbiComp* (2011), ACM.
- 159 MAKE. *Maker Faire Bay Area*. Make:, San Mateo, CA, USA, 2012.

References

- 160 Kuznetsov, S. and Paulos, E. Rise of the Expert Amateur: DIY Projects, Communities, and Cultures. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries* (Reykjavik, Iceland 2010), ACM, 295-304.
- 161 Franke, N. and von Hippel, E. Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software. *Research Policy*, 32, 7 (July 2003), 1199-1215.
- 162 Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.
- 163 Dix, A., Abowd, G., Beale, R., and Finlay, J. *Human-Computer Interaction*. Prentice Hall, Europe, 1998.
- 164 Toussaint, G. Solving geometric problems with the rotating calipers. In *MELECON'83* (1983).
- 165 Bentley, J.L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18, 9 (1975), 509-517.
- 166 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.
- 167 Buryanov, A. and Kotiuk, V. Proportions of Hand Segments. *International Journal of Morphology*, 28, 3 (Sep. 2010), 755-758.
- 168 Wolf, D. *TouchPTV: Leveraging Television Experience through Projected Touch Interaction*. University of Ulm, Ulm, Germany, 2013.
- 169 Osswald, K. *Novel Applications for Gesture-Based Interaction with Entertainment Systems—Design, Implementation and Evaluation*. University of Ulm, Ulm, Germany, 2013.
- 170 Leutenegger, S., Chli, M., and Siegwart, R. Y. BRISK: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV)* (2011), IEEE, 2548-2555.
- 171 Funk, M., Korn, O., and Schmidt, A. An augmented workplace for enabling user-defined tangibles. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems* (Toronto, Ontario, Canada 2014), ACM, 1285-1290.
- 172 Matoba, Y., Takahashi, Y., Tokui, T., Phuong, S., and Koike, H. AquaTop Display. In *Proceedings of the Virtual Reality International Conference: Laval Virtual* (Laval, France 2013).
- 173 Almeida, R., Blackstock, M., Lea, R., Calderon, R., Prado, A., and Guardia, H. Thing broker: a twitter for things. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication* (Zurich, Switzerland 2013), ACM, 1545-1554.
- 174 Calderon, R., Blackstock, M., Lea, R., Fels, S., and Almeida, R. Developing Cross-Display Applications Using the Really Easy Displays (RED) Framework. In *International Symposium on Pervasive Displays* (Mountain View, California 2013), ACM.
- 175 Bueno, A., Anacleto, J., Calderon, R., Fels, S., and Lea, R. ICT to support

References

- community gardening: a system to help people to connect to each other in real life. In *Proceedings of the 2014 Companion Publication on Designing Interactive Systems* (Vancouver, BC, Canada 2014), ACM, 133-136.
- 176 Buxton, B. The Long Nose of Innovation. *Bloomberg Businessweek Innovation and Design* (Jan 2008). <http://www.businessweek.com/stories/2008-01-02/the-long-nose-of-innovationbusinessweek-business-news-stock-market-and-financial-advice> (Online).
- 177 Vilk, J., Molnar, D., Ofek, E. et al. *SurroundWeb: Least Privilege for Immersive 'Web Rooms'*. Technical Report MSR-TR-2014-25, Microsoft Research, 2014.
- 178 BBC RESEARCH AND DEVELOPMENT. *Unconventional Screens: Exploring the potential of future display technologies*. BBC, Manchester, UK, 2014. <http://www.bbc.co.uk/rd/projects/unconventional-screens>.
- 179 Iftode, L., Borcea, C., Kochut, A., Intanagonwiwat, C., and Kremer, U. Programming computers embedded in the physical world. In *Distributed Computing Systems, 2003. FTDCS 2003* (2003), 78-85.
- 180 Bull, C., Cruickshank, L., and Whittle, J. Studios in software engineering education: towards an evaluable model. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA 2013), IEEE Press, 1063-1072.
- 181 Cao, X., Forlines, C., and Balakrishnan, R. Multi-user interaction using handheld projectors. In *Proceedings of the 20th annual ACM symposium on User interface software and technology* (2007), ACM, 43--52.